History (or) evolution (or) origins (or) Introduction of Java

* Java is object oriented prograraming language
* Java was developed by James Gosling at Sun microsystem in 1991.
* Java was initially called as oak (tree name) but it was renamed as Java in 1995
  Java is a seed name of coffee and it symbolically reprented as cup and saucer.
* primary motivation for developing Java language is to develop software for Electronic devices such as TVs, remote control and micro ovens etc
* secondary motivation is WWW [world wide web]
  learning Java langure is very simple

Editions of Java

1. J2SE [Standard Edition]
2. J2 EE [Enterprise Edition]
3. J2 ME [micro Edition]

Structure of Java program

The structure of Java program contains the following Elements.

* Documentation
* package declaration
* Import statement
* Interface
* class definition
* class variable
* Main method class
* Methods and behaviors.

Documentation

* The documetation section is an important section but optional for a Java program
* It includes basic information like authors name, company name, version, program name
* The comments in the Java program may be single line, multi line and documentation comments

single line comments

It starts with a pair of forwarding slash (//)

multi line comments

It starts with a /* and ends with */.

Documentation comment

It starts with the delimiter (/**) and ends with */.

package declaration

* The package declaration is optional. It is placed just after the documentation section.
* In this section we declare the package name
* We use the keyword package to declare the package name

Example package packagename;

Import statement

* The package contains the many predefined classes and interfaces
* If we want to use any class of a particular package we need to import that class
* We use the import keyword to import the class

Example import java.util.Scanner;
import java.util.*;

Interface Section

* It is an optional section. we can create an interface in this section if required.
* we can use interface in classes by using "implements" keyword.

Example

```
interface car
{
  void start();
  void stop();
}
```

Class definition

* It is important part of a java program. with out class we can not create any java program.
* Every java program has at least one class that contains the main() method

Example

```
class student
{
  }
```

Class variable and constants

we define variable and constants that are to be used later in the program.

Example

```
class student
{
  string Sname;
  int id;
}
```

Main Method class

It is essential for all java programs because the Execution of all java programs starts from the

main() method.

Example

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

Method and behavior

In this section we define the functionality of the program by using the methods.

Example

```
class Demo
{
public static void main (String[] args)
{
void display()
{
System.out.println (" welcome to Java");
}
}
}.
```

Writing Simple Java program

To create a simple Java program you need to create a class that contains the main method.

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

Save this file as Simple.java

To Compile – javac Simple.java

To Execute – java Simple.

Output

Hello to Java.

* class keyword is used to declare a class in Java.
* public keyword is an access modifier which represents visibility.
* static is a keyword. If we declare any method as static it is known as static method.
* void is the return type of method.
* main represents the starting point of program
* String[] args is used for command line arguments.
* System.out.println() is used to print statement.

Elements (or) tokens in Java

Java supports 5 types of tokens which are

1. keywords
2. Identifiers
3. Literals
4. operators
5. Special Symbols.

keywords

* keywords in Java are predefined or reserved words that have special meaning to the Java compiler.

* Each keyword is assigned a special task or function and can't be changed by the user.
* we can't use keywords as variables or identifiers
* A keyword always be written in lowercase because java is a case sensitive language.

Abstract

Abstract keyword is used to declare abstract class. abstract classes can implemented by using interface concept.

Boolean

This keyword is used to declare a variable of a boolean type [True and False]

Break

It break the current flow of the program at specified condition.

Byte

It is used to declare a variable that can hold 8-bit data values.

case

It is used in switch statement to mark blocks of text

catch

It used to catch the Exceptions generated by try
It must be used after the try block only.

char

used to declare a variable 16-bit characters.

continue

It continues the current flow of the program and skip the remaining code at specified condition.

default

To specify the default block of code in switch

DO

It can iterate a part of the program several times.

double

To declare a variable of 64-bit floating point numbers.

else

To indicate alternative choice in if statement.

enum

To define a fixed set of constants.

Extends

To indicate a class is derrived from another class or interface.

Final

To indicate that a variable holds a constant value. It is used to restrict the user.

Finally

It indicate a block of code in try-catch structure

Float

declare a variable of 32-bit floating point number

For

To Execute a set of instructions repeatedly when some conditions become true

if

If keyword tests the condition.

implements

It is used to implement an interface

import

It makes classes and interfaces available

instanceof

It is used to test whether the object is an instance of the specified class.

int

To declare a variable of 32-bit signed integer

interface

It is used to declare an interface.

long

used to declare a variable of 64-bit integer.

native

It is used to specify a method is implemented.

new

It is used to create new object.

null

It removes the garbage value

package

To declare a Java package that includes classes

private

It is a access modifier. A method or variable may be accessed only in the class.

protected

It is an access modifier. It can be accessible within package and outside the package but throug Inheritance only.

public

It is also an access modifier. It is used to access item anywhere

return

It used to return from a method when its execution is complete.

1-6

Short
It is used to declare a variable of 16-bit integer.

Static
It is used for memory management.

Strictfp
To restrict the floating point calculation.

Super
It can be used to invoke immediate parent class method.

Switch
To tests the equality of variable against multiple values

Synchronized
To specify the methods in multithreding.

this
used to refer the current object in a method.

throw
It is mainly used to throw custom Exception.

throws
It is used to declare an Exception

transient
This keyword is used to serialization

try
used to start a block of code that will be tested for Exceptions.

void

volatile
A variable may change asynchronously.

while
loop iterates a part of the program several times

Java identifiers are the user defined names of variable, methods, classes, arrays, packages and interfaces.

* Identifiers must begin with a letter, $ or _ (underscore)
* Identifiers in Java are case sensitive
* Java Identifier can be of any length.
* Identifiers name can't contain white space
* Any identifiers name must not begin with a digit

Example

valid identifiers

$myvariable
_variable
variable
edu_identifier_name
edu2021var

Invalid Identifiers

edu variable
$variable
23identifier
switch

Literals/constants

Literals represents a fixed value

Java provides five types of literals are as follows.

* Integer
* Floating point
* character
* String
* Boolean

Literal	Type
23	int
9.86	floating point
false, true	Boolean
'x', 'y', '_'	char
"Java"	string

operators

operators are the special symbols that tells the compiler to perform a special operation

There are eight types of operators in Java

1. Arithmetic operators [+, -, /, *, %]
2. Assignment operators [+=, +=, -=, *=, /=, %=]
3. Relational operators [==, !=, <, >, <=, >=]
4. Unary operators [++, --]
5. Logical operators [&& , ||]
6. Ternary operators [condition ? (State 1) : (State 2);
7. Bitwise operator [&, |, ~, ^]
8. Shift operators. [<<, >>, >>>]

5. Special Symbols

Special symbols are used in Java having some special meaning

* Brackets [ ] are used as array element reference.
* parentheses ( ) are used to indicate functions

* Braces { } curly braces are used start and end of a block of code containing more than one executable statements.
* Comma(,) is used to separate more than one statement
* astorick(*) is used to create pointer variable
* Semicolon(;) is used to initialization of list.

Java statement

Java supports three different types of statement

1. Expression statements

It can change values of variables, call methods and create objects

2. Declaration statements

These statements are declare the variable

3. Control flow statements

It determine the order of statements are Excuted.

Example

```
int number;        // declaration statement
number = 4;        // Expression statement
if (number < 10)   // Control flow statement
{ System.out.println(number + " is less than ten");
  }
```

```
System.out.println(number+" is less than ten");
}
```

Command line argument

The Java command line argument is an argument passed at the time of running the Java program so it provides a convenient way to check the behavior of the program for the different values i.e you can pass N (1,2,3,--) number of arguments from the command prompt.

Example

```
class A
{
 public static void main (String [] args)
    {
      for (int i=0; i<args.length; i++)
        System.out.println(args[i]);
    }
  }
```

compile by> javac A.java

run by> Java A Sai kumar 1 2 abc

output

```
Sai
kumar
1
2
abc
```

User input to programs

* The Scanner class is used to get user input and it is found in the java.util package
* To use the Scanner class, create the object of the class and use any of the available methods found in the scanner class documentation.

Example

```
import java.util.Scanner;
class Main {
 public static void main (String[] args)
 Scanner myobj = new Scanner (System.in);
 System.out.println ("Enter username");
 String userName = myobj.nextLine();
 System.out.println ("username is: "+ userName);
 }
}
```

Output

```
Enter username
 markapur
Username is: markapur.
```

Escape sequences comments

A character preceded by backslash (\) is an escape sequence and has a special meaning to the compiler.

\t — insert a tab in the text

\t — Insert a backspace in the text

\n — insert a newline in the text.

\f — insert a form feed in the text

\' — insert a single quote character

\" — insert a double quote character

\\ — insert a backslash character

Example

```
public class Test {
  public static void main (String[] args)
    {
      System.out.println (" Good morning "); or
    }                        ("Good morning \n How are you");
  }
```

Output

Good morning.

or

Good morning
How are you.

programming style

programming style refers to the technique used in writing the source code for a computer program.

Some general rules (or) guidelines of programming styles

1. clarity and simplicity of Expression.- The objectives of the program is clear.

2. Naming

In a program you are required to name the module, processes and variable

Example a = 3.14*r*r; (✓)

area of circle = 3.14 * radius * radius; (x)

3. Control constructs

It is desirable that as much as a possible single entry and single Exit.

4. Information hiding

The information secure in the data structure shoud be hidden from the rest of the system.

5. Nesting

It also becomes difficult to understand the program logic so it is desirable to avoid deep nesting.

6. User defined types

make use of user defined data types like class structure etc so your program code easy to write and easy to understand.

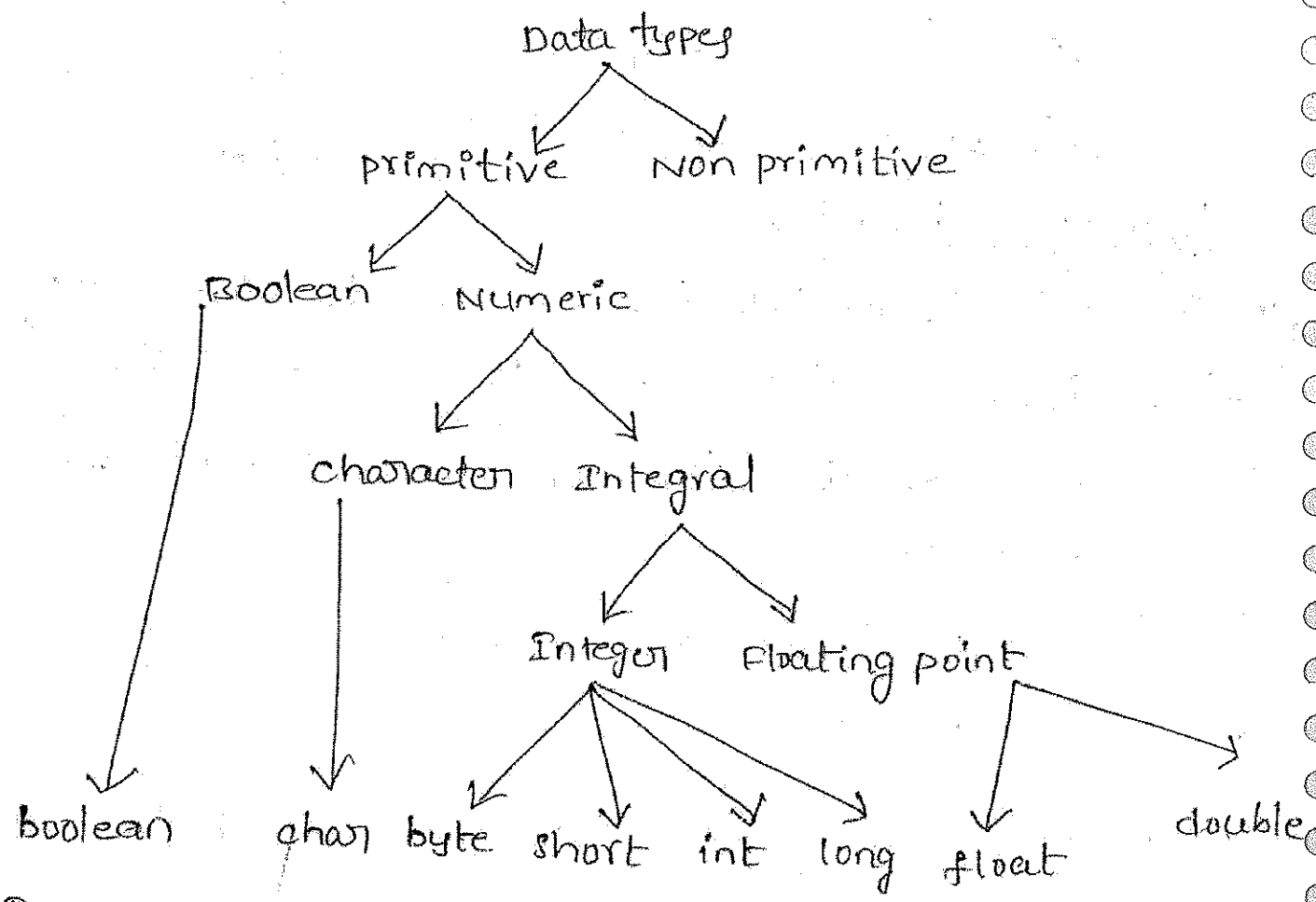
7. module size The module size should be uniform. the size of module should not be too big or too small

8. module interface

A module with a complex interface should be carefully examined.

Data types

* Data types specify different size and values that can be stored in the variable
* There are two types of data types in Java.



* In Java primitive data types are the building blocks of data manipulation.

Data type	Default value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Boolean data type

* The boolean data type is used to store only two possible values are true and false
* The boolean data type specifies one bit of information

Example Boolean one=false

Byte datatype

* Its value range lies between -128 to 127.

Its minimum value is -128 and maximum value is 127. Its default value is 0.

* The byte datatype is used to save memory in large arrays.

Example byte a=10, byte b=-20

Short data type

* Its value range lies between -32,768 to 32,767. Its minimum value is -32,768 and maximum value is 32767, default value is 0.

* Short data type can also used to save memory

Example short s=10000, short r=-5000

Int data type

* Its value lies between -2^{31} to $2^{31}-1$ and its minimum value is -2^{31} and maximum value is $2^{31}-1$, default value is 0.

* The int data type is generally used for a default data type for integral values

Example

int a=1,00,000

Long data type

Its values range between -2^{63} to $2^{63}-1$ and its minimum value is -2^{63} and maximum value is $2^{63}-1$, default value is 0.

* The long data type used when you need a range of values more than provided by int data type

Example long a = 1,0,0,000L.

Float data type

* Its value range is unlimited and its default value is 0.0F, It is singleprecision of 32 bit

Example float f1 = 235.4f

Double data type

* The double data type is a double precision 64 bit and its value range is unlimited.
* Its default value is 0.0d.

Example double d1 = 12.3

Char data types

* Its value range lies between '\u0000' to '\uffff'
* The char data type is used to store characters

Example char A = 'A'

Declaration of variables

variable is name of reserved area allocated in memory

There are three types of variables in Java.

1. local variable
2. instance variable
3. static variable

Local variable

* A variable declared inside the body of the method is called local variable
* A local variable can not defined with static keyword.

Instance variable

* A variable declared inside the class but outside the body of the method is called instance variable
* It is not declared as static.

Static variable

* A variable which is declared as static is called static variable
* It can not be local.
* memory allocation for static variable happens only once when the class is loaded in memory.

Example

```
class A {
    int data=50; // instance variable
    static int m = 100; // static variable
    void method() {
        int n=90; // local variable
    }
}
```

How to declare variables

```
int count;
↓   ↓
Type  name
```

type type of data that can be stored in variable

name Name given to the variable

How to initialize variables

```
int  age = 20; ← value
↓    ↓
Data type  variable name
```

Type casting

* Type casting is a process that converts a datatype into another datatype in both ways manually and automatically.
* The automatic conversion is done by the compiler and manually conversion done by the programmer.

There are two types of type casting.

1. Widening (or) implicit (or) casting down.
2. Narrowing (or) Explicit (or) casting up

Widening type casting

converting a lower data type into a higher one is called widening type casting.

* It is done automatically and it is safe because there is no chance to lose data

1-13

byte → short → char → int → long → float → double

Example

```
class Widening
{
  public static void main (String[] args)
  {
    int x=7;
    long y=x;
    float z=y;
System.out.println("before conversion, int value"+x);
System.out.println("After conversion, long value" +y);
System.out.println("After conversion, float value", +z);
  }
}
```

output

before conversion, ~~the~~ int value is: 7

after conversion, long value is: 7

after conversion, float value is: 7.0

Narrowing type casting

* converting higher data type into a lower one is called narrowing type casting.
* It is done by manually by the programmer.

double → float → long → int → char → short → byte

Example

```
class Narrowing.
{ public static void main (String[] args)
  {
    double d = 155.55;
    long l = (long)d;
    int i = (int) l;
  System.out.println (" before conversion " +d);
  System.out.println ("after conversion into long type:" + l);
System.out.println (" after conversion into int type:" + i);
  }
}
```

output

before conversion : 155.55

after conversion into long type: 155

after conversion into int type: 155

Literals constants

Literals are number, text or anything that represent a value (OR) Literals in Java are the constant values assigned to the variable.

types of Literals

1 Integral
2 Floating point
3 char
4 string
5 Boolean

1. Integral literals

We can specify the integer literals in 4 different way.

(i) Decimal (Base 10)

Digits from 0-9 are allowed in this form.

int x = 101;

(ii) octal (Base 8)

Digits from 0-7 are allowed. It should always have a prefix 0.

int x = 0146;

(iii) Hexa decimal (Base 16)

Digits 0-9 are allowed and also characters from a-f are allowed in this form.

int x = 0x123face;

(iv) Binary

A literal should have a prefix 0b and 0B and specify 0 and 1.

Example int x = 0b1111;

Floating point literals

Here datatypes can only specified in decimal form and not in octal or hexadecimal form.

Char Literals

There are four types of char literals

(i) Single Quote

char data type as a single character within a single quote

char ch = 'a';

(ii) char as integral

A char literal can specify as integral literal which also represents the ~~au~~ unicode value of a character.

char ch= 062;

(iii) unicode representation

char literals can specify in unicode representation

char ch= '\u0061';

(iv) Escape sequence

Escape sequence can also specify as char literal

char ch= '\n';

4. String literals

String literals are any sequence of characters with a double quote

String s= " Hello";

5. Boolean Literal

They allow only two values i.e true and false

boolean b = true;

For each loop

In java the for each loop is used to iterate through elements of arrays and collections

Syntax

```
for (datatype item: array)
  {

    }
```

Here array - is an collection of elements.
item - each item of array is assigned to this variable
datatype - which datatype of the array.

program

```
class Main
{
public static void main (String[] args)
  {
   int[] numbers = {3, 9, 5, -5};
   for (int number: numbers)
     {
     System.out.println(number);
     }
   }
  }
```

output

```
3
9
5
-5
```

Symbolic constants

constants may appear repeatedly in number of places in the program.

Syntax

final type symbolicname = value;

Example

```
float PI = 3.14159;
```

~~int~~

* constant values are assigned to some names at the beginning of the program.

Formatted output with printf() method

The printf() method consists of two types with parameter

Syntax

1. printf (String format, object args)
2. printf (local l, String format, object args)

system.out.print() and System.out.println() take a single argument but printf() may take multiple arguments.

Example program

```
class printtest
{public static void main (String[] args)
  {string name= " suresh";
   int age = 30;
   System.out.printf(or) format ("%-10S - %-4d\n" name,age);
  }}
```

output Suresh -30

abstract	assert	boolean	break	byte
case	catch	char	class	const
continue	default	do	double	else
enum	extends	final	finally	float
for	goto	if	implements	import
instance of	int	interface	long	native
new	package	private	protected	public
return	short	static	strictfp	super
switch	synchronized	this	throw	throws
transient	try	void	volatile	while

UNIT-I [R-19] JAVA (JP) 1-17

Literals

Literals are used to express certain fixed value within the source code of the program. Literals can be assigned to any primitive variable

Example

```
boolean flg = false;
```

Boolean is datatype
flg is variable
false is literal

1. Integer literals

Integer literal are used to represent a decimal, binary (or) hexadecimal value

Example

```
int decInt=18
int hexInt= 0X12
int binInt= 0b10010
```

In java binary literals starts with 0b and hexadecimal literals starts with 0x.

2. Floating-point literals

* A floating point literal is used to represent value for a float and double variable or constant
* A floating point literals ends with 'f' or 'F' otherwise its type of double.
* A double type can optionally end with 'D' (or) 'd' they can also expressed in scientific notation using E or e

Example

```
double mydouble = 3.4;
float myfloat = 3.4F;
double mydoublescientific = 3.445e2;
```

String & Character literals

* string literals are represented as a sequence of characters surrounded by double quotes and
* a character literal is represented as single character surrounded by single quotes
* java allows use of escape sequence in string and character literals

Example \b (backspace), \t (tab), \\ (backslash), \" (double quotes) \' (single quotes), etc

Example

```
char mychar = 'A';
String myString = "JAVA"
```

Boolean literely

Boolean literely are used to represent true and false value

Example

```
boolean flag = true;
```

operators

An operator is a special symbol that is used to carry out some specific operation on its operand.

1. Assignment
2. Arithmetic
3. Unary
4. Relational
5. Logical
6. conditional
7. Bitwise
8. instanceof
9. Dot(.)

Assignment

Assignment operators are used to assign value to a variable

Operator	Description	Expression
=	Assignment	a=b
+=	add and assign	a+=b ⇒ a=a+b.
-=	subtract and assign	a-=b ⇒ a=a-b.
*=	multiply and assign	a*=b ⇒ a=a*b
/=	divide and assign	a/=b ⇒ a=a/b
%=	mod and assign	a%=b ⇒ a=a%b
<<=	Left shift and assign	a<<=5 ⇒ a=a<<5
>>=	Right shift and assign	a>>=5 ⇒ a=a>>5
&=	Bitwise and assign	a&=5 ⇒ a=a&5
^=	Bitwise Exclusive OR and assign	a^=5 ⇒ a=a^5
|=	Bitwise inclusive OR and assign	a|=5 ⇒ a=a|5

Arithmetic

Arithmetic operators are used to perform arithmetic operations like addition, subtraction, multiplication division, modulus, Exponent etc

Operator	Description a=20, b=10	Expression
+	addition	a+b = 30
-	subtraction	a-b = 10
*	multiply	a*b = 200
/	division	a/b = 2
%	modulus	a%b = 0

Unary

In java ++ and -- are known as increment and decrement operators respectively.

Unary operators means they works on single operand

Operator	Example	Description
++ [prefix]	++a	value of after increment
++ [post fix]	a++	value of before increment
-- [prefix]	--a	value of after decrement
-- [post fix]	a--	value of before decrement

Comparision

Comparision operators are used evalete a comparision between two operands. The result of a comparision operation is a boolean values [true or false]

Operator	Description	Example [a=20, b=10]
>	grater than	a>b ⇒ True
<	Less than	a<b ⇒ false
>=	grater than or equal	a>=b ⇒ True
<=	less than or equal	a<=b ⇒ false
==	is equal to	a==b ⇒ false
!=	not equal to	a!=b ⇒ True

Logical

Logical operators are used to combine Expressions with conditional statement using logical operator [AND, OR, NOT] which results in true or false

operator	name	Description	a=1, b=0
&&	AND	a&&b ⇒ false	
||	OR	a||b ⇒ True	
!	NOT	!a	~~false~~ True

Conditional (or) (Ternary operator)

In Java conditional operator is considered as short hand for if-else statement

Syntax

Condition ? result1 : result2

If condition is true Expression will return result1 if it is not it will return result2.

Bitwise operator

Bitwise operator are used to perform bit level operation over its operand

operator	meaning	Example
&	Binary AND	A&B
|	Binary OR	A|B
^	Binary XOR	A^B

~	onescomplement	(~A)
<<	shift left	A<<1
>>	shift right	A>>1

Example

```
public class main
{
public static void main (String[] args);
  {
   int num1 = 40, num2 = 5, num3 = 0;
System.out.println("num1 & num2 =" + (num1 & num2));
System.out.println(" num1 | num2 =" + (num1 | num2));
System.out.println("num1 ^ num2 =" + (num1 ^ num2));
  num3 = num1 << 1;
System.out.println("num1 << 1 = " + num3);
  num3 = num1 >> 1;
System.out.println(" num1 >> 1 = " + num3);
System.out.println("~num1 = " + ~num1);
  }
}
```

output

num1 & num2 = 0

num1 | num2 = 45

num1 ^ num2 = 45

~~~num1 << 1~~~

num1 << 1 = 80

num1 >> 1 = 20

~num = -41

>> → n/2

40 =

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	1	0	1	0	0	0

~~40<<1~~

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	1	0	1	0	0	0	0

40<<1

⇓

40×2=80

↓

$2^6 + 2^4 = 64 + 16 = \underline{\underline{80}}$
~~~

Conditional operator

conditional (or) ternary operator is considered as short hand for if else statement

Example

```
public class Main
{ public static void main (string[] args);
  {
   String result;

   result =(10>15) ?("greater"):("smaller");
   System.out.println(result);
  }
}
```

Output smaller

Instanceof

The Instanceof operator is used to determine whether an object is an instance of a class, a subclass or not.

Example

```
public class Main
{
public static void main (String[] args);
String str =" Hi";
boolean result;
result = str instanceof String;
System.out.println (result);
}
```

Output

True

Dot operator

(.) operator is also known as member operator it is used to access the member of a package or a class

Arithmetic Example

```
public class Main
{
public static void main (String[] args );
 {
      int a = 20;
      int b = 10;
  System. out. println (a+b);
  system. out. print ln (a-b);
  system. out. print ln (a / b);
  System. out. print ln (a * b);
  System. out. print ln (a % b);
 }
}
```

Output

```
30
10
2
200
0
```

Relational Example

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

```
if (a>b)
{
System.out.println("a is grater than b");
}
else
{
System.out.println("b is grater than a");
}
}
```

Output

a is grater than b

Logical Example

```
public class main
{
public static void main(String[] args);
{
boolean bool1 = true, bool2 = false;
System.out.println("bool1 && bool2 =" + (bool1 && bool2));
System.out.println("bool1 || bool2 =" + (bool1 || bool2));
System.out.println("! (bool1 && bool2) =" + ! (bool1 && bool2));
}
}
```

Output

bool1 && bool2 = false

bool1 && bool2 = true

! (bool1 && bool2) = true

1-20
1-22

Assignment Example

```
public class main
{
public static void main (String[] args);
 {
   int a=30;
   int b= 5;
 a+=b;
System.out.println ("a+=b:" + a);
 a- =b;
System.out.println ("a-=b:" + a);
 a* = b;
System.out.println ("a*=b:" + a);
a /= b;
System.out.println ("a/=b:" + a);
 a% = b;
System.out.println ("a%=b:" + a);
 }
}
```

output

```
a+=b:35
a-=b:30
a*=b:150
a/=b:30
a%=b:0
```

Unary Example

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

```
    int x=10;
System.out.println(x++);
System.out.println(++x);
System.out.println(x--);
System.out.println(--x);
  }
}
```

output

```
10
12
12
10
```

Dot operator

```
class Dot
{
void display()
 {
 double d=20.3;
  int i =(int)d;
 System.out.println(i);
 }
public static void main (String[] args)
  {
    Dot s=new Dot();
     s.display();
  }
}
```

output

```
20
```

Operator precedence and associativity

operator precedence defines the order in which given mathematical Expression is evaluated.

operator with higher precedence is evaluated first and operator with lowest precedence is evaluated at last.

operator associativity

operator with same precedence follows operator associativity.

In Java operators can either follow left associative right associative or have no associality.

operator with left associality are evaluated from left to right.

operator with right associativity are evaluted from right to left

operators with no associatitility does not follow any predefined order

category	operator	Associativity
postfix	++ --	Left to right
unary	+ - ! ~ ++ --	Right to left
multiplicative	* / %	Left to right
additive	+ -	Left to right
shift	<< >>	Left to right

relational	< <= >> =	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
conditional	?:	right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left

premitive type conversion

when a data type is converted into another type is called type conversion

Types of conversion

1. Implicit conversion [coercion]
2. Explicit conversion [casting]

Implicit conversion

In implicit (or) automatic conversion compiler will automatically change one type of data into another

Implicit conversion can happen if both type are compatible and target type is larger than source type

byte → short → int → long → float → double

widening.

In implicit conversion a smaller type is converted into a larger thus it is also known as widening conversion

Example

```
short a=2000;
int b = a;
~~Sop~~;( b);
```

Explicit conversion

There may be situations when we want to convert a value having larger type to smaller type.

double → float → long → int → short → byte

Narrowing

Example

```
int a=10;
byte b=(int) a;
```

control flow statements

In java statements inside your code are generally excuted sequentially from top to bottom.

control flow statements types

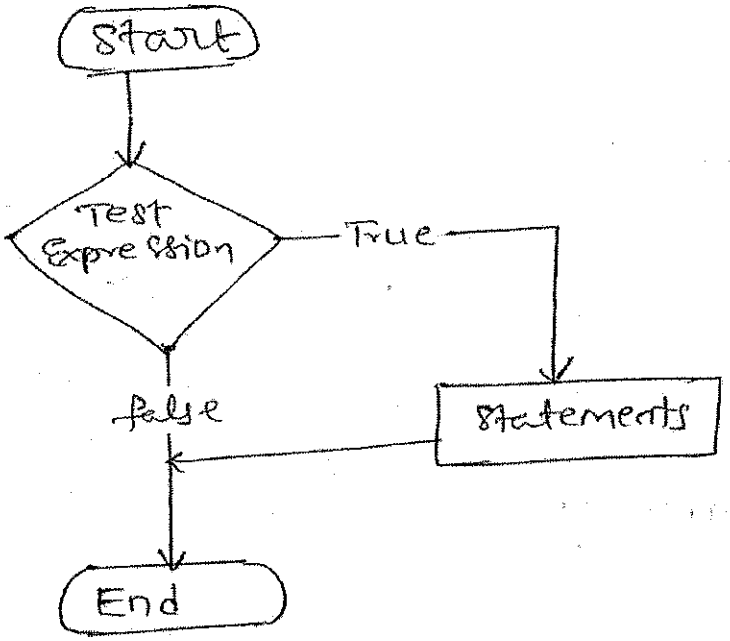
1. Selection statements [if, ifelse, ifelseif ladder, nested if, switch]
2. Iteration statements [for, while, dowhile]
3. Jump statements [break, continue, return]

1. Selection statements

selection statements are also reffered to decision making statements and evaluates single (or) multiple test Expressions

if statement

If statement allows a block of code to be Executed only when a Specified Condition is true



Syntax

```
if (condition)
{
    statements
}
```

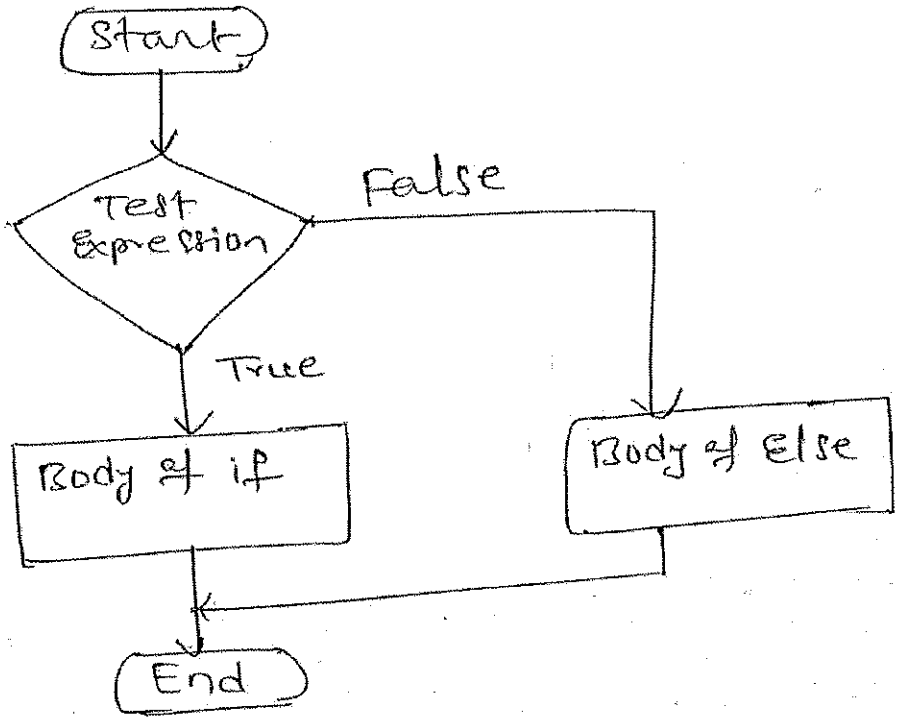
Example

```
public class main
{
  public static void main(String[] args)
  {
    int num=75;        num=20 o/p java if
    System.out.println(" if statement");
     if (num>50) {
    System.out.println("number grater than50");
  }
}
}
```

o/p number grater than50

if else

when we want to execute a block of code when if condition is true and another block of code when if condition is false



Syntax

```
if (condition)
  { statements
  }
  else
   { statements
   }
```

Example

```
public class main
{
 public static void main (String [] args)
  {
    int a=10;
    int b=20;
 system.out.println(" if else statement");
```

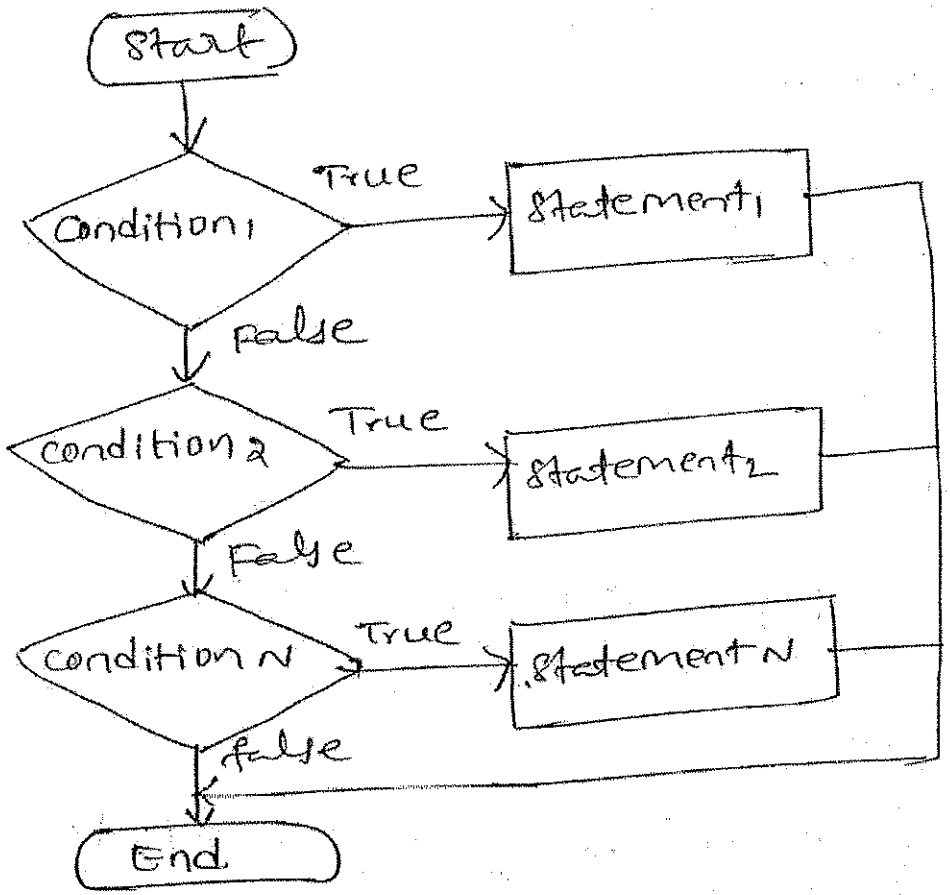
```
if (a>b)
{
System.out.println(" a is greater than b");
 }
 else
 {
System.out.println("b is greater than a");
} }
```

output

if else statement
b is greater than a.

if else if ladder

if else if statement is used to select one among several blocks of code to be executed.



Syntax

```
if (condition1)
{
   statement
}
else if (condition2)
{
   statement
}
=
else if( condition N )
{
   statement
}else
{
   statement
}
```

Example

```
public class main
{public static void main (String[] args);
  { int a=10;
    int b=10;
System. out. println (" if else statement ")
 if (a>b)
{
System. out. println (" a is grater than b");
}
else if (a==b)
{
System. out. println (" a and b are equal");
}
```

```
else
{
System.out.println(" b is greater than a");
}
}
```

output

if else if statement

a and b are equal.

Nested if else

when there is an if statement inside another if statement then it is known as nested if else.

Nested if else can be simplified using java switch case statement

syntax

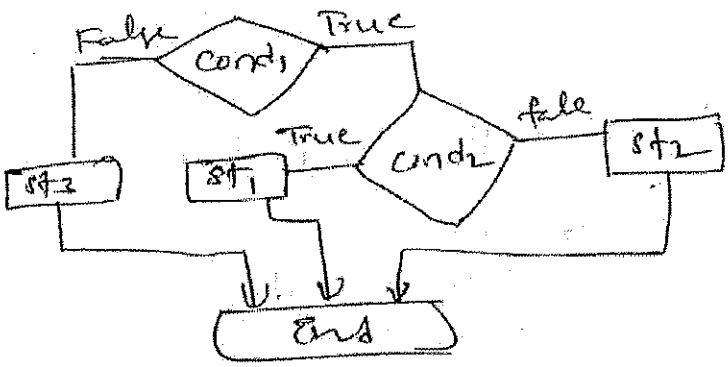
```
if (condition1)
{ if (condition2)
    { statements
    }
  else
    {
      statements
    }
  }
else {
    if (condition3)
    { statement
    }
    else
      { statements
      }
  }
```

(or)

```
if (condition1)
{
  if (condition2)
    statement
    else
    statement
}
else
  {
    if (cond3)
      state
      else
      state
  }
```

Example

```
public class main
{
public static void main (String[] args);
  {
    int a=10;
  System.out.println (" nested if statement ");
   if (a==10)
   {
     if (a<15)
      {
      System.out.println (" a is less than 15 ");
      if (a<12)
      System.out.println ("a is less than 12");
      else
      }
      System.out.println (" a is grater than 12 ");
   }
  }
}
```



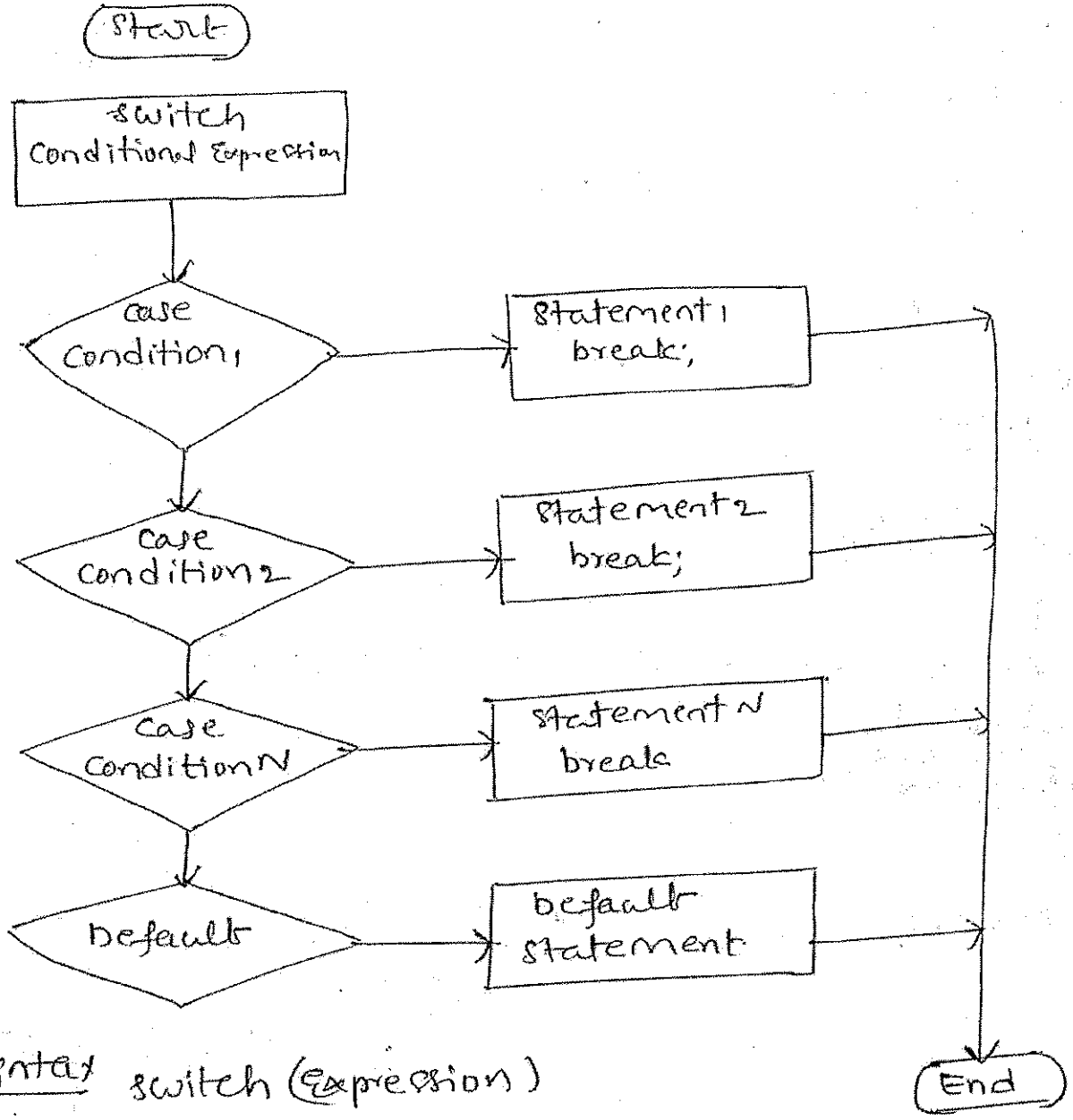
output

nested if statement

a is less than 15
a is less than 12

switch case

switch case statement is simplified from nested if else if, it helps to avoid long chain of if else if statements



<u>Syntax</u> switch (Expression)

```
{ case 1:
      statements
      break;

  case 2:
      statements
      break;

  default:
      statement
      break;
}
```

```
        break;
      }
    }
  }
```


output

Switch case statement

Today is friday.

2. Iteration Statement

Iteration statements are used to execute the block of code repeatedly for specified number of times until it meets a specified condition

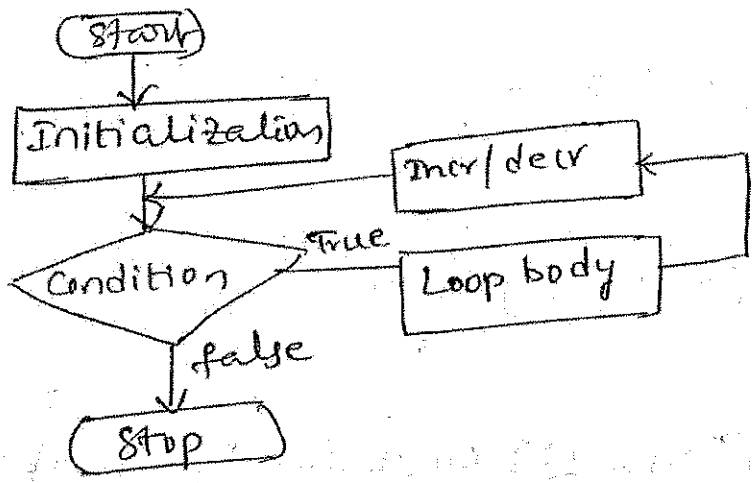
(i) for
(ii) while
(iii) do while

For

For loop takes a variable as iterate through the loop body and assign it with an initial value and iterate through the loop body as long as the test condition is true

Syntax

for (initialization; condition; incr/decr)



Example

```
public class main
{
public static void main (System[] args);
  {  int dayofweek =5;
System.out.println (" switch case statement');
  switch (dayofweek)
  {case1:
System.out.println("Today is monday");
  break;
case 2:
System.out.println(" Today is Tuesday");
  break;
  case3:
System.out.println(" Today is wednesday");
  break;
  case4:
System.out.println (" Today is Thursday");
  break;
  case5:
  System.out.println (" Today is friday");
  break;
  case 6:
  System.out.println ( " Today is saturday");
  break;
  case7:
  System.out.println(" Today is sunday");
  break;
  default:
  System.out.println ("Invalid weekday"); break; }}]
```

Example

```
public class main
{
public static void main (String[] args);
 {
 System.out.println (" for loop statement");
   for (int ctr=1; ctr<=5; ctr++)
  {
  System.out.println (ctr);
  }
 }
}
```

Output

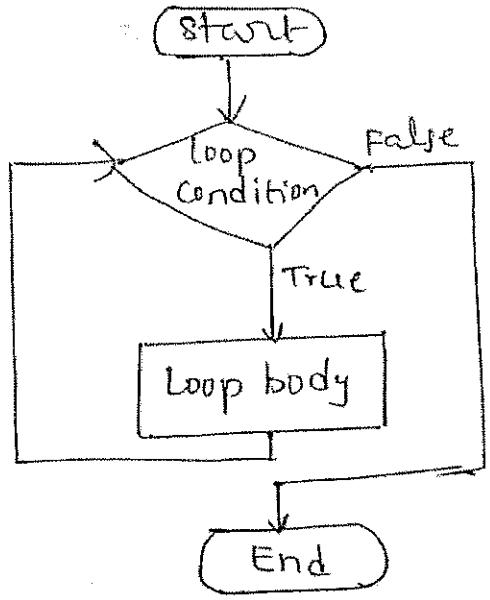
for loop statement
1
2
3
4
5.

While [entry control]

The while loop will Excute a block of statement as long as test expression is true. while loop is useful when the number of iterations can not be predicted where as for loop is iterations are known.

Syntax

```
while (condition)
 {  loop body
 }
```



Example

```
public class Main
{
public static void main(String[] args);
{
    int ctr=1;
    int maxctr=5;
System.out.println(" while loop statement");
  while (ctr<=maxctr)
{
 System.out.println(ctr);
  ctr = ctr+1;
  }
}
}
```

output

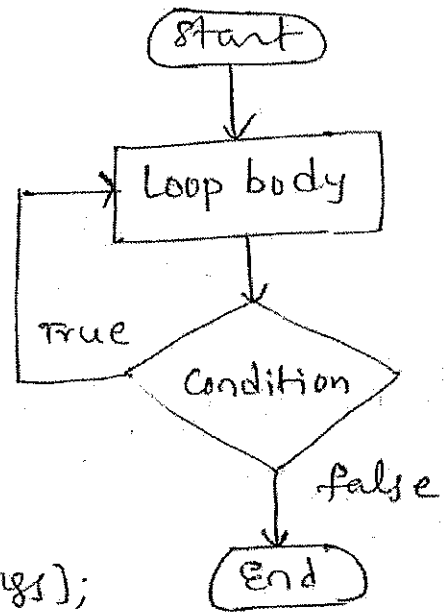
```
  while loop
  1
  2
  3
  4
  5
```

do while (Exit control)

do while loop executes block of statement inside loop body first and then test the condition for next iteration

Syntax

```
do {
    loop body
  } while (condition);
```



Example

```
public class Main
{
public static void main(String[] args);
{ int ctr=1
  int maxctr=5;
 System.out.println(" do while");
 do {
System.out.println(" value is:" +ctr);
    ctr = ctr+1;
  }
  while (ctr<=maxctr);
 }
}
```

output

```
do while loop
value is: 1
value is: 2
value is: 3
value is: 4
value is: 5
```

Jump Statement

Jump statements are used to alter or transfer the control to other section in your program from the current section.

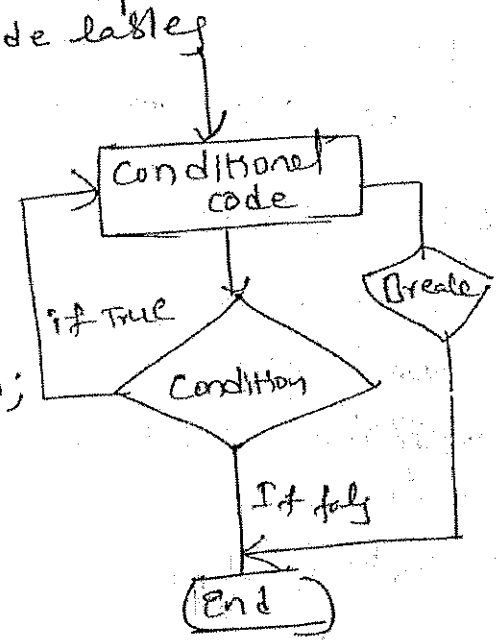
Break

Break statement inside any loop gives you way to break the Execution of loop and transfer the Excution to the next statement following the loop.

* Break statement used in switch, loops and inside lables

Syntax

```
break;
```



Example

```
public class Main
{
public static void main ( String[] args );
 { int count=0;
 while (count <=10) {
  count = count+1;
 if (count==5) {
    break;
 }
 System.out.println ("insideloop" + count);
 }
System.out.println ("out of while loop");
 }
}
```

output

```
inside loop 1
inside loop 2
inside loop 3
inside loop 4
out of while loop
```

continue statement

unlike break statement the continue statement does not break the loop, it skips the specific part of the loop and jumps to the next iteration of the loop immediatly

Example program

```
class continue
{
public static void main (String[] args)
{
for (int i=0; i<=2; i++)
  {
  for (int j=i; j<=5; j++)
    {
     if (j==4)
      {
       continue;
      }
     System.out.println(j);
    }
  }
}
}
```

output

```
0
1
2
3
5
```

1
2
3
5
2
3
5

Return statement

A return statement causes the program control to transfer back to the caller of a method.

* Every method in Java is declared with a return type and it is mandatory for all Java method.

Example program

```
class Returntype
{
 int add()
  {
    int x=30
    int y=70
    int z= x+y;
    return z;
  }
public staticvoid main(String[] args) {
 Returntype test=new Returntype();
    int add = test.add();
  System.out.println(" The sum of x and y is:"+add);
  }
}
```

output

The sum of x and y is: 100.

Class

A class is a group of objects which have common properties

Everything in Java is associated with classes and objects, along with its attributes and method

Example ①

fruit is class and Apple, Banana, mango are the objects of class fruit

② car is an class and volvo, audi and toyota are the objects of 'car' class.

object An object is an instance of a class

* An object is a real world entity
* object is an entity which has state and behaviour

object characteristics

(i) state - represents data (value) of an object

(ii) Behavior - functionality of an object such as deposit, withdraw

(iii) Identity - object identity is an unique ID, Object ID not visible to External user and it is used internally by JVM

A class contains

* fields
* methods
* Constructors
* Nested class and interface

Class declaration

Syntax

```
class classname
{
    field;
    method;
}
```

A class should always starts with an uppercase frist letter and name of the java file should match the class name

Example

```
public class Myclass
{
    int x=5;
}
```

Create an object

In java object is created from a class. we have already created the class named 'Myclass' then to create an object of Myclass, specify the class name, followed by the object name and use the keyword new:

Example

```
public class Myclass
{
```

```
    int x=5;
  public static void main (String[] args)
   {
     Myclass myobj = new Myclass ();
     System.out.println (myobj.x);
   }
 }
```

output

5.

create multiple object

you can create multiple objects of one class but object identity is unique

Example

```
public class Myclass
{ int x=5;
public static void main (String[] args);
  {
    Myclass myobj1 = new Myclass ();
    Myclass myobj2 = new Myclass ();
    System.out.println (myobj1.x);      // object1
    System.out.println (myobj2.x);      // object2
  }
}
```

output

5
5

Method

In java a method is like a function which is used to expose the behaviour of an object

advantages

* code reusability
* code optimization.

methods are declared within a class and that they are used to perform certain actions.

Example

```
public class Myclass
{
static void mymethod();
  {
   System.out.println("Hello word");
  }
}
```

myMethod() prints a text when it is called. To call a method write method name followed by two parentasis and a semicolon;

Inside main, call myMethod():

```
public class Myclass
{ static void MyMethod ()
{
System.out.println("Hello world");
 }
public static void main (String[] args)
 {
   myMethod ();
 }
}
```

output

"Hello world"

ifference between static and public method. 23

tic method

ublic static Myclass

tic void my staticMethod ()

ystem.out.println ("static methods can be called without creating objects");

}

ublic method

public static Myclass

public void mypublicMethod ()

{

System.out.println (" public method must be called by creating objects");

}

Example main method

public static void main (String[] args)

{

mystaticMethod(); // call the static method

my publicMethod (); // This would compile an error.

}

or

myclass myobj = new Myclass ();

myobj.mypublicMethod (); // call public method on object

Access methods with an object

```
public class car
{
  public void fullthrottle()
   {
    System.out.println("The car is going as fast as it can");
   }

  public void speed (int maxspeed)
  {
  System.out.println("max speed is: " + maxspeed);
   }

 public static void main (String[] args)
  {
   car mycar = new car(); // create mycar object
   mycar.fullthrottle();   // call fullthrottle() method
   mycar.speed(200);       // call speed() method.
  }
}
```

output

The car is going as fast as it can

Max Speed is: 200.

Constructor

Constructor is a block of codes similar to the method.
It is a special type of method which is used to initialize object.

There are two types of constructors in Java

1. Default constructor [no arg constructor]
2. parameterized constructor

Default constructor

A constructor when it does not have any parameter is called default constructor.

Syntax

```
Classname()
{
}
```

Example

```
class Bike1
{
  Bike1()
{
System.out.println("Bike is created");
}
public static void main (String[] args)
{  Bike1 b = new Bike1();
}
}
```

Output

Bike is created

parameterized constructor

* A constructor which has a specific number of parameters is called parameterized constructor.
* This constructor is used to provide different values to distinct objects. However you can provide same values also. but in default constructor used to provide default value to the object like 0, null

Example

```
class Student4
{
 int id;
String name;
Student4(int i, String n)
{
 id = i;
name = n;
 }
void display ()
{
System.out.println(id+" "+name);
 }
public static void main (String[] args)
 {
Student4 s1 = new Student4 (111, "Kiran");
Student4 s2 = new Student4 (222, "Arya");
    s1.display();
    s2.display();
} }
```

Output

111 Kiran
222 Arya

Difference b/w constructor and method

constructor	method
constructor name must be same as class name	1. method name may or may not be same as class name
2. Java compiler provides a default constructor if you donnot have any constructor in a class	2. method is not provided by compiler.
3. constructor is invoked implicitly	3. method is invoked explicitly.
4. constructor must not have any return type	4. method must have a return type
5. constructor used to initialize the object	5. method used to expose behaviour of an object.

constructor overloading

* constructor overloading in Java is a technique of having more than one constructor with different parameter lists.

They are differentiated by the compiler by the number of parameters in the list and their types

Example

```
class students
  {  int id;
     String name;
     int age;
     students(int i, String n) // creating two argument
       {                                constructor
         id=i;
         name=n;
       }
     students(int i, String n, int a) // creating three argument
       {                                constructor
         id=i;
         name= n;
         age= a;
       }
     void display()
     {
       System.out.println(id+" "+name+" "+age);
     }
     public static void main(String[] args);
       {
       students s1 = new students(111, 'kiran');
       students s2= new students(222, "Arya", 25);
         s1.display();
         s2.display();
  }
}
```

Output

```
111  kiran  0
222  arya   25
```

The static keyword in java is used for memory management.

We can apply static keyword with

* variable
* method
* Block
* Nested classes

Static variable

* If you declare any variable of static it is known as static variable
* variable used to refer common property of all objects

Ex College name of student

* variable gets memory only once in the class area at the time of class loading

Advantage

* It makes your program memory efficient

Example

```
class student {
  int rollno;   // instance variable
  string name;
 static string college ='KITS';  // static variable

  student (int r, string n) // constructor
  {
     rollno =r;
     name = n;
  }
  void display() {
  system.out.println(rollno+" "+name+" "+college); }

  }
```

```
~~public~~ class Test~~StaticVariable~~, {
public static void main (String[] args)
{
  Student s1 = new Student (111, "Kiran");
  Student s2 = new Student (222, "Arya");
        s1.display();
        s2.display();
  }}
```

output

```
111  Kiran  KITS
222  Arya   KITS
```

Method

If you apply static keyword any method it is known as static method.

* A static method can be invoked without need for creating an instance of a class
* A static method can access static data member and can change the value of it.

Example

```
class calculate {
 static int cube(int x) {
  return x*x*x;
  }
 public static void main (String[] args)
 {
  int result = calculate.cube(5);
  System.out.println(result);
 }
}
```

output 125

Restrictions for static method

1. The static method can not use non static data.
2. this and super can not be used in static context.

Ex
```
Class A {
  int a=40; // non static   [static int a=40 => output 40]
 public static void main(String[] args)
 { System.out.println(a);
  }
}
```

result //Non static variable

compile time error

Static block

* It is used to initialize the static data member
* It is excuted before the main method at the time of classloading.

Example
```
Class A2 {
static { System.out.println("static block is involked");}
 public static void main(String[] args)
 {
  System.out.println("Hello main");
  }
}
```

output

static block is involked

Hello main.

3/ this keyword

* this keyword is used to refer to current object
* this can be passed as an argument to another method
* this can be used to invoke current class constructor.

super() this()	super, this
* These are constructor class to call super class and current class constructor	* These are keywords to refer super class and current class instance members
* we can use only in constructor as first statement only	* we can use anywhere except static area
* we can use only one, but not both simultaionsly	* we can use any number of times

Example

```
class P
{
  String s="parent variable";
}
class C Extends P
{
    String s="child variable";
    public void m1()
    {
System.out.println(" s); // child instance by default
System.out.println(this.s); // child (or) current object reference
System.out.println(super.s); // parent object reference
  }
}
```

```
class Test
{
public static void main (String[] args)
    {
        C c = new C();
        c.m1();
    }
}
```

output

```
child variable
child variable
parent variable
```

Arrays

* An arrays is an indexed collection of fixed no of homogeous data elements.
* The main advantage of arrays is we can represent huge no of values by using single variable so that readability of code will be improved.

But the main disadvantage of arrays is fixed in size i.e once we create an array there is no chance of increase/decrease size based on our requirement.

* Hence to use arrays concept compulsory we should know the size in advance, which may not possible always

Array declaration

one dimensional array declaration

```
int[] x;
int [] x;
int x[];
```

} all are correct but highly recommended int[] x; because name is clearly separated from type

Example 2

```
class A
{
public static void main (String args[])
  {
    for (int i=0; i<args.length; i++)
        System.out.println (args[i]);
  }
}
```

output

```
JAVA A SONOO RAMU 13 abc
  SONOO
  RAMU
   1
   3
  abc
```

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

output

```
JAVA Test hi 123 bye
 hi
 123
 bye
```

String data type is used as command line arguments because most of object are used in string datatype in Java and the string data type can be converted to any other data types.

Nested classes

A class defined within another class is known as nested class

Advantages

* It is a way of logically grouping classes that are only used in one place

* It increases encapsulation
* It can lead to more readable and maintainable code
* code optimization

Types of nested class

1. Static nested class

A class defined within another class, has static modifier applied in it, then it is called as static nested classes.

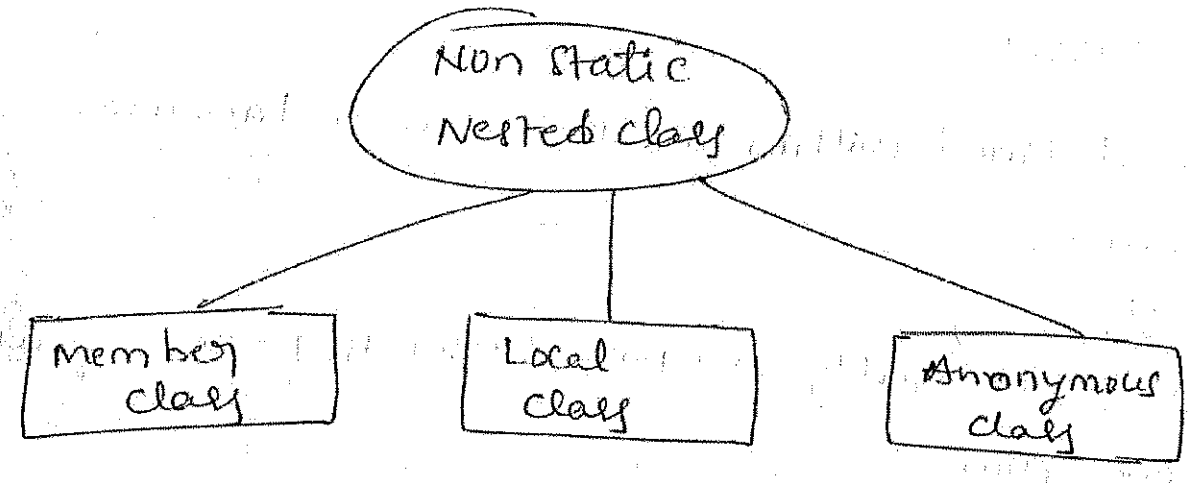
Static nested class is rarely used.

2. Non static nested class

* Non static nested class is the most important type of nested class. It is also known as Inner class

Inner class access to all variables and methods of outer class including its private data members and methods directly. but outerclass cannot directly access members of innerclass.

Inner class can be declared private, public, protected and default access whereas an outer class can have only public or default access.



Member inner class

A non static nested class that is created outside a method is called member inner class

Local inner class

A non static nested class that is created inside a method is called local inner class

Anonymous class

A class without any name is called anonymous class This annonymous class will provide implementation of type() method.

* Its name is decided by the java compiler

The main difference between nested class and inner class is the inner class is a part of nested class and non static nested classes are known as inner classes.

Modifiers

We divide modifiers into two types

1. Access modifiers
2. Non access modifiers.

Access modifiers

There are four types of access modifiers available in java.

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

Default

* The data members, class which are not declared using any access modifiers.
* Default access modifiers are accessible only within the same package

private

* The private access modifiers is specified using keyword private
* The method or data members declared as private are accessible only within the class in which they are declared
* Any other class of same package will not able to access these members.

protected

* The protected access modifier is specified using the keyword protected

* The method or data members declared as protected are accessible within the same package or subclass in different package.

public

* The public access modifier is specified using the keyword public.
* Classes, methods are declared as public are accessible from everywhere in the program..

Non access modifiers

Non access modifiers do not control access level but provides other functionality.

* final - The class can not be inherited by other classes
* abstract - The class can not be used to create objects.
* static modifier for creating class methods and variables

passing arguments by value and by reference

call by value

* call by value means calling a method with a parameter as value. The argument value is passed to the parameter.
* In call by value the modification done to the parameter passed does not reflect the caller

call by reference

* call by reference means calling a method with a parameter as a reference.
* The argument reference is passed to the parameter
* In call by reference the modification done to the parameter passed does reflected the caller

call byvalue program

```
data operation
{
 int data =50;
 void change (int data)
  {
  data = data +100; //change in local variable only
  }
 public static void main (string[] args)
  {
  operation op= new operation();
  system.out.println(" before change " + op.change(data);
```

```
op.change(500);
System.out.println("after change" + op.data);

    }
  }
```

output

before change 50
after change 50

pass by Reference program

```
class CallByReference
{
 public void display(StringBuffer sb2)
  {
   sb2.append("world");
   System.out.println(sb2);
  }
 public static void main(String[] args)
 {
CallByReference cbr=new CallByReference();
 StringBuffer sb1 = new StringBuffer("Hello");
    cbr.display(sb1);
   System.out.println(sb1);
  }
}
```

output

Hello world

Method

* A method is a block of code which only runs when it is called.
* you can pass data known as parameter into a method

create a method

* A method must be declared within a class
* It is defined with the name of the method followed by parentheses ().
* Java provide some predefined methods

Example

```
class main
{
static void mymethod()
{

}

}
```

→ mymethod() is the name of the method.

→ static means method belongs to main class and not an object of main class.

call a Method

To call a method in Java write the methods name followed by two parentheses () and a semicolon;

mymethod() is used to print a text when it is called

program

```
class main
```

```
{
Static void mythod()
[
System.out.println(" I got Excuted");
}
public static void main(String[] args)
[
mymethod();
}
}
```

output

I got Excuted.

Example (2)

```
class main{
static void mymethod()
{ System.out.println(" I got Excuted");
}
public static void main(String[] args)
{
mymethod();
mymethod();
}
}
```

output

I got Excuted

I got Excuted.

Method overloading

method overloading is a concept that allows to declare multiple methods with same name but different parameters in the same class.

* Java support method overloading but not method overriding.
* method overloading can be done by changing number of arguments or changing data type of arguments.

There are two different ways of method overloading

1. Different types of arguments.
2. Different number of arguments

1. changing data type of arguments

we have two sum() methods that take integer and float type arguments respectively.

```
class calculate
{
 void sum (int a, int b)
  {
  System.out.println("sum is"+(a+b));
  }
 void sum (float a, float b)
  {
  System.out.println(" sum is" +(a+b));
  }
 public static void main (String[] args)
  {
 Calculate cal=new Calculate();
```

```
    cal.sum(8,5);          // method is called
    cal.sum(4.6f,3.8f);    // method is called
  }
}
```

output

sum is 13.

sum is 8.4.

2. changing no of arguments

```
class Demo
{
void multiply(int l, int b)
  {
    System.out.println("Result is" + (l*b));
  }
void multiply(int l, int b, int h)
  {
   System.out.println("Result is"+(l*b*h));
  }
public static void main(String[] args)
  {
    Demo ar=new Demo();
    ar.multiply(8,5);  // method is called
    ar.multiply(4,6,2); // method is called
  }
}
```

output

Result is 40

Resultt is 48

Access control for class Member

* A public class member can be accessed by any other code
* A private class member only be accessed within its class
* A defullt access class member has no access specifier.
* A protected feature of class is available to all classes in the same package and its subclass

program

```
class Test
int a;    // default access
public int b; // public access
private int c; // private access

void setc(int i)
{
 c = i;
}
int getc()
{
 return c;
}
}
public class main
```

```
{
public static void main (String[] args)
{
 Test obj =new Test();
      obj.a=1;
      obj.b=2;

      ~~obj.c=100~~
      obj.setc(100);
 System.out.println("a, b and c:" + obj.a + " "
          + obj.b + " " + obj.~~getc()~~getc());
}
}
```

<u>output</u>

a, b and c: 1 2 100

Recursive method

* Recursion is a technique that making a method call itself.
* This technique provides a way to break complicated problems down into simple problems which are easier to solve.
* Recursion may be difficult to understand.

program

```
class Factorial
{
static int factorial (int n)
  {
   if(n!=0)
     return n*factorial(n-1);
   else
     return 1;

  }
  public static void main (String[] args)
  {
  int number = 4, result;
  result = Factorial (number);
```

```
System.out.println(number+ "factorial= "
        + result);
    }
}
```

<u>output</u>

4 factorial = 24

The above program factorial() method is calling itself.

Nesting of methods

When a method in Java calls another method in the same class it is called Nesting of method

program

```
class Test
{
 int a, b
  test (int p, int q)
   {
     a= p;
     b= q;
   }
  int greatest ()
  {
    if (a>=b)
     return (a);
    else
     return (b);
  }
  void display ()
  {
   int great= greatest();
  System.out.println (" The greatest value="
              + great);

  }
}
~~class temp~~
```

```
public static void main (String[] args)
 {
  test  t1 = new test (25,24);
   t1. display ();

  }
 }
```

<u>output</u>

The greatest value = 25

Method overriding

* Declaring a method in sub class which is already present in parent class is known as method overriding
* In this case the method in parent class is called overridden method and method in child class is called overriding method

Example

* We have two classes. A chaild class Boy and a parent class Human.
* The Boy class Extends Human class. Both the classes have a common method void eat().
* The Boy class is giving its own implementation to the eat()method. so that when it call this method, it prints Boy is eating instead of Human is eating

program

```
class Human
{
public void eat()
 {
  System.out.println("Human is eating");
  }
}
class Boy Extends Human
{
```

```
public void eat()
{
System.out.println(" Boy is eating");
}
public static void main (String[] args)
{
Boy b=new Boy();
b.eat();
}
}
```

output

Boy is eating

Advantage

The main advantage of method overriding is class can give its own specific implementation to a inherited method without modifying the parent class code

UNIT-III

Array

An array is a collection of similar types of data.

Example

If we want to store the names of 100 people then we can create an array of the string type that can store 100 names.

```
String[] array = new String[100];
```

How to declare an array

```
dataType[] arrayname;
```

→ dataType means like int, char, double and byte etc

→ arrayname means it is an identifier.

Example

```
double[] data;
```

In Java we can declare and allocate memory of an array in one single statement.

```
double[] data = new double[10];
```

Here array can store 10 Elements.

How to Initialize array

In Java we can initialize array during declaration.

```
int[] age = {12, 4, 5, 2};
```

Here we have created an array named age and initialized with the value inside the curly brackets.

* In Java array each memory location is associated with a number. The number is known as array index.
* We can also initialize arrays in Java using the index number

Example

```
int[] age = new int[5];   // declare an array.

age[0] = 12;
age[1] = 4;
age[2] = 5;
age[3] = 2;
age[4] = 5;
}  // initialize an array
```

program

```
class Array
{
public static void main (String[] args)
{
int a[] = new int[5]; // declaration & instantiation

 a[0] = 10;   // initialization
 a[1] = 20;
 a[2] = 70;
 a[3] = 40;
 a[4] = 50;
for (int i=0; i<a.length; i++)
 System.out.println(a[i]);
 }
}
```

output

```
10
20
70
40
50.
```

age[0]	age[1]	age[2]	age[3]	age[4]
12	4	5	2	5

* Array indexes always start from 0. i.e the first element of an array is at index 0.
* If the size of an array is 'n' then the last element of the array index is 'n-1'.
* The main advantage of array is we can represent huge no of values by using single variable so that readability of code will be improved.

* The main disadvantage of array is fixed in size. i.e once we create an array then there is no chance to increase/decrease of size base on our requirement

Acessing Elements of array

* once the array is created you can access an array Element by using the name of an array followed by an index enclosed between a pair of square brackets.
* The index of an array indicates the position of an Element in the array.
* The index of first Element in the array is always '0' (zero) and the second Element has an index 1 and so on.
* The index of last Element is always less than the number of Elements in the array

The syntax for accessing an array Element is

```
array Refvar[index];
```

program

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

```
{
 int[] num = new int[5];
   num[0] = 5;
   num[1] = 15;
   num[2] = 25;
   num[3] = 30;
   num[4] = 50;
 System.out.println("Array Element num[0]:" + num[0]);
 System.out.println("Array Element num[1]:" + num[1]);
 System.out.println("Array Element num[2]:" + num[2]);
 System.out.println("Array Element num[3]:" + num[3]);
 System.out.println("Array Element num[4]:" + num[4]);
  }
 }
```

output

```
Array Element num[0]: 5
Array Element num[1]: 15
Array Element num[2]: 25
Array Element num[3]: 30
Array Element num[4]: 50
```

operations on array Elements

* Allocation of array in java is dynamic

*Basic operations

1. Traverse

It prints all the Elements in the array one by one

2. Insertion

Add an Element at the given index.

3. Deletion

Delete an Element at the given index

4. Search

Searches an Element in the array using the given index

5. update

update an Element at the given index.

Traversal program

```
class Traversal
{
public static void main(String[] args)
  {
int[] intarr = {2,3,5,7,11,13,17,19};
```

```
System.out.println(" The array Elements are");
System.out.println (intarr[i] +" ");
    }
  }
}
```

output

The array elements are

2 3 5 7 11 13 17 19

Insertion

```
class Insertion
{
public static void main(String[] args)
 {
 int n=5, pos=x;
 int intarr[] = new Int[n+1];
 Scanner sc = new Scanner (System.in);
 System.out.println("enter the array elements:");
 for (int i=0; i<n; i++)
  {
  intarr[i] = sc.nextInt();
  }
```

```
System.out.println("array Element before insertion:");
for (int i=0; i<n; i++)
{
System.out.println(intarr[i]+" ");
}
pos=3;
x=24;
for (int i=(n-1); i>=(pos-1); i--)
{
intArr[i+1] = intArr[i];
}
intArr[pos-1] = x;
System.out.println("In array Elements after
                    insertion:");
for (int i=0; i<n; i++)
{
System.out.println(intArr[i]+" ");
}
System.out.println(intarr[n]);
}
}
```

output

Enter the array Elements

2 3 5 7 11

Array Elements before insertion

2 3 5 7 11

Array Elements after insertion

2 3 24 5 7 11.

Deletion program

```
import java.util.Scanner;
class Deletion
{
public static void main (String[] args)
{
int n=5, x=10, flag=1, loc=0;
Scanner s= new Scanner (System.in);
int intarr[] = new int[n];
Scanner sc = new Scanner(System.in);
System.out.println("Enter array Element:");
for (int i=0; i<n; i++)
{
intarr[i] = sc.nextInt();
}
```

```
System.out.println("array Elements before insertion:");
 for (int i=0; i<n; i++)
 {
 System.out.println(intarr[i]+" ");
 }
 for (int i=0; i<n; i++)
 {
  if (intarr[i]==x)
  {
  flag=1;
   loc =i;
   break;
  }
  else
  {
  flag=0;
  }
 }
 if (flag==1)
 {
 for (int i=loc+1; i<n; i++)
  {
   intarr[i-1] = intarr[i];
  }
  System.out.println("\n array Elements after
                deleting:");
  for (int i=0; i<n-2; i++)
  {
```

3.b

```
System.out.print(intarr[i]+ " ");
}
System.out.print (intarr[n-2]);
}
else
{
System.out.println(" Element not found");
}
}
}
```

output

Enter the array Elements

5 10 15 20 25

Array Elements before ~~insertion~~ deletion

5 10 15 20 25

Array Elements after deletion

5 15 20 25.

Searching

```
import java.util.Scanner;
class Searching
{
public static void main(String[] args)
{
int search_ele = 15;
int n = 5, flag = 0;
Scanner s= new Scanner(System.in);
int intarr[] = new int[n];
Scanner sc = new Scanner (System.in);
System.out.println("Enter array Element:");
for (int i=0; i<n; i++)
{
intarr[i] = sc.nextInt();
}
System.out.println("The array Element are:");
for(int i=0; i<n; i++)
{
System.out.print (intarr[i]+" ");
}
for(int i=0; i<5; i++)
{
if (intarr[i] == search_ele)
{
flag =1;
break;
```

```
    }
  }
  if (flag == 1)
 {
 System.out.println("\n Element found");
  }
  else
  {
  System.out.println("\n Element not found");
    }
  }
}
```

output

```
Enter the array Elements
 1 2 3 4 6
The array Elements are:
 1 2 3 4 6
 Element not found
```

updation

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

```
}
int Search_ele = 15;
int n=5, x;
Scanner sc = new Scanner(System.in);
int intarr[] = new int[n];
Scanner sc = new Scanner(System.in);
System.out.println("\n Enter array Element:");
for (int i=0; i<n; i++)
{
intarr[i] = sc.nextInt();
}
System.out.println("\n Enter array Element are:");
for (int i=0; i<n; i++)
{
System.out.print(intarr[i]+" ");
}
System.out.println("\n Enter new element:");
x = sc.nextInt();
for (int i=0; i<5; i++)
{
if (intarr[i] == Search_ele)
{
intarr[i] = x;
break;
}
}
```

```
System.out.println("array Element after updating:");
 for(int i=0; i<n; i++)
 {
 System.out.println(intarr[i]+" ");
  }
 }
}
```

output

Enter the array Element

5 10 15 20 25

The array Elements are

5 10 15 20 25

Enter the new Element:

35

array Elements after updating:

5

10

35

20

25.

Assigning array to another array

array in Java can be copied to another array using following ways.

* using variable assignment but is not better way.
* create a new array of the same length and copy each element
* use the clone method of array. clone method create a new array of same size.
* use system.arraycopy() method. arraycopy() can be used to copy a subset of an array.

program

```
class Copyarray
{
public static void main(String[] args)
  {
   int[] arr1 = new int[] {1,2,3,4,5};
   int[] arr2 = new int[arr1.length];
   for(int i=0; i<arr1.length; i++)
   {
    arr2[i] = arr1[i];
   }
  System.out.println("Elements of original array");
for(int i=0; i<arr1.length; i++)
```

```
    {
     System.out.println(arr1[i]+" ");
    }
    System.out.println();
    System.out.println("Elements of new array:");
    for(int i=0; i<arr2.length; i++)
     System.out.println(arr2[i]+" ");

    }
  }
}
```

output

Element of original array

1 2 3 4 5

Elements of new array

1 2 3 4 5.

Dynamic change of array size

* Dynamic array in Java means either grow or shrank the size of the array depending upon user requirements.

* while an Element is removed from an array the array size must be reduce and If an Element is added to an array then the array size becomes stretch. (or) grow.

declaration of dynamic array

An array can be declared in three ways.

1. Array[]

Example int Array[] = {1,2,4};

2. [] Array

Example int[] Array = {1,2,4};

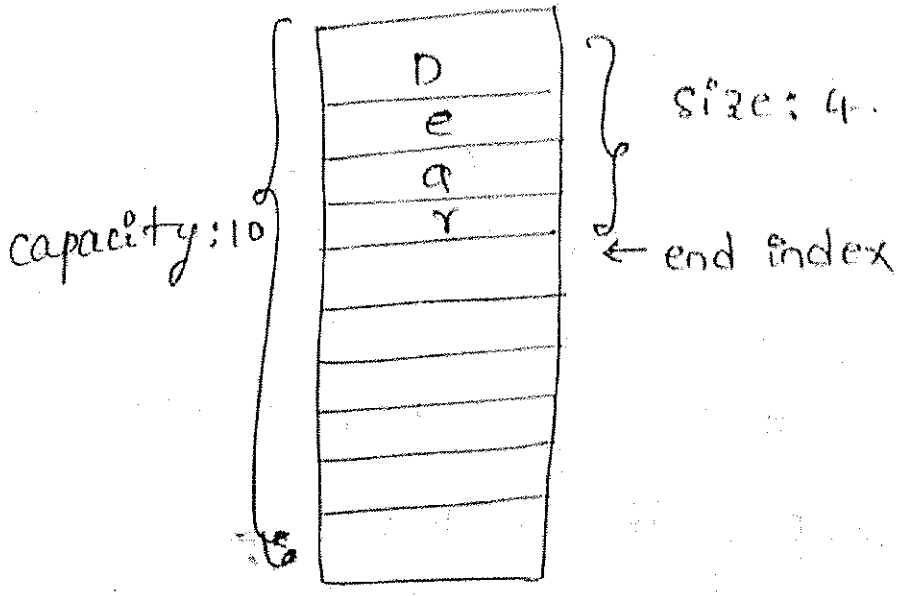
3. [] A

Example int [] Array = {1,2,4};

Size Vs capacity

* Array to use. 10 indices but at present we append four items to our dynamic array.

Therefore the dynamic array size is from '4' and its capacity is '10'



Delete an Element

while deleting an element from an array the remove() removes an element from the end and automatically stores zero at the last index.

0	1	2	3	4	5	6	7	8	9
11	22	33	44	0	0	0	0	0	0
3300	3304	3308	3312	3316	3320	3324	3328	3332	3336

Remove(1)

0	1	2	3	4	5	6	7	8	9
11	33	44	0	0	0	0	0	0	0
3300	3304	3308	3312	3316	3320	3324	3328	3332	3336

Resize an array

when arrays have no data at the right side of array which than take unnecessary memory the method srinkSize() frees Extra memory.

0	1	2	3	4	5	6	7	8	9
11	22	33	44	55	0	0	0	0	0
3300	3304	3308	3312	3316	3320	3324	3328	3332	3336

SrinkSize()

0	1	2	3	4
11	22	33	44	55
3300	3304	3308	3312	3316

→ index values

→ Elements

→ address

Add element

There can be create a fixed size array if we required to add some more elements in the array. usually it creates a new array of double size.

0	1	2	3	4
19	20	21	22	23
100	104	108	112	116

0	1	2	3	4	5	6	7	8	9
19	20	21	22	23	0	0	0	0	0

program

```
class Array
{
private int arr[];
private int count;
public Array(int length)
{
  arr = new int[length];
 }
public void printArray()
{
for (int i=0; i<count; i++)
  {
  System.out.print(arr[i]+" ");
  }
 }
public void insert(int element)
{
  if (arr.length==count)
    {
    int newArr[] = new int[2*count];
    for (int i=0; i<count; i++)
      {
       newArr[i]= arr[i];
      }
      arr = newArr;
    }
    arr[count++] = element;
    }
    }
```

```
Class CFG
{
public static void main (String[] args)
  {
  Array numbers = new Array(3);
   numbers.insert(10);
   numbers.insert(20);
   numbers.insert(40);
   numbers.insert(50);
   numbers.printArray();
  }
```

output

10 30 40 50.

Sorting of Arrays

* Sorting is a way to arrange elements of a list in certain order. The order may be in ascending or descending order.

Ascending order

The ascending order arranges the elements in the lowest to highest order.

We can perform sorting in the following ways

1. using sort() method
2. without using method.
   → using for loop
   → using user defined method.

Using sort()

* java.util package provides sort() method to sort an array in ascending order.
* It accepts an array type int, float, double, long, char, byte

program

```
import java.util.Arrays;
  class Sortarray
  {
 public static void main(String[] args)
   {
    int[] array = new int[] {90,23,5,109,12,22,67,34};
    Arrays.sort(array);
   System.out.println(" Elements of array sorted in
                    ascending order:");

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

```
    }
  }
}
```

Output

Array Elements in ascending order:

5 12 22 23 34 67 90 109.

Without using method

using for loop

```
class sortfor
{
public static void main(String[] args)
 {
 int[] arr = new int[]{78,34,1,3,90,34,-1,-4,6,55};
 System.out.println("Array elements after sorting:");
  for (int i=0; i<arr.length; i++)
   {
    for(int j= i+1; j<arr.length; j++)
     {
      int tmp =0;
       if (arr[i] > arr[j])
        {
```

```
        tmp = arr[i];
        arr[i] = arr[j];
        arr[j] = tmp;
        }
      }
    System.out.println(arr[i]);

      }
    }
  }
```

output

-4, -1, 1, 3, 6, 34, 34, 55, 78, 90.

Using user defined Method

we have defined method name is sortArray().

```
class Sorduserdefined
{
public static void main(String[] args)
 {
  int i;
  int array[] = {12, 45, -1, 1};
System.out.println("Array Elements before sorting");
 for(i=0; i< array.length; i++)
 System.out.println(array[i]);
  sortArray(array, array.length);
```

```
System.out.println("after Elements sorting:\n");
 for (i=0; i<array.length; i++)
  {
  System.out.println(array[i]);
  }
 }
private static void SortArray(int array[], int n)
 {
 for (int i=1; i<n; i++)
  {
   int j=i;
   int a=array[i];
  while ((j>0) && (array[j-1]>a))
  {
   array[j] = array[j-1];
   j--;
  }
   array[j]=a;
  }
 }
}
```

output

array Element before sorting

12 45 -1, 1

array Element after sorting

-1, 1 12 45

Descending order

The descending order arranges the elements in highest to lowest order. we can perform sorting following ways

1. Using reverseOrder() method
2. Without using method.
   - → using for loop
   - → using User Defined Method.

reverseOrder()

Java class provides the reverseorder() method to sort the array in reverse order.

program

```
import java.util.Arrays;
import java.util.Collections;
class sortdeverse
{
public static void main(String[] args)
 {
Integer[] array = {23, -9, 78, 102, 4, 0, -1, 11, 6, 110, 205};
  Arrays.sort(array, Collections.reverseOrder());
System.out.println("array Elements in descending:"+
                    Arrays.toString(array));
}
}
```

output

Array Elements in descending: 205, 110, 102, 78, 23, 11, 6, 4, 0, -1, -9

program (string)

```
import java.util.Arrays;
import java.util.Collections;
class sortdescstring
{
public static void main (String[] args)
  {
String[] starray = {"mango", "Apple", "Grapes", "Banana"};
 Arrays.sort(starray, Collections.reverseOrder());
System.out.println(" Array Elements in descending
                order:"+ Arrays.toString(starray));
  }
}
```

output

Array Elements in descending order: mango Grapes Banana
Apple

Search for value in Arrays

In java given array the task is to check whether a certain element is present in this array or not.

Example

Input – arr[] = { 5, 1, 9, 7, 2, 6, 10 } key = 7

output – true

Input – arr[] = { -1, 1, 5, 8 } key = -4.

output – false

There are various ways to search value in array some of those are.

1. Linear search

In this list or array is traversed sequentially and every element is checked.

program

```
import java.util.Arrays;
import java.util.stream.IntStream;
class Linear {
private static void check(int[] arr, int toCheckValue)
{
 boolean test = false;
 for (int element : arr)
  {
   if (element == toCheckValue)
    {
```

```
            test = true;
            break;
        }
    }
    System.out.println("Is" + toCheckValue + "present
            in the array:" + test);
    }
public static void main(String[] args)
{
int arr[] = {5,1,1,9,7,2,6,10};
int toCheckValue = 7;
System.out.println(" Array:" + Arrays.toString(arr));
 check(arr, toCheckValue);
}
```

<u>output</u>

Array: {5, 1, 1, 9, 7, 2, 6, 10}

Is 7 present in the array: true.

2. Binary search method

In this search sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array.

If the value of search key is less than the item in the middle of the interval the it take lower half otherwise it take upper half. Repeatedly check until the value is found or the interval is empty.

program

```
import java.util.Arrays;
import java.util.stream.IntStream;
class Binarysearch
{
private static void check(int[] arr, int toCheckValue)
{
  Arrays.sort(arr);
int res = Arrays.binarySearch(arr, toCheckValue);
boolean test= res>0 ? true: false;
System.out.println(" Is"+ toCheckValue+"
            present in the array:"+test);

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

```
int arr[] = { 5,1,9,7,2,6,10};
int toCheckValue = 7;
System.out.println("Array:"+ Arrays.toString(arr));
check(arr, toCheckValue);
  }
}
```

<u>output</u>

Array: 5, 1, 9, 7, 2, 6, 10

Is 7 present in the array; true

<u>class Arrays</u>

* The java.util.Arrays class contains various static methods for sorting and searching arrays, comparing arrays, and filling array elements.
* These methods overloaded for all primitive types

1. public static ~~void~~ int binarySearch object[]a, object key

search the specified array of object Byte, int, double etc. for the specified value using the binary search algorithm.

2. public static boolean equals long[]a, long[]a2

It returns true if the two specified arrays of longs are equal to one another.

3. public static void fill int[]a,intval.

Assign the specified int value to each element of the specified array of int, same method could be used by all primitive data types

4. public static void sort object[]a

sort the specified array of objects into ascending order, according to natural ordering of its elements. same method could be used by all other primitive data types.

Two-dimensional Array

* Two dimensional array is the simplest form of multidimensional array.
* A two dimensional array can be seen an array of one dimensional array for easier understanding.

Declaration Syntax

datatype[][] arrayname = new datatype[X][Y];

Example int[][] arr = new int[10][20];

Initilization Syntax

arrayname [row index][column index] = value;

Example arr[0][0] = 1;

[OR]

Declaration Syntax

datatype[][] arrayname = { {value R1C1, value R1C2},
{value R2C1, value R2C2} }

Example

int[][] arr = { {1,2}, {3,4} };

Accessing Elements of 2D Array

Elemements in two dimensional array are commonly oreffered by x[i][j] where 'i' is the row number and 'j' is the column number.

Syntax

x [row index] [column index]

Example

int[][] arr= new int [10][20];

Representation of two dimensional array

	column 0	column 1	column 2
Row 0	x [0][0]	x [0][1]	x [0][2]
Row 1	x [1][0]	x [1][1]	x [1][2]
Row 2	x [2][0]	x [2][1]	x [2][2]

program 2D Array

```
import java.util.Scanner;

class Twodimensional
{
```

```
public static void main (String[] args)
{
int row, col, i, j;
int arr[][] = new int[10][10];
Scanner sc = new Scanner(System.in);
System.out.println(" Enter no of rows for array
                    (max 10):");
row = sc.nextInt();
System.out.println(" Enter no of column for array
                    (max 10):");

col = sc.nextInt();
System.out.println(" Enter"+(row*col)+" Array
                    Elements:");

for (i=0; i<row; i++)
{
 for (j=0; j<col; j++)
   {
     arr[i][j] = sc.nextInt();
   }
 }
System.out.println("The array is:\n");
for (i=0; i<row; i++)
   {
     for (j=0; j<col; j++)
       {
```

```
System.out.println(arr[i][j]+" ");
    }
    System.out.println();
    }
  }
}
```

<u>Output</u>

```
Enter number of row for array(max 10): 3

Enter number of col for array(max 10): 3

 Enter 9 array Elements

  1
  2
  3
  4
  5
  6
  7
  8
  9

 The Array is:

   1   2   3

   4   5   6

   7   8   9.
```

Three dimensional Array

* Three dimensional array is complex form of a multidimensional array.
* A Three dimensional array can be seen an array of two dimensional array for easier understanding.

Declaration Syntax

datatype[][][] arrayname = new datatype[x][y][z];

Example int[][][] arr = new int[10][20][30];

Initialization Syntax

arrayname [array index] [row index] [column index] = value;

Example arr[0][0][0] = 1;

accessing Elements of

Elements in three dimensional array are commonly referred by x[i][j][k] where 'i' is the array number

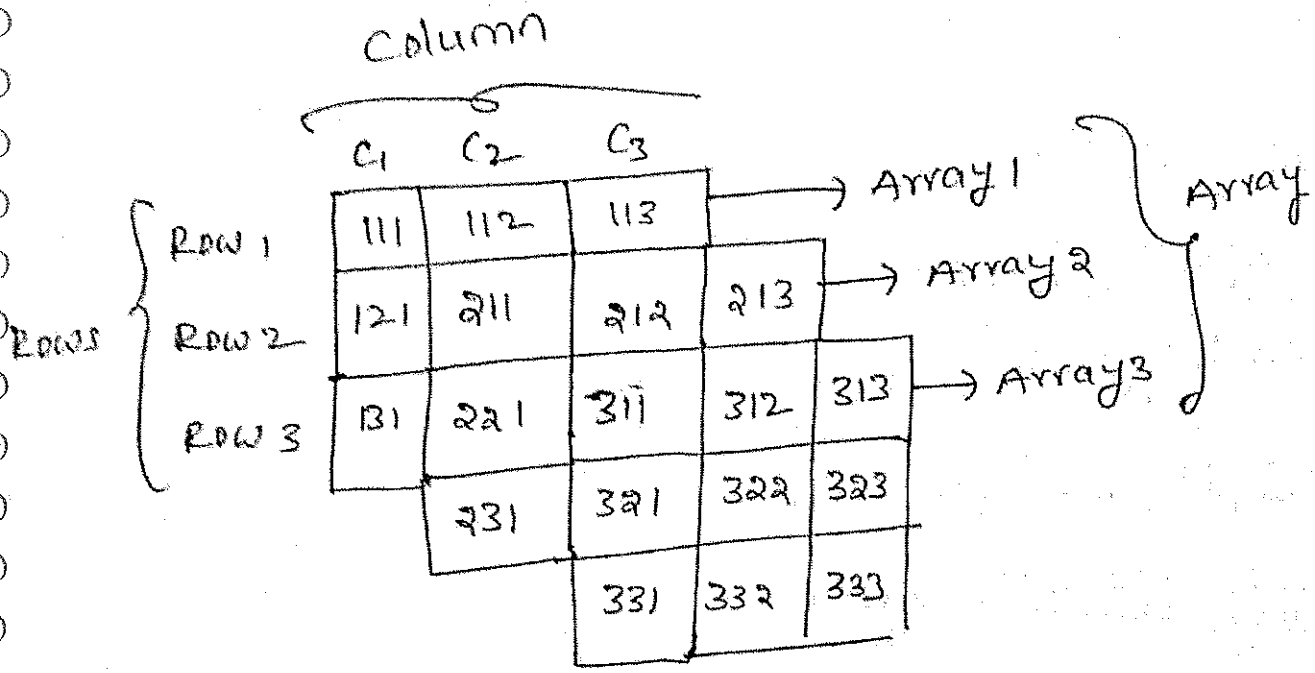
'j' is the row number

'k' is the column umber

Example

```
int[][][] arr = new int[10][20][30];
arr[0][0][0]=1;
```

Representation of 3D Array



program for 3D Array

```
import java.util.Scanner
class Threedimensional
{
public static void main(String[] args)
{
int arr[][][] = new int[3][4][2];
int i, j, k, num=1;
```

```
for (i=0; i<3; i++)
 {
   for (j=0; j<4; j++)
     {
       for ( k=0; k<2; k++)
          {
            arr[i][j][k] = num;
            num++;
          }
       }
     }
   for (i=0; i<3; i++)
    {
      for (j=0; j<4; j++)
        {
          for (k=0; k<2; k++)
            {
              System.out.println("arr["+i+"]
["+j+"] ["+k+"]="+arr[i][j][k]+" \t");
 }
 System.out.println();
}
System.out.println();
```

int, char, long etc. whereas vector can hold only java objects.

Size

To find size of the vector, we can call its size() method, whereas an array has a length property that stores its length

Dimensions

Arrays in Java supports single dimensional array as well as multidimensional arrays.

A vector has no concept of dimension but we can easily construct a vector of vectors.

Support for generics

Vector supports generics to ensure type safty while an array does not support generics.

Inheritance

Inheritance in Java is a mechanisum in which one object acquires all the properties and behaviors of a parent object.

Why use inheritance

* For method overriding
* For code reusability

Terms used in inheritance

* **class** A class is a group of object which have common properties
* **subclass/child class**
  The class that inherits from another class
* **super class/parent**
  The class being inherited from
* **Reusability**
  you to reuse fields and method of Exsiting class when you create a new class.

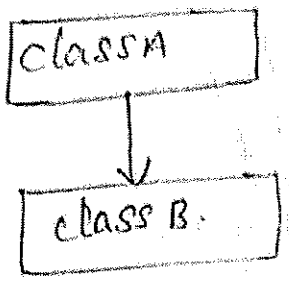
Types of Inheritance

1. Single Inheritance
2. multiple inheritance

3. multilevel inheritance
4. Hierarchical inheritance
5. Hybrid inheritance

1. single inheritance

In single inheritance one class Extends another class



class B Extends only class A. class A is super class and class B is subclass.

program

```
class Animal
{
void eat()
  {
  System.out.println(" eating ");
  }}
class Dog Extends Animal {
    void bark()
      { System.out.println(" barking");
      }}
    class Testinheritance
    {
    public static void main(String[] args)
    {
```

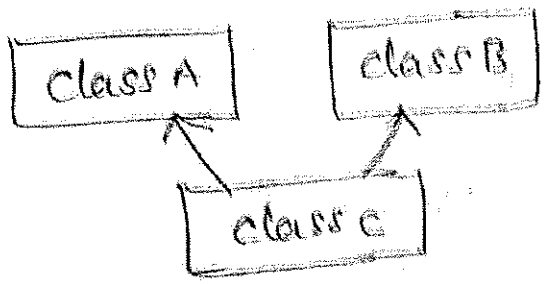
```
    Dog d = new Dog();
      d.bark();
      d.eat();
    }
}
```

output

barking
eating.

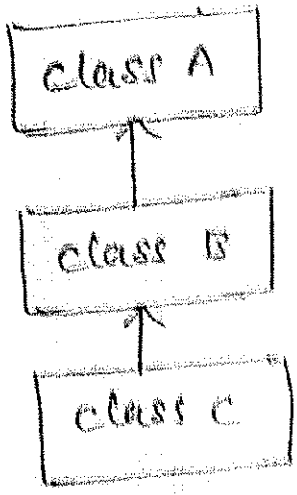
2. multiple inheritance

* when one class inherits multiple classes it is known as multiple inheritance
* multiple inheritance is not supported in java



3. multilevel inheritance

when there is a chain of inheritance is known as multilevel inheritance



program

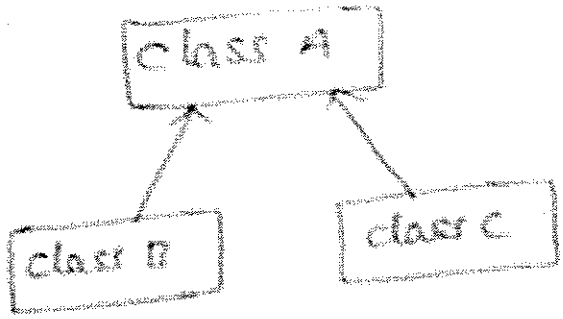
```
class Animal {
void eat()
{ system.out.println("eating"); }}
class Dog Extends Animal {
 void bark()
 { system.out.println("barking"); }
}
class babydog Extends Dog {
 void weep()
 { system.out.println("weeping"); }
}
class Test {
public static void main(string[] args)
{
  Babydog d=new BabyDog();
    d.weep();
    d.bark();
    d.eat();
  }
}
```

output

```
weeping
barking
eating.
```

4. Hierarchical inheritance

when two or more classes inherits a single class it is known as Hierarchical inheritance



program

```
class Animal
{
void eat()
{ system.out.println("eating"); }
}
class Dog Extends Animal {
void bark()
{ System.out.println("barking");
}
class Cat Extends Animal {
void meow()
{ system.out.println("meowing"); }
}
class Testinheritance {
public static void main (String[] args)
{
    Cat c = new Cat();
    c.meow();
```

```
        c.eat();
    }
}
```

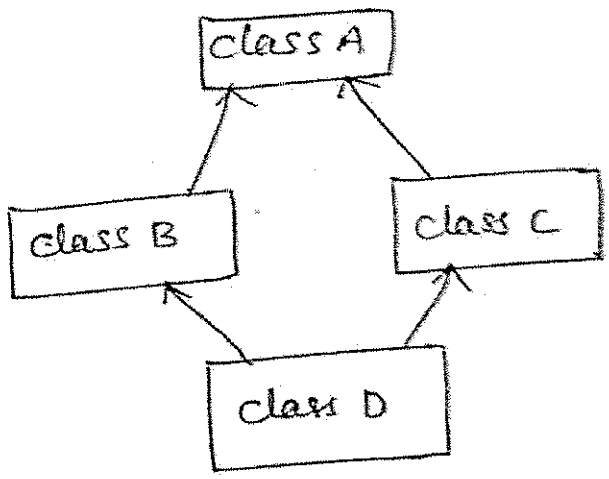
output

Meowing

eating.

5. Hybrid inheritance

In Java multiple and hybrid inheritance is supported through interface only.



Hybrid inheritance is a combination of both single and multiple inheritance

Inhibiting inheritance of class using final

* final is a keyword in java used for restricting some functionalities. we can declare variable, methods and classes with final keyword.

* During inheritance we much declare methods with final keyword to follow the same implementation throughtout all the derived classes.

* The main intention of making a method final that content of the method should not be changed by any outsider.

program

```
class Bike{
final void run()
{ system.out.println("running"); }
}
class Honda extends Bike{
public static void main (String[] args)
{
new Honda().run();
}
}
```

output

running

Application of keyword super

* The super keyword is reference variable which is used to refer immedate parent class.
* super can used to invoke immediate parent class method
* super() is used to invoke immediate parent class constructor.

program

```
class Animal          // superclass (parent)
{
  void animalsound()
  {
  System.out.println("The animal makes a sound");
  }
}
class Dog Extends Animal   // subclass [child]
  {
    void animalsound()
    {
    super.animalsound(); // call superclass method
    System.out.println("The dog says: bow");

  }
}
```

```
public class Main
{
public static void main(String[] args)
  {
  Animal mydog = new Dog(); //create Dog object
    mydog.animalsound();
  }
}
```

<u>output</u>

The animal makes a sound

The dog says: bow.

Dynamic Method dispatch (or) Runtime polymorphism

Dynamic method dispatch is the mechanism in which call to an overridden method is resolved at run time instead of compiletime.

Java uses the principles of superclass reefference variable can refer to a subclass object to resolve calls to overridden methods at run time

Advantages

* It allows Java to support overriding of methods

* It allows a class to define methods that will be shared by all its derived classes while also allowing these sub classes to define their specific

program

```
class Apple
{
 void display()
   {
   System.out.println("Inside Apples");
   }
}
```

```
class Banana Extends Apple
 {
  void display()      // overriding display()
   {
   System.out.println("Inside Banana");
   }
 }
 class Cherry Extends Apple
   {
    void display()     // overriding display()
     {
     System.out.println("Inside cherry's");
     }
   }
class Fruitdispatch
 {
 public static void main(String[] args)
   {
    Apple a= new Apple();     // object of Apple
    Banana b= new Banana();   // object of Banana
    Cherry c= new cherry();   // object of Cherry
    Apple ref;   // taking reefference of Apple
     ref = a;
```

```
    ref. display();
    ref = b;
    ref.display();
    ref = c;
    ref. display();
  }
}
```

<u>output</u>

```
Inside Apple
Inside Banana
Inside Cherry
```

This program creates one superclass 'Apple' and two subclasses of it (Banana and Cherry). subclass banana and cherry override the display() declare in Apple

Abstract class

A class which is declared with the abstract keyword is known as an abstract class

Abstraction

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

[OR]

It shows only essential things to the user and hides the internal details.

Example

Sending SMS where you type the text and send the message. you don't know the internal processing about the message delivery.

There are two ways to achieve abstraction in java

1. Abstract class
2. Interface

→ An abstract class must be declared with abstract keyword

→ It can not be instantiated

→ It can have abstract and non abstract methods

→ It can have constructors and static methods also

→ It can have final methods

program

```
abstractclass Bike
{
abstract void run();
}
Class Honda extends Bike
{
void run()
{
system.out.println("running safely");
}
publis static void main (String[] args)
{
Bike obj=new Honda();
obj.run();
}
}
```

output

running safely.

Interface

The interface in Java is a mechanism to achieve abstraction.

There are mainly three resons to use interface

1. It is used to achieve abstraction.
2. using interface to implement the multiple inheritance in Java.
3. It is used to achieve loose coupling

Declaration of interface

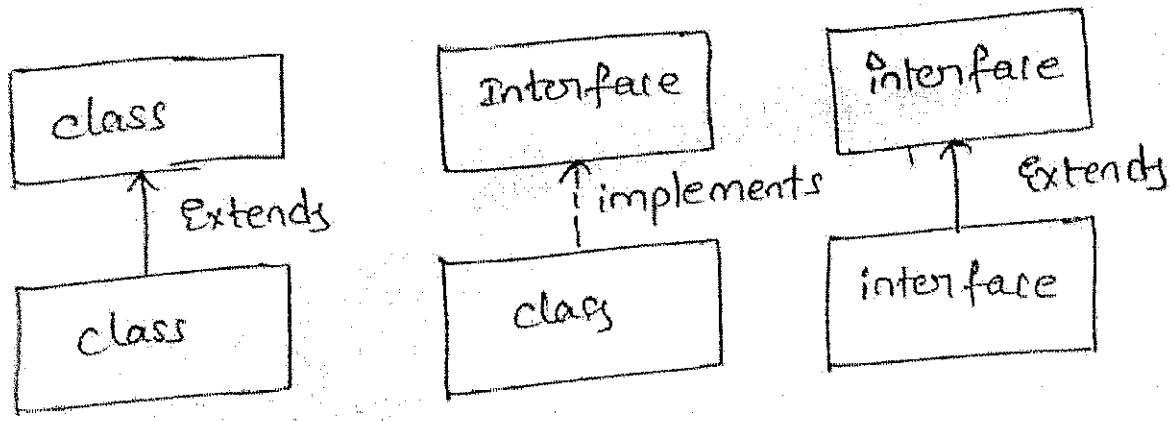
Syntax

```
interface   interface name
  {

  }
```

Relationship between classes and interfaces

a class extends another class, an interface extends another interface but a class implements an interface



program

```
interface printable
{
 void print();
}
Class A implements printable
{
public void print()
    {
    System.out.println(" Hello");
    }
  public static void main(String[] args)
    {
      A obj = new A();
        obj.print();
      }
    }
```

output

Hello

Multiple interfaces

A java class can only extend one parant class. multiple inheritance is not allowed. Interfaces are not classes however interface can extend more than one parant interface

Example if the hockey interface implements the both Sports and event.

program

```
interface Firstinterface
{
public void mymethod();  //interface method
}
interface Secondinterface
{
public void myothermethod();  //interface method
}
class Demo implements Firstinterface, Secondinterface
{
public void mymethod()
{
```

```
System.out.println(" Some text");
}
public void myothermethod()
{
System.out.println(" some other text");
}
}
class Main
{
public static void main(String[] args)
{
Demo myobj = new Demo();
myobj.mymethod();
myobj.myothermethod();
}
}
```

output

Some text

Some other text

Nested interface

* An interface declared within another interface is known as nested interface
* The nested interface must be referred by the outer interface or class but can't be accessed directly.

Syntax

```
interface interfacename
{
interface nestedinterfacename
  {
   - - -
  }
}
```

* nested interface are declared static implecitely (or) default.
* Nested interface declared within an interface must be public.

program

```
class Animal
{
```

```
interface Activity {
 void move();
 }
}
class Dog implements Animal.Activity {
  public void move()
  {
  System.out.println(" Dogs can walk and run");
  }
}
~~public~~ class Tester
{
public static void main(String[] args)
 {
  Dog dog=new Dog();
    dog.move();
  }
 }
```

<u>output</u>

Dogs can walk and run

Default methods in Interface

* Java provides a facility to create default methods inside the interface.
* methods which are defined inside the interface and tagged with default are known as default methods.
* These methods are non-abstract methods.

program

```
interface Sayable
{
default void say()   // default method
 {
System.out.println("Hello this is default method");
 }
void Saymore(String msg);  // Abstract method
 }
public class Defaultmethods implements Sayable
 {
  public void Saymore(String msg)
  {
```

```
    System.out.println(msg);
  }
  public static void main(String[] args)
  {
  Defaultmethods dm = new Defaultmethods();
    dm.say()    // calling default method
    dm.saymore(" workship");//calling abstract method

  }
}
```

output

Hello this is default method

workship.

Static methods in interface

* We can also define static methods inside the interface.
* static methods are used to define utility methods.
* They are declared using static keyword and will be loaded into the memory along with the interface

program

```
interface Myinterface
{
public void demo();
public static void display()
{
System.out.println("This is Static method");
}
}
public class InterfaceEx
{
public void demo()
{
```

```
System.out.println("Implements demo method");
}
public static void main (String[] args)
{
Interfaceex obj=new interfaceex();
    obj.demo();
  myinterface.display();
}
}
```

<u>output</u>

Implements demo method

This is the static method.

The difference between static and default method is - we can call the static method using name of the Interface while to call defaut methods using object of implementing class.

Functional interface

* An interface that contains Exactly one abstract method is known as functional interface
* It can have any number of defaut, Static methods but can contain only one abstract method.
* Functional interface is also known as Single abstract Method interface
* It is a new feature in Java which helps to achieve functional programming approach.

program

```
interface Sayable
{
 void say(String msg);
}
public class FunctionalInterface implements
                    Sayable
{
```

```
    public void say(String msg)
    {
    System.out.println(msg);
    }
  public static void main(String[] args)
  {
Functionalinterface  fe = new Functionalinterface();
    fe.say(" Hello there");
    }
  }
```

<u>output</u>

Hello there.

Annotations

Annotations in java are used to provide additional information

There are two types

1. Built in Annotation.
2. custom Annotation (or) user defined Annotations

Built in Annotations

There are several built in annotations in java.

* @override
* @ suppressWarnings
* @ Deprecated.

@Override This annotation assures that the subclass method is overriding the parent class method so compile time error occurs.

@ suppressWarnings

It is used to suppress warnings issued by the compiler.

@Deprecated

@Deprecated annotation marks that method is deprecated so compiler prints warnings

Custom Annotations

There are three types

1. Marker Annotation

An annotation that has no method is called marker annotations.

Ex
```
@interface myAnnotation { }
```

2. Single value Annotation

An Annotation that has one method is called single value Annotation

Ex
```
@ interface myannotation {
        int value();
}.
```

3. Multi value Annotation

An annotation that has more than one method is called multivalue Annotation.

Ex
```
@interface myannotation {
        int value1();
        String value2();
        String value3();
}
```

R19 JAVA - 4 UNIT

Exception Handling

The Exception handling in java is one of the powerful mechanisum to handle the runtime Errors so that normal flow of the application can be maintained

Example

```
statement 1;
statement 2;
statement 3; // Exception occurs
statement 4;
statement 5;
statement 6;
```

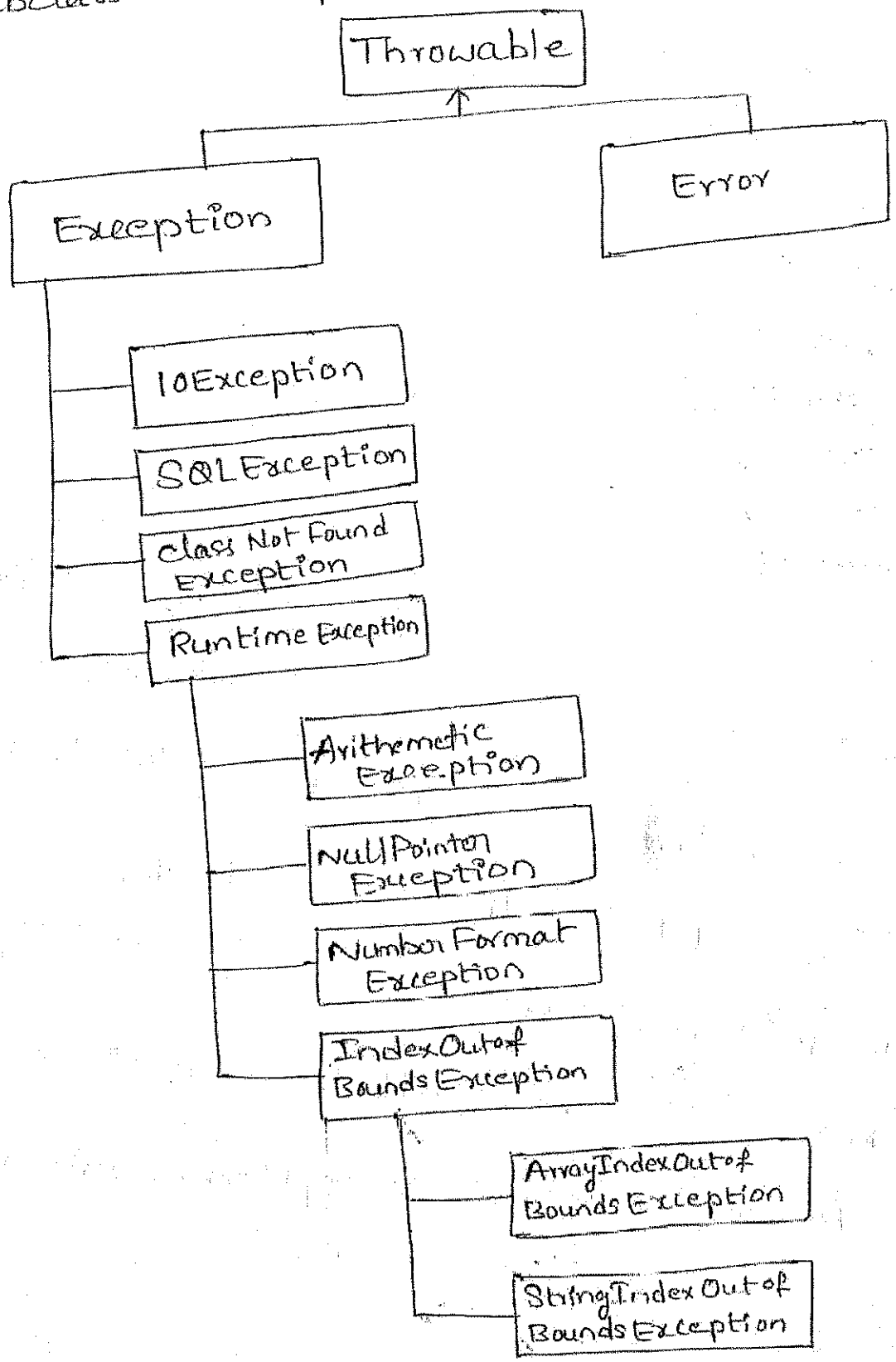
Suppose there are 6 statements in your program and there occurs an Exception at statement 3 the rest of the code will not be excuted i.e statement 4 to 6 will not Excuted,

If we perform Exception handling the rest of the statement will be Excuted.

That is why we use Exception handling in java.

Hierarchy of Exception classes

The java.lang. Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses: Exception and Error.



Try block

* Java try block is used to enclose the code that might throw an exception. It must be used within the method.
* If an Exception occurs at the particular statement of try block, the rest of the block code will not execute so it is recommended not to keeping the code in try block
* Java try block must be followed by either catch or finally block

Syntax for try-catch

```
try
{
 code that may throw an exception
 }
 catch (Exception class name)
  {

    }
```

Syntax of try-finally

```
try
{
 // code that may throw an exception
  }

finally
  {
    }
```

catch block

catch block is used to handle the Exception by declaring the type of Exception within the parameter

The declared Exception must be parent class Exception.

The catch block must be used after try block only.

We can use multiple catch block with a single try block.

problem without Exception handling

```
class TryCatch
{
public static void main(String[] args)
  {
   try
    {
     int data = 50/0; // may throw Exception.
    }
    catch(Exeption e)
     {
       System.out.println(e);
     }
```

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

output

```
java.lang.ArithmeticException: / by zero
   rest of the code
```

Resolve Exception in catch

```
class TryCatch
{
public static void main (String[] args)
 {
  int i=50;
  int j=0;
  int data;
  try
   {
   data= i/j; // may throw exception
   }
  catch(Exception e)
   {
```

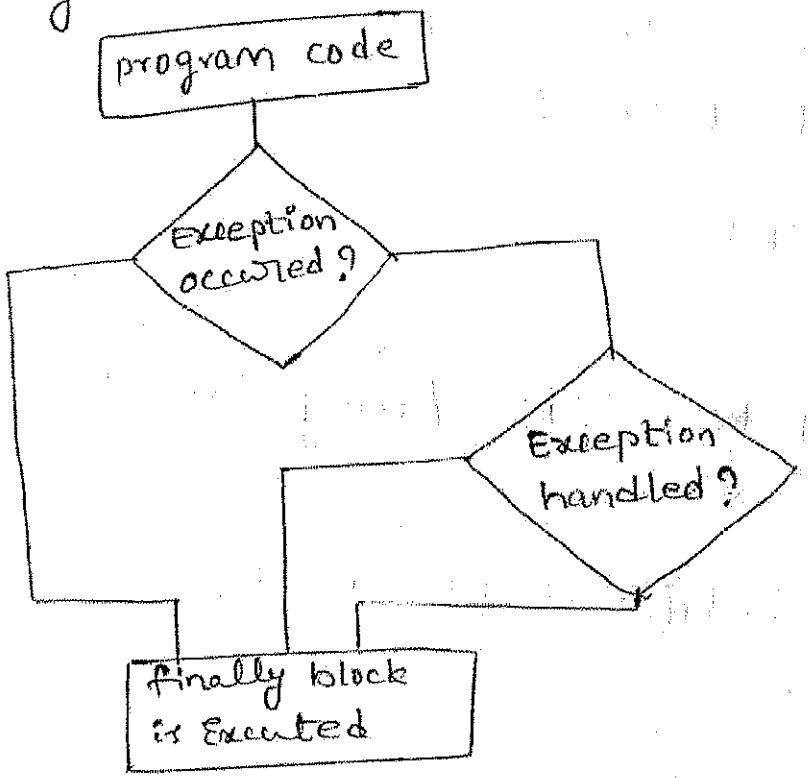
```
System.out.println (i/(j+2)); // resolving Exception
        }
    }
}
```

output

25.

Finally block

* Finally block is used to Execute important code such as closing connection
* Java finally block follows try or catch block.



Different cases where java finally block can be used.

case 1

Exception doesnot occur.

program

```
class Finally
{
public static void main (String[] args)
 {
  try
   {
    int data=25/5;
    System.out.println (data);
   }
  catch (NullPointerException e)
   {
    System.out.println (e);
   }
  finally
   {
   System.out.println (" finally block executed");
   }
```

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

output

5
finally block is Excuted
rest of the code

Case 2

Exception occurs and not handled.

program

```
class Finally
{
public static void main (String[] args)
 {
 try
  {
   int data = 25/0;
   System.out.println(data);
  }
  catch(NullPointerException e)
   {
```

```
  }
 finally
  {
  System.out.println("finally block is Executed");
  }
 System.out.println(" rest of the code");
  }
}
```

output

finally block is Executed.

Exception in thread main java.lang.ArithmeticException: / by zero

Case 3

Exception occurs and handled

program

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

```
try
 {
  int data = 25/0;
  System.out.println(data);
  }
 catch(ArithmeticException e)
  {
  System.out.println(e);
  }
 finally
  {
  System.out.println("finally block is Executed");
   }
  System.out.println("rest of the code");
  }
 }
```

<u>output</u>

```
Exception in thread main java.lang.ArithmeticException: / by zero
finally block is Executed
rest of the code.
```

throw

* The java throw keyword is used to explicitly throw an exception.
* We can throw either checked or unchecked exception in java
* The throw keyword is mainly used to throw custom exception

Syntax

```
throw Exception;
```

program

```
class Throw {
static void validate(int age)
{
if (age<18)
throw new ArithmeticException ("not valid");
else
System.out.println ("welcome to vote");
}
public static void main(String[] args)
{
validate(13);
System.out.println("rest of the code");
}
}
```

output Exception in thread main java.lang.ArithmeticException: not valid

throws

* throws keyword is used to declare an exception. It gives an information to the programmer that there may occur an exception
* Exception handling is mainly used the checked exceptions

Syntax

```
throws Exception classname
 {
   }
```

program

```
import java.io.IOException;
class throws
{
void m()throws IOException
 {
throw new IOException("device error");
 }
  void n()throws IOException
   {
    m();
   }
```

```
void p()
{
 try
  {
  n();
 }
catch(Exception e)
{
System.out.println("exception handled");

}
}
public static void main(String args[])
{
 Throws obj = new Throws();
  obj.p();
 System.out.println("normal flow");
 }
}
```

<u>output</u>

```
Exception handled

normal flow.
```

Multiple catch caluses

* A try block can be followed by one or more catch blocks.
* Each catch block must contains a different exception handler.
* At a time only one exception occurs and at a time only one catch block is executed.

program

```
class Multiplecatch
{
public static void main(String[] args)
 {
  try
  {
   int a[] = new int[5];
   a[5] = 30/0;
  }
  catch (ArithmeticException e)
    {
   System.out.println("Arithmetic Exception occur");
   }
  catch (ArrayIndexOutOfBoundsException e)
   {
System.out.println("ArrayIndexOutOfBounds Exception occurs");
   }
```

```
catch (Exception e)
 {
 System.out.println("parent Exception occurs");
 }
 System.out.println("rest of the code");
 }
}
```

Output

Arithmetic Exception occurs
rest of the code

class Throwable

* Throwable class is the superclass of every error and Exception in the java language
* Throwable class is the root class of java Exception heirarchy and is inherited by two subclasses

1. Exception
2. Error

* throwable class implements Serializable interface and direct known classes

to throwable are error and Exception.

throwable constructors

1. Throwable() – It is a non parameterized constructor which constructs a new throwable with null

2. Throwable(String message)

It is a parameterized constructor with specific detailed message

3. Throwable(String message, Throwable cause)

It is a parameterized constructor with the specific detailed message and a cause

4. Throwable(Throwable cause)

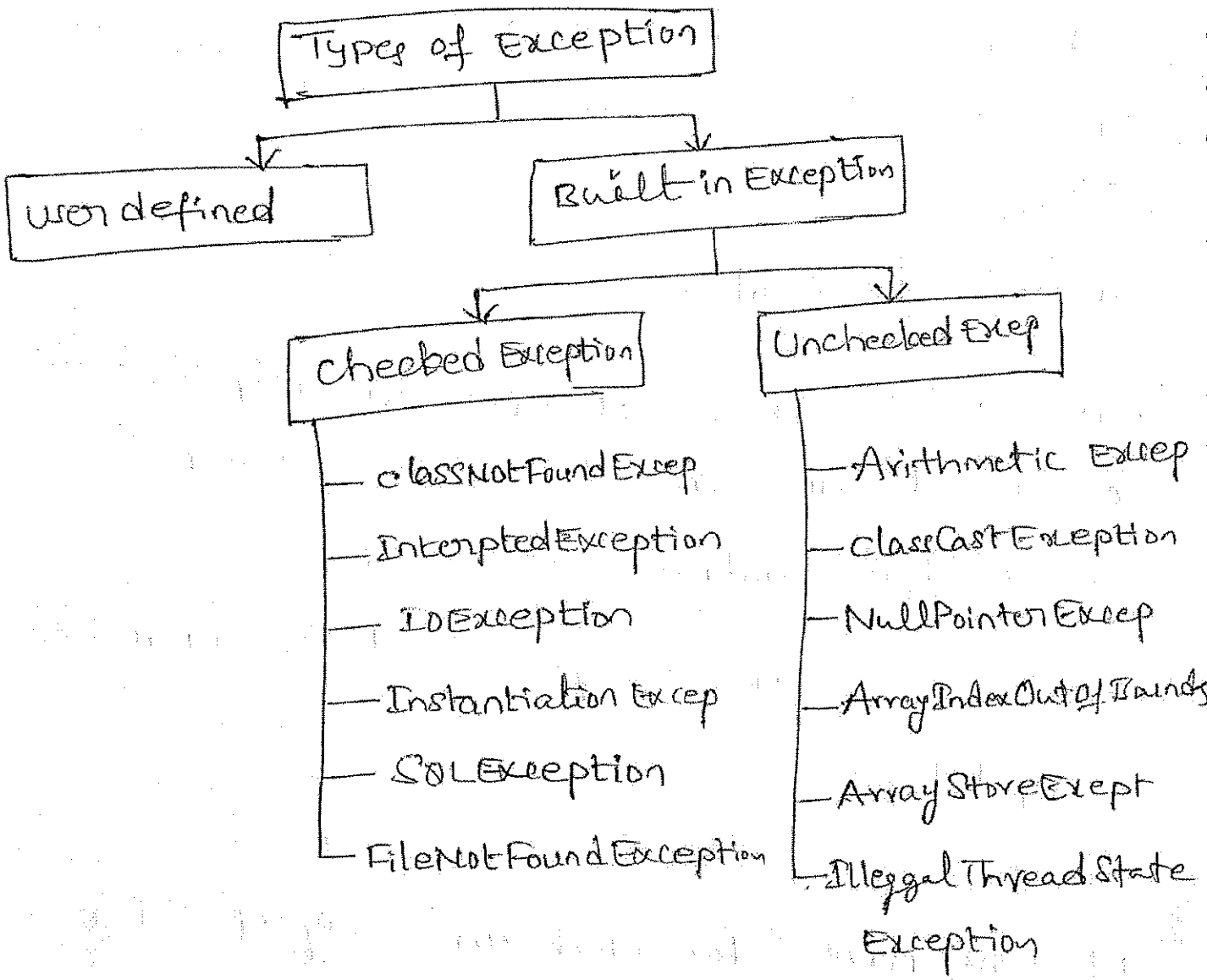
It is a parameterized constructor with specific cause.

checked Exception

In Java Exception is an event that occurs during the execution of a program

Exception can be categorized into two ways

1. Built in Exceptions
   * Checked Exception
   * Unchecked Exception
2. User defined Exception



Built in Exception

Exceptions that are already available in Java libraries are reffered as built in exeption.

Checked Exeeption

* Checked Exeeptions are called compile time Exeeptions because these Exeeptions are checked at compile time by the compiler
* The programmer should handle the Exeeption otherwise the system has shown a compilation error.

Unchecked Exeeptions

* An unchecked Exeeptions is an Exeeption that occurs at the time of Execution. These are also called Runtime Exeeptions.
* Runtime errors are ignored at the time of compilation.

Example

If you have declared an array of size

5 in your program and trying to call the 6th element of an array then the ArrayIndex OutOfBoundsException occurs.

program

```
class Uncheked
{
public static void main (String[] args)
 {
 int num[] = {1,2,3,4};
 System.out.println (num[5]);
 }
}
```

output

```
Exception in thread main java.lang.ArrayIndexOutof
        BoundsException: 5
```

Custom Exception

* If you are creating your own Exceptions that is known as custom Exception or user-defined Exception
* Java custom Exceptions are used to customize the Exception according to user need.
* By the help of custom Exception you can have your own exceptions and message.

program

```
class MyException extends Exception
 {
  String str1;
  MyException(String str2)
   {
     str1 = str2;
   }
  public string tostring()
   {
   return ("myException occured:" + str1);
   }
}
```

```
class Example
{
public static void main(String[] args)
  {
   try
    {
System.out.println("starting of try block");
  throw new MyExeption("This is my error message");
  }
catch (MyExeption exp)
  {
System.out.println("catch block");
System.out.println(exp);
  }
 }
}
```

<u>output</u>

```
starting of try block
catch block
MyException occurred: This is my error message
```

Nested try and catch

The try block within a try block is known as nested try block in Java.

why use nested try

Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases exception handlers have to be nested.

Syntax

```
try
 {
  statement 1;
  statement 2;
     try
      {
       statement 1;
       statement 2;
       }
      catch (Exception e)
      {
          }
      }
```

```
catch (Exception e)
{
    }
```

program

```
class Nestedtry
 {
 public static void main (String[] args)
  {
   try
    {
    try
      {
     System.out.println("going to divide");
     int b= 39/0;
   }
  catch (ArithmeticException e)
   {
   System.out.println (e);
   }
   try
    {
    int a[]= new int[5];
```

```
    a[5] = 4;
  }
catch(ArrayIndexOutOfBoundsException e)
  {
  System.out.println(e);
  }
  System.out.println(" other statement");
  }
 catch (Exception e)
  {
  System.out.println(" handeled");
  }
  System.out.println("normal flow");
  }
 }
```

<u>output</u>

```
going to divide
java.lang.ArithmeticException: I by zero
java.lang.ArrayIndexOutOfBoundsException: 5
 other statement
 normal flow.
```

Difference Between checked and unchecked.

checked	unchecked
* checked Exception occur at compile time	* unchecked Exception occur at runtime
* The compiler checks a checked Exception	* The compiler does not check these types of Exceptions
* They are subclass of Exception class	* They are not part of the Exception class
* JVM needs the Exception to catch and handle	* JVM does not require Exception to catch and handle
* Example of checked Exceptin File not found Exception Interrupted Exception class not found Exception	Example of unchecked Arithmetic Exception Nullpointer Exception ArrayIndexOutofBounds Exceptions

try with resources

* try with resource statement is a try statement that declares one or more resource
* The try with resource statement ensures that each resource is closed at the end of the statement execution

program

```
import java.io.FileOutputStream;
class Trywithresource
{
public static void main (String[] args)
{
try(FileOutputStream fileoutputstream=new FileOutput
                                    Stream("hi"))

  {
    String msg = "welcome";
    byte byteArray[] = msg.getBytes();
    fileOutputStream.write(byteArray);
  System.out.println("Message written to file");
  }
  catch(Exception e)
    {system.out.println(Exception);
```

```
    }
  }
}
```

<u>output</u>

message written to file

<u>output of file</u>

welcome

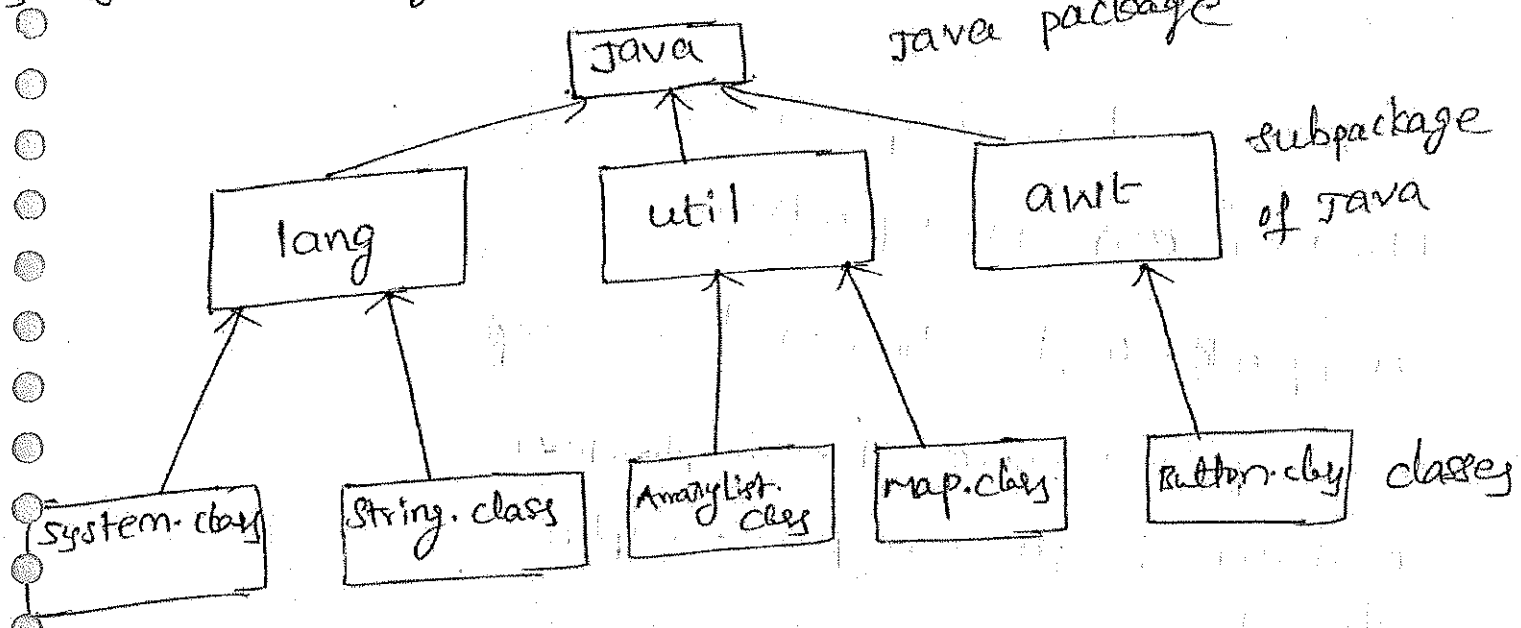
Package

* A Java package is a group of similar types of classes, interfaces and sub-packages
* package in Java can be categorized in two types
1. Built in package
2. User defined package

There are many built in package such as lang, awt, javax, swing, io, util and sql etc

Advantages

1. Java package is used to categorize the classes and interface so that can be easily maintained.
2. Java package provides access protection
3. Java package ~~provides~~ removes naming collision.



Simple program

```
package mypack;
   class Simple
{
public static void main (String[] args)
 {
 System.out.println ("welcome to package");

 }
}
```

How to Compile java package

javac -d directory javafilename

Example

javac -d . Simple.java

How to run java package

mypack.simple to run the class

To compile : javac -d . simple.java

To Run : java mypack.simple

output

welcome to package

How to access package from another package

There are three ways the package access from outside the package

1. import package.*;
2. import package.classname;
3. fully qualified name.

1. using packagename.*;

If you use package.* then all the classes and interfacy of this package will be accessible but not subpackages

```
//save by A.java
 package pack;
  class A
   {
   public void msg()
   {
   System.out.println("Hello");
   }
  }

// save by B.java
   package mypack;
   import pack.*;
   class B
   {
```

```
public static void main(String[] args)
{
  A obj = new A();
  obj.msg();
}
}
```

output

Hello

2. using packagename.classname

If you import package.classname then only declared class of this package will be accessible

```
// save by A.java
package pack;
  class A
public void msg()
{
System.out.println("Hello");
}
}
// save by B.java
  package mypack;
  import pack.A;
```

```
class B
{
public static void main(String[] args)
 {
   A obj=new A();
     obj.msg();
   }
 }
```

Output

Hello.

3. Using fully qualified name

* If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import
* It is generally used when two packages have same classname
Ex java.util and java.sql packages contains data class.

```
// save by A.java
package pack;
   class A
{
public void msg()
{
System.out.println("Hello");
}
}
```

```
// save by B.Java
package mypack;
class B.
{
public static void main (String[] args )
{
pack.A obj = new pack.A(); // using fully qualified name
    obj.msg();
  }
}
```

<u>output</u>

Hello

Access control in packages

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

There are four types of Java access modifiers

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

1. private

The private access modifier is accessible only within the class

We have created two classes A and Simple. A class contains private data member and method. We accessing these private members from outside the class then compile time error occur.

```
class A {
private int data=40;
private void msg()
  {
  System.out.println("Hello");
  }
}
```

```
class simple
{
public static void main(String[] args)
{
 A obj = new A();
System.out.println(obj.data); // compile time error
   obj.msg(); // compile time error
 }
}
```

2. Default

* If you don't use any modifier then it is treated as default modifier
* The default modifier is accessible only within package

```
// save by A.java
package pack;
 class A
{
void msg()
{
System.out.println("Hello");
 }
}
```

```
// save by B.java.
 package mypack;
 import pack.*;
 class B.
 {
 public static void main (String[] args)
 {
  A obj=new A (); // compile time error.
   obj.msg(); // compile time error
 }
}
```

class A and its method msg() is default so it can not be accessed from outside the package

3. protected

It is accessible within package and outside the package but through inheritance only.

```
// save by A.java
  package pack;
   class A
 {
 protected void msg()
 {
 System.out.println(" Hello");
 }
}
```

```
// save by B.java
package mypack;
import pack.*;
class B extends A
{
public static void main(String[] args)
{
  B obj = new B();
   obj.msg();
  }
}
```

output

Hello

4. public

It is accessible everywhere. It has the widest scope among all other modifiers

```
// save by A.java
 package pack;
  class A
 {
 public void msg()
  {
  System.out.println("Hello");
  }
}
```

```
//save by B.java
package mypack;
Import pack.*;
Class B
{
public static void main(String[] args)
 {
  A obj =new A();
     obj.msg();
  }
 }
```

output

Hello

Java.lang package and its classes

Java.lang package provides classes that are fundamental to the design of java programming language

Following are the important classes in Java.lang

1. Boolean – Boolean class wraps primitive type boolean in an object.
2. Byte – Byte class wraps primitive type of byte in an object
3. Float – Float class wraps primitive type float in an object
4. Double – Double class wraps primitive types double in an object
5. Integer – Integer class wraps primitive type integer in an object

6. Long - Long class wraps primitive type long in an object
7. short - short class wraps primitive type short in an object
8. Thread - A thread is execution of program.
9. ThreadGroup - It represents set of threads
10. ThreadLocal - This class provides thread local variable
11. String - The string class represents character strings.
12. ProcessBuilder - This class is used to create operating system processes
13. package - Implementation and specific function of a Java package
14. object - class object is root of the class hierarchy.
15. math - math class containing methods for performing basic numeric operations such as, exponential, logarithm, square root and trigonometric
16. Enum - This is common base class of all Java language enumeration type

Math class

Java math class provides several methods to work on math calculations like min(), max(), avg(), sin(), cos(), tan(), round(), ceil(), floor(), abs() etc.

For arithmetic operation like increment, decrement, divide, absoulte value occurs only with a specific minimum or maximum value

program

```
class Math
{
public static void main(String[] args)
{
  double x=28;
  double y=4;
System.out.println("Maximum number of x and y is:"
                +Math.max(x,y));
System.out.println("squre root of y is:"+Math.sqrt(y));
System.out.println("power of x and y is:"+Math.pow(x,y));
System.out.println("logarithm of x is:"+math.log(x));
System.out.println("Exp of a is:"+math.exp(x));
```

```
System.out.println ("expm1 of a is:" + Math.expm1(a));
                    // [a power of 2] - 1
    }
}
```

<u>Outputs</u>

```
maximum number of x any y is: 28.0
square root of y is: 2.0
power of x and y is: 614656.0
logorithm of x is: 3.332204510175204
exp of a is: 1.4462570642947 5E12
expm1 of a is: 1.4462570 6429147 5E12
```

Wrapper classes

* The wrapper classes in ~~provi~~ Java provides mechanisum to convert primitive into Object and object into primitive
* J2SE 5.0 autoboxing and unboxing feature convert primitives into objects and objects into primitives automatically.
* The automatic conversion of primitive into object is known as autoboxing
* The automatic conversion of objects into primitive is known as unboxing

List of eight wrapper classes

primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	short
int	Integer
long	Long
float	Float
double	Double

program

```
class Wrapper
{
public static void main (String[] args)
{
 int a=20;
Integer i = Integer.valueOf(a);
Integer j=a;

System.out.println (a+" "+i+" "+j);

 }
}
```

output

20 20 20

Autoboxing

* The automatic conversion of primitive data type into its corresponding wrapper class [objects] is known as autoboxing.

* For Example byte to Byte, char to character, int to Integer, long to Long, float to Float, boolean to Boolean, double to Double and short to Short.

* Java we need to use the valueOf() method of wrapper classes to convert the primitive into objects.

program

Autoboxing Example int to Integer.

```
class WrapperInt
{
```

```
public static void main (String[] args)
{
  int a = 20;
  Integer i = Integer.valueOf(a);
  Integer j = a;

  System.out.println (a+" "+i+" "+j);

  }

}
```

<u>output</u>

20 20 20.

Date It represents a specic instant time with milliseconds

Dictionary It represents abstract parent on any class

Simple time zone It represents a time zone for use with a Gregorian calendar.

Stack It represents last in frist out (LIFO) stack of objects

Timer A facility for thread to schedul tasks for future excution.

Interfaces :-

collection<E> -- It is root interface in collection hierarchy

comparator<T> This is a comparision function.

Deque<E> Elements insertion and removal at both ends.

Enumeration<E> Enumeration interface generates a series of elements one at a time

Iterator<E> This is a iterator over a collection

List<E> This is an ordered collection.

ListIterator<E> programmer to traverse the list either direction, modify the list during iteration

map<K,V> maps keys to the values

map.Entry<K,V> This is map entry (key value pair)

Queue<E> This is a collection of elements prior to processing.

Set<E> This is a collection that contains no duplicate elements

observer when it wants to be informed of changes in observable objects

Unboxing

* The automatic conversion of wrapper type into its corresponding primitives type is known as unboxing
* It is the reverse process of autoboxing.
* In Java we need to use of intValue() method of wrapper classes to convert the wrapper type into primities

program

Unboxing Example of Integer to int

```
class WrapperUnboxing
{
public static void main (String[] args)
{
Integer a= new Integer(3);
int i=a.intValue();
int j=a;
```

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

output

3 3 3.

Java Util classes and interfaces

classes

AbstractCollection – implementation of collection interface

AbstractList – To minimise the effort required to implements this interfale

AbstractQueue Implementation of some Queue operations

Calender methods for converting between a specic instant time and set calender fields such as YEAR, MONTH, DAY_OF_MONTH, HOUR etc

Currency – Represents a currency

Formatter class

* The formatter is a built in class in Java used for layout justification and alignment
* Common formats for numeric, string and date/time and local specific output in Java programming.

The formatter class in Java the following constructors

1. Formatters()

It creates a new formatter.

2. Formatter (Appendable a)

It creates a new formatter with specified destination.

3. Formatter (Appendable a, Local l)

It creates a new formatter with specified destination and local.

4. Formatter (File file)

It creates a new formatter with specified file

5. Formatter (File file, String charset)

It creates a new formatter with specified file and charset.

6. Formatter (File file, String charset, Local l)

It creates a new formatter with specified file, charset and local.

7. Formatter (OutputStream os)

Created with specified output stream.

8. Formatter (OutputStream os, String charset)

It created with outputput stream and charset.

9. Formatter (outputStream os, String charset, local l)

It created with specified outputStream, charset and local.

10. Formatter (PrintStream ps)

It created with specified printStream.

11. Formatter (String filename)

It creates with specified file name

12. Formatter (String filename, String charset)

Created with specified filename and charset.

13. Formatter (String filename, String charset, local l)

It created with specified filename, charset and local

Random class

Java Random class is used to generate a stream of pseudorandom numbers.

Methods

doubles() Return unlimited stream of pseudorandom double values.

ints() Return unlimited pseudorandom int values

longs() Return unlimited pseudorandom long values

next() Generates next pseudorandom number.

nextBoolean()

It return next pseudorandom boolean value from the random generator sequence

nextByte()

Generates random bytes and puts them into a specified byte Array.

nextDouble()

Return double value between 0.0 and 1.0 from generator sequence

nextFloat()

Return float value between 0.0 and 1.0 from generator sequence

nextGaussian()

Return double value with mean 0.0 and standard deviation 1.0 from generator sequence

nextInt()

Return int value generated from generator sequence

nextLong()

Return long value from generator sequence

program

```
import java.util.Random;
class RandomExample
{
public static void main(String[] args)
{
Random random = new Random();
System.out.println(" long value:"+random.longs());
boolean val = random.nextBoolean();
System.out.println("Random boolean value:" + val);
byte[] bytes = new byte[10];
random.nextBytes(bytes);
System.out.print(" Random bytes = (" );
for (int i=0; i< bytes.length; i++)
```

```
    {
    System.out.printf(" %d", bytes[i]);
    }
    System.out.print(")");
    }
}
```

Output

```
Longs value: java.util.stream.longpipeline$Head@14ae5a5

Random boolean value: True

Random bytes = (57 77 8 67 -122 -71 -79 -62 53 19)
```

DATE ANT TIME

The java.time, java.util, java.sql and java.text packages contains classes for representing date and time

The java.time package contains java 8 date and time classes

1. java.util.Date class
2. java.sql.Date class

3. java.util.Calendar class
4. java.util.GregorianCalendar class
5. java.util.TimeZone class
6. java.sql.Time class
7. java.sql.Timestamp class

Formatting Date and Time

We can format date and time in java by the use of following classes.

(i) java.text.DateFormat class
(ii) java.text.SimpleDateFormat class

program

```
import java.time.LocalDate;
class LocalDate
{
public static void main(String[] args)
{
LocalDate date=LocalDate.now();
LocalDate yesterday = date.minusDays(1);
LocalDate tomorrow = yesterday.plusDays(2);
```

```
System.out.println(" Today date:"+date);
System.out.println("yesterday date:"+yesterday);
System.out.println(" Tomorrow date:"+tomorrow);
  }
}
```

Output

```
Today date : 2017-01-13
yesterday date : 2017-01-12
Tommorow date: 2017-01-14.
```

Instant class

* Java instant class is used to represent the specific moment on the time line.
It inherits the object class and implements the Comparable interface

Declaration

public final class Instant Extends object

implements Temporal, TemporalAdjuster, Serializable

Methods

* Temporal adjustInto(Temporal temporal) - It is used to adjust specified temporal object to have this instant
* int get(TemporalField field) – To get the value of the specified field from this instant an int.
* boolean isSupported(TemporalField field) — To check if the specified field is supported.
* Instant minus(TemporalAmount amountToSubtract) – to return a copy of this instant with the specified amount subtracted.
* Static Instant parse(CharSequence text) – To obtain instance of instant from a text string.
* Static Instant now() – To obtain current instant from the system clock.
* Instant plus(TemporalAmount amountToAdd) – To return a copy of this instant with the specified amount added.

Instant program - ①

```
import java.time.Instant;
class InstantExample
{
public static void main(String[] args)
 {
  Instant inst=Instant.parse("2017-02-03T10:37:30:002");

  System.out.println(inst);

 }
}
```

output

2017-02-03T10:37:30Z

Example - ②

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

```
Instant instant = Instant.parse("2017-02-03T10:25:30.002");
    instant = instant.minus(Duration.ofDays(125));
    System.out.println(instant);
  }
}
```

Output

```
2016-10-01T11:25:30Z
```

Temporal Adjusters class

Temporal Adjusters class in Java provides Adjusters which are key tool for modifying temporal object Example and adjuster that sets the date like Second saturday of the month or Next Tuesday or last day of the month.

Methods

1. dayOfWeekInMonth (int ordinal, DayOfWeek) - It return the day of week in month.
2. firstDayOfMonth() - It return a new date set to the first day of the current month.
3. firstDayOfNextMonth() - It return a new date set to the first day of the next year
4. firstDayOfYear() It return a new date set to the first day of the current year.
5. lastDayOfYear()
   which return a new date set to the last day of current year

FristInMonth(DayOfWeek dayOfWeek) – which return a new date in the same month with the first matching day of week.

LastInMonth (DayOfWeek dayOfweek) – which return a new date in the same month with the last matching day of week.

lastDayOfMonth()

which return a new date set to the last day of the current month.

next (DayOfWeek dayOfweek)

which adjusts the date to frist occurence of the specified day of week after date being adjusted

previous (DayOfWeek dayOfweek)

which adjust the date to the first occurence of the specified day of week before date being adjusted unless it is already on that day in which case the same object is returned.

Program

```
import java.time.LocalDate;
import java.time.temporal.TemporalAdjusters;
import java.time.DayOfWeek;

class JavaTester
{
public static void main(String[] args)
 {
 JavaTester j = new JavaTester();
     j.testAdjusters();
 }
 public void testAdjusters()
 {
 LocalDate date1 = LocalDate.now();
 System.out.println("current date:" + date1);
 LocalDate nextTuesday = date1.with(TemporalAdjusters.
        next(DayOfWeek.TUESDAY));
   System.out.println("Next Tuesday on:" + nextTuesday);

  LocalDate firstInYear = LocalDate.of(date1.getYear(),
        date1.getMonth(), 1);
   LocalDate secondSaturday = firstInYear.with(TemporalAd
   justers.nextOrSame(DayOfWeek.SATURDAY)).with(
```

```
TemporalAdjusters.next(DayOfWeek.SATURDAY));

System.out.println("Second saturday on:" + Second
        Saturday);

    }

}
```

output

```
Current date : 2014-12-10

Next Tuesday on : 2014-12-16

Second Saturday on : 2014-12-13.
```

Java String Handling

* String is an object that represent sequence of characters.
* It is probably the most commonly used class in Java library.
* String objects are immutable that means once a string object is created it can not be changed.
* Java String class implements Serializable, comparable and charSequence

Creating String object

String can be created in number of ways

1. using a String literal

String literal is a simple string enclosed in double quotes." ".

program

```
class Demo
{
public static void main(String[] args)
 {
 String s1 = "Hello Java";
 System.out.println(s1);
 }
}
```

output

Hello Java

2. using new keyword

We can create a new String object using new operator that allocates memory for the object

program

```
class Demo
```

```
{
public static void main(String[] args)
{
String s1 = new String("Hello Java");
System.out.println(s1);
}
}
```

output

Hello java

Concatenating String

There are two methods to concatenate two or more strings.

1. using concat() method
2. using + operator

1. using concat() method

concat() is used to add two or more string

into a single object

program

```
class demo
{
public static void main(String[] args)
{
String s="Hello",
String str="java",
String str1 = s.concat(str);
System.out.println(str1);
}
}
```

output

Hellojava.

2. using + operator

java uses "+" operator to concatenate two String objects into single one

* It can also concatenate numeric value with 5-3 String object.

program

```
class Demo
{
public static void main(String[] args)
{
String s="Hello";
String str="JAVA";
String str1= s+str;
String str2="JAVA"+11;
System.out.println(str1);
System.out.println(str2);

}
}
```

output

```
HelloJAVA
JAVA11
```

String Concatenation

Interface char Sequence

* The Java API includes a useful interface called charSequence
* All three of these classes - String, StringBuilder and StringBuffer implements this interface
* Towards end of several methods of String, StringBuilder, StringBuffer classes use charsequence as a parameter type
* char charAt(int) - returns the character at the specified position.
* int length() - Returns length of the sequence
* subsequence(int start, int end) - Returns substring indicated by start and end parameters
* toString() - Returns a string representation of the sequence

Methods for extracting character from String

The String class provides a number of ways in which characters can be extracted from a string.

1. charAt()
2. getChars()
3. getBytes()
4. toCharArray()

1. charAt(int index)

To Extract a single character from a string you can reffer directly to an individual character via the charAt() method.

program

```
class Demo
{
public static void main (String[] args)
{
String str = " welcome to String handling guide",
char ch1 = str.charAt(0);
char ch2 = str.charAt(5);
char ch3 = str.charAt(11);
char ch4 = str.charAt (20);
```

```
System.out.println("character at 0 index is:" +ch1);
System.out.println("character at 5th index is:" +ch2);
System.out.println("character at 11th index is:" +ch3);
System.out.println("character at 20th index is:" +ch4);

  }
}
```

output

character at 0 index is : W.
character at 5 index is : m
character at 11th index is : S
character at 20th index is : n.

2. getchars()

If you need to Extract more than one character at a time you can use getchars() method.

program

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

```
String s= "This is a demo of the getchars";
    int start = 0;
    int end = 4;
  char buf[] = new char[end - start];
   s.getChars (start, end, buf, 0);
   System.out.println(buf);
  }
}
```

<u>output</u>

This

3. <u>getBytes()</u>

This is an alternative to getchars() that stores the characters in an array of bytes.

There are four versions of getBytes()

1. <u>byte[] getBytes()</u> - Encoding this string into a sequence of bytes using default charset.

2. <u>byte[] getBytes(Charset charset)</u> - Encode this String into a sequence of bytes using given charset.

3. void getBytes(int srcBegin, int srcEnd, byte[] dst, int dstBegin)

   - Deprecated.

(iv) byte[] getBytes(String charsetName) — Encodes this String into a sequence of bytes using named charset.

4. toCharArray()

This method converts this string to a new character array.

program

```
class Demo
{
public static void main (String[] args)
{
String str = "java"
char[] characters = str.toCharArray();
for (char c : characters)
{
System.out.println(c);
}
}
}
```

output

```
j
a
v
a
```

Methods for comparision of String

We can compare string in Java on the basis of content and reference

There are three ways to compare string in Java.

1. By equals() method.
2. By == operator.
3. By compareTo() method.

1. By equals() method

* The String equals() method compares the original content of the string
* It compares values of string for equality

String class provides two methods

(i) public boolean equals(object another) – compare this string to the specified object.

(ii) public boolean equalsIgnoreCase(String another) – compares this string to another string

program

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

```
String s1 = "mahi";
String s2 = "mahi";
String s3 = new String("mahi");
String s4 = "Ram";
System.out.println(s1.equals(s2));
System.out.println(s1.equals(s3));
System.out.println(s1.equals(s4));
}
}
```

output

true

true

false

2. String by == operator

The == operator compares references not values

program

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

```
String s1 = "mahi";
String s2 = "mahi";
String s3 = new String("mahi");
System.out.println(s1 == s2);
System.out.println(s1 == s3);
}
}
```

output

true

false

3. By compareTo() method

compareTo() method compare values lexicographically and returns an integer value that describe if frist string is less than, equal to or grater than second string.

suppose s1 and s2 are two string variable if

s1 == s2 : 0

s1 > s2 : positive value

s1 < s2 : negative value

program

```
class Demo
{
public static void main (String[] args)
{
String s1 = " sachin";
String s2 = " sachin";
String s3 = " ratan";
System.out.println (s1.compareTo(s2));
System.out.println (s1.compareTo(s3));
System.out.println (s3.compareTo(s1));
}
}
```

output

```
0
1
-1
```

Methods for modifying a String

We are very clear that Strings are immutable i.e once created they can not be changed Thus to modify them we use the following methods.

1. Substring()
2. concat()
3. replace()
4. trim().

1 Substring()

A part of string is called substring. You can get substring from the given string by one of the two methods.

(i) public String Substring(int startindex) – This method return new string object containing the substring of the given string from specifed start index

(ii) public String Substring(int startindex, int endindex); – This method return substring of given string from specified startindex to endindex.

program

```
class Demo
{
```

```
public static void main(String[] args)
{
 String s = "SachinTendulkar";
 System.out.println(s.substring(6));
 System.out.println(s.substring(0,6));

 }
}
```

output

Tendulkar
Sachin

2. concat()

concat() method combines specified string at the end of the string. It returns combined string

3. Replace()

replace() method return a string replacing all the old char to new char.

There are two types of replace methods in JAVA

1. public String replace(char oldchar, char newChar)

program

```
class Replace
{
public static void main(String[] args)
{
String S1 = " Java programming";

String replaceString = S1.replace('a','e');

 System.out.println(replaceString);

}
}
```

output

Jeve progremming

ii) public String replace (CharSequence target, CharSequence replacement)

program

```
class Demo
{
public static void main(String[] args)
{
String S1 = "my name is Java my place is Andhra";

String replaceString = S1.replace("is", "was");
```

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

output

my name was java my place was Andhra

4. trim()

trim() method eliminates leading and trailing spaces

program

```
class Demo
{
public static void main(String[] args)
{
 String s1 = " hello java";
 System.out.println(s1+" programming");
 System.out.println(s1.trim()+" programming");
 }
}
```

output

hello java programming
hello javaprogramming.

String Buffer class

Java String Buffer class is used to create mutable [mutable means it can be changed]

Methods

1. StringBuffer append()

The append method concatenates the given argument with the string.

program

```
class StringBuffer
{
public static void main(String[] args)
{
 StringBuffer sb=new StringBuffer("Hello");
  sb.append("Java");
  System.out.println(sb);
 }
}
```

output

Hello Java

2. StringBuffer insert()

The insert() method given string with the string at

at the given position.

program

```
class StringBufferInsert
{
public static void main(String[] args)
{
StringBufferInsert sbi = new StringBufferInsert("Hello");
sbi.insert(1,"Java");
System.out.println(sbi);
}
}
```

output

HJavaello.

3. StringBuffer replace()

The replace() given string from the specified begin index and end index

program

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

```
StringBufferreplace sbr = new StringBufferreplace ("Hello");
  sbr.replace (1,3, "java");
 System.out.println(sbr);
  }
 }
```

output

H java lo

4. StringBuffer delete()

The delete() method delete the string from the specified beginIndex to end index.

program

```
class StringBufferdelete
{
public static void main (String[] args)
{
StringBufferDelete sbd = new StringBufferDelete ("Hello");
  sbd.delete (1,3);
 System.out.println (sbd);
}
}
```

output

Hlo

5. StringBuffer reverse()

The reverse() method of reverse the current string.

program

```
class StringBufferReverse
{
public static void main(String[] args)
{
StringBufferReverse sbr=new StringBufferReverse("Hello");
  sbr.reverse();
 System.out.println(sbr);
 }
}
```

Output

olleH.

6. StringBuffer capacity()

The capacity() method of StringBuffer class returns the current capacity of the buffer.

The defaulit capacity of the buffer is 16.

If the number of character increases from its current capacity, it increase capacity by (oldcapacity * 2) + 2.

Example (16*2) + 2 = 34.

StringBuilder class

* Java StringBuilder class is used to create mutable string.
* The Java StringBuilder class is same as StringBuffer class is same as StringBuffer class except that it is non synchronized.

methods

1. StringBuilder append()
2. StringBuilder insert()
3. StringBuilder replace()
4. StringBuilder delete()
5. StringBuilder reverse()
6. StringBuilder capacity()

The above methods example programs are similar to the StringBuffer class.

Difference between String and StringBuffer

String	StringBuffer
1. String class is immutable	1. StringBuffer is mutable
2. String is slow and consume more memory when you concat too many string	2. StringBuffer is fast and consume less memory when you concat string
3 String class override the equals() of object class	3. StringBuffer class does not override equals() of object class

Difference between StringBuffer and String Builder

StringBuffer	String Builder
1. StringBuffer is synchronized. It means two threads cannot call the methods of StringBuffer simultaneously.	1. StringBuilder is non-synchronized. It means two threads can call the methods of StringBuilder simultaneously
2. StringBuffer is less efficient than StringBuilder	2. StringBuilder is more efficient than StringBuffer.

Methods for searching string

Searching a string for finding the location of either a character or group of characters

The string class provides the following methods for performing these tasks

1. indexOf()
2. lastIndexOf().

1. indexOf() int indexOf(int ch)

This method return index position of the first occurance of character 'ch' in the string. If the specified character 'ch' is not found within the string then -1 is return

2. lastIndexOf()

* This method locates the last occurence of the character or substring in the string depending upon the argument passed.
* This method performs the search from the end of the string towards begining of the string.

If it finds the character or substring the index is returned otherwise it return -1

Methods for conversion String

1. String to int

we convert String to an int in java using Integer.parseInt() method.

```
class StringInt
{
public static void main(String[] args)
{
String s="200";
int i = integer.parseInt(s);
System.out.println(i);
}
}
```

output

200

2. int to String

we can convert int to string in java using string.valueOf()

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

```
int i=200;
String s= String.valueOf(i);
System.out.println(i+100);
 }
}
```

Output

300

String to long

we can convert string to long in java using Long.parseLong() method

```
class StringLong
{
public static void main(String[] args)
{
String s= "9989998999";
long l= Long.parseLong(s);
System.out.println(l);
}
}
```

Output

9988998999

Long to String

We convert long to string using String.valueOf() and Long.toString()

```
class LongString
{
public static void main(String[] args)
{
 long i="9989998999";
 String s=String.valueOf(i);
System.out.println(s);
 }
}
```

output

9989998999

String to float

we can convert string to float using float.parseFloat()

```
class StringFloat
{
public static void main(String[] args)
 {
  String s="23.6";
  float f=Float.parseFloat("23.6");
System.out.println(f);}}
```

output 23.6

float to String

we can convert float to String using Float.toString() or String.valueOf().

```
class FloatString.
{
public static void main(String[] args)
{
 float f = 12.3F;
 String s= String.valueOf(f);
 System.out.println(s);
}
}
```

output

12.3.

String to double

we convert String to double using Double.parseDouble()

```
class StringDouble {
public static void main(String[] args) {
  String s="23.6"
  double d= Double.parseDouble("23.6");
  System.out.println(d);
}
}
```

output

23.6

Double to String

we convert double to string in java using String.valueOf() and Double.toString().

```
class DoubleString
{
public static void main(String[] args)
{
double d= 12.3;
String s= String.valueOf(d);
System.out.println(s);
}
}
```

output

12.3.

String to Date

we convert String to Date using parse() of DateFormat and SimpleDateFormat

Date to String

we can convert Date to string using format() method of java.text.DateFormat class.

String to char

we can convert string to char using charAt()

Char to String

convert char to string using String.valueOf() or

Character.toString(char) method

String to object

we can convert string to object with assignment operator.

```
class stringobject
{
public static void main (String[] args)
{
 String s="hello";

 object obj = s;
 System.out.println (obj);
  }
}
```

output

hello

object to string

we can convert object to string using toString() or string.valueOf(object) method

Int to long

we can convert int to long using assignment operator.

long to int

we can convert long to int using typecasting.

Int to double

We can convert int to double using assignment operator

double to int

We can convert double to int using typecasting.

Char to int

we can convert char to int using various ways

Ex we direct assign char variable to int.

int to char

we can convert int to char using typecasting

String to Boolean

we can convert string to boolean using the method

Boolean.parseBoolean(String).

Boolean to String

we can convert boolean to string in Java

using String.valueOf(boolean). method

Binary to decimal

we can convert binary to decimal in java using

Integer.parseInt()

decimal to Binary

we can convert decimal to binary using

Integer.toBinaryString() method

Hexadecimal to decimal

we can convert hexadecimal to decimal using Integer.parseInt() method.

Decimal to hexadecimal

we can convert decimal to hexadecimal using

~~Integer.toHexadecimal~~

Integer.toHexString()

Octal to Decimal

we convert octal to decimal using Integer.parseInt()

Decimal to octal

we can convert decimal to octal in Java using Integer.toOctalString() method.

Multithreading

multithreading in java is a process of executing multiple threads simultaneously.

A thread is lightweight sub process, the smallest unit of processing.

Advantages

1. you can perform many operations together so it saves time
2. Threads are independent so it does not affect other threads

multitasking

multitasking is a process of executing multiple tasks simultaneously.

multitasking can be achieved in two ways.

1. process-based [multiprocessing]
2. Thread-based [multithreading]

1. process-based

* Each process has address in memory
* A process is heavyweight.

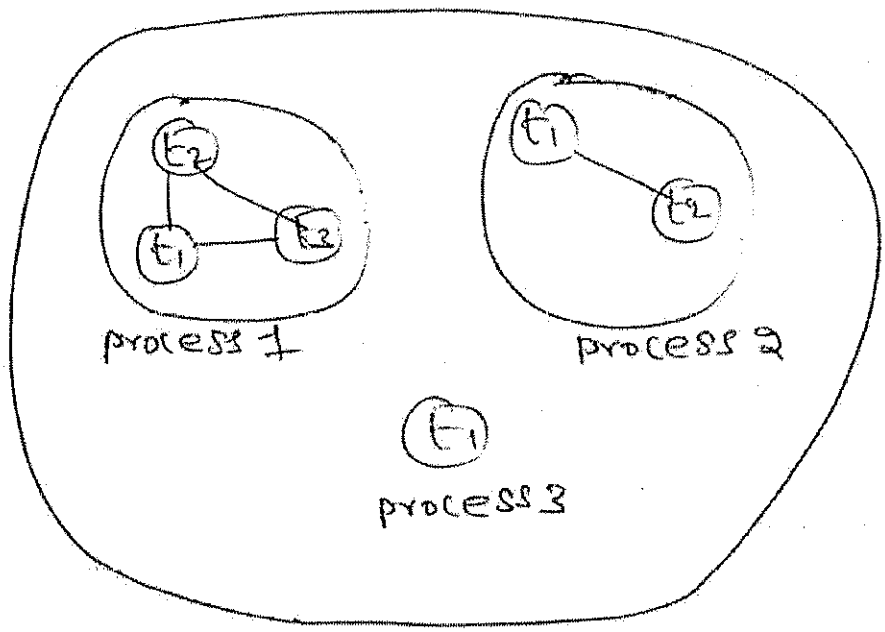
* cost of communication between the process is high

2. Thread - based

* Threads share the same address space
* A Thread is lightweight
* cost of communication between the thread is low

What is a thread

* A thread is a lightweight subprocess, the smallest unit of processing. It is separate path of execution.
* Thread are independent. If there occurs Exception in one thread it does not affect other threads.
* It uses a shared memory area



There can be multiple processes inside the OS and one process can have multiple threads.

Thread class

Thread class provides constructors and methods to create and perform operations on a thread.

Thread class Extends object class and implement Runnable interface

methods

start() – It is used to start Excution.

run() –

getstate() – It returns the state of the thread

getNAME() – It returns name of the thread

getpriority() – It returns priority of the thread

sleep() – Stop the thread for specified time

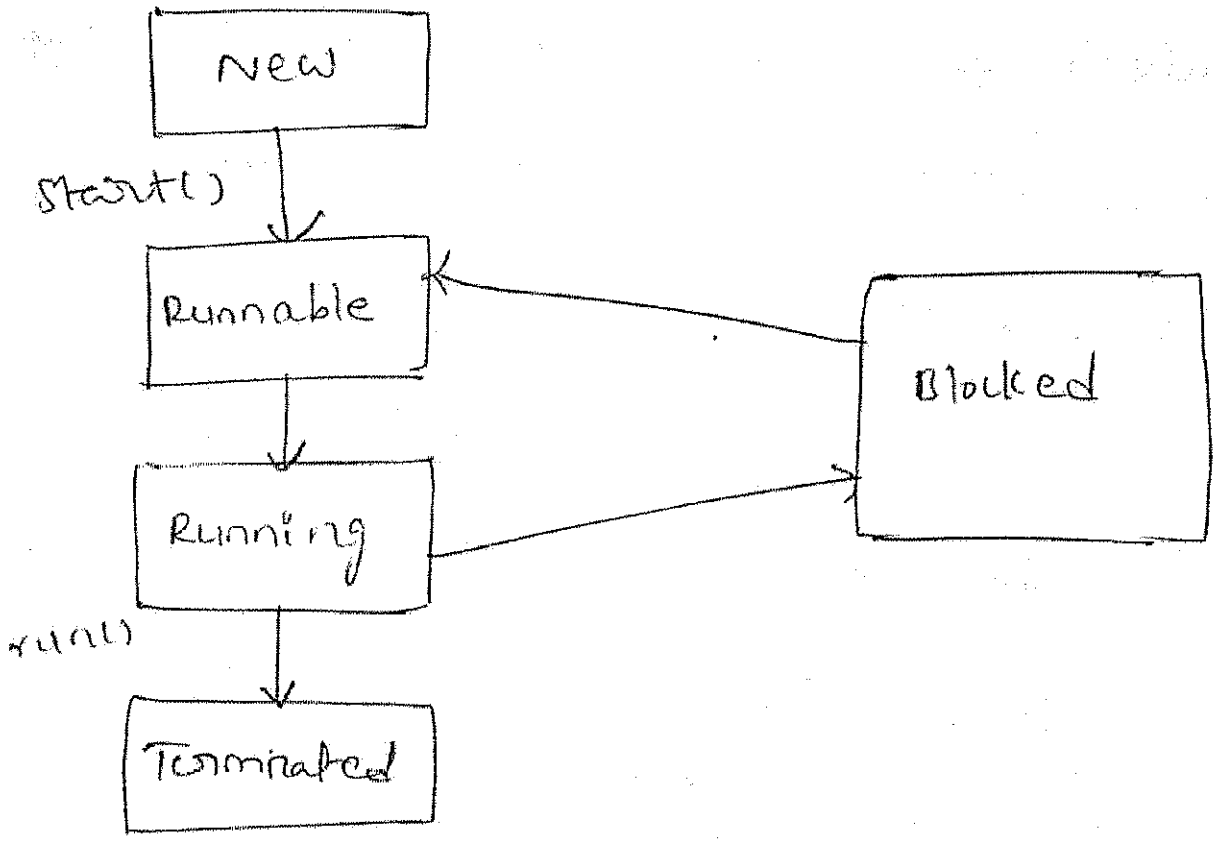
join() – Stop the current thread until the called thread gets terminated

Life cycle of thread

The life cycle of the thread in java is controlled by JVM.

The thread states are as follows.

1. New
2. Runnable
3. Running
4. Blocked
5. Terminated



1. New

The thread is new state if you create an instance of thread class but before the invocation of start() method

2. Runnable

The thread is runnable state after invocation of start() but thread scheduler has not selected it to be running thread.

3. Running

The thread is running state if the thread scheduler has selected it.

4. Blocked

This is the state when the thread is still alive but is currently not eligible to run.

5. Terminated

A thread is in terminated or dead state when its run() method exists.

Need for multiple thread

* The main purpose of multithreading is to provide simultaneous excution of two or more parts of a program to maximum utilize the cpu time.
* A multithreaded program contains two or more parts that can run concurrently.
* Each such part of a program called thread.

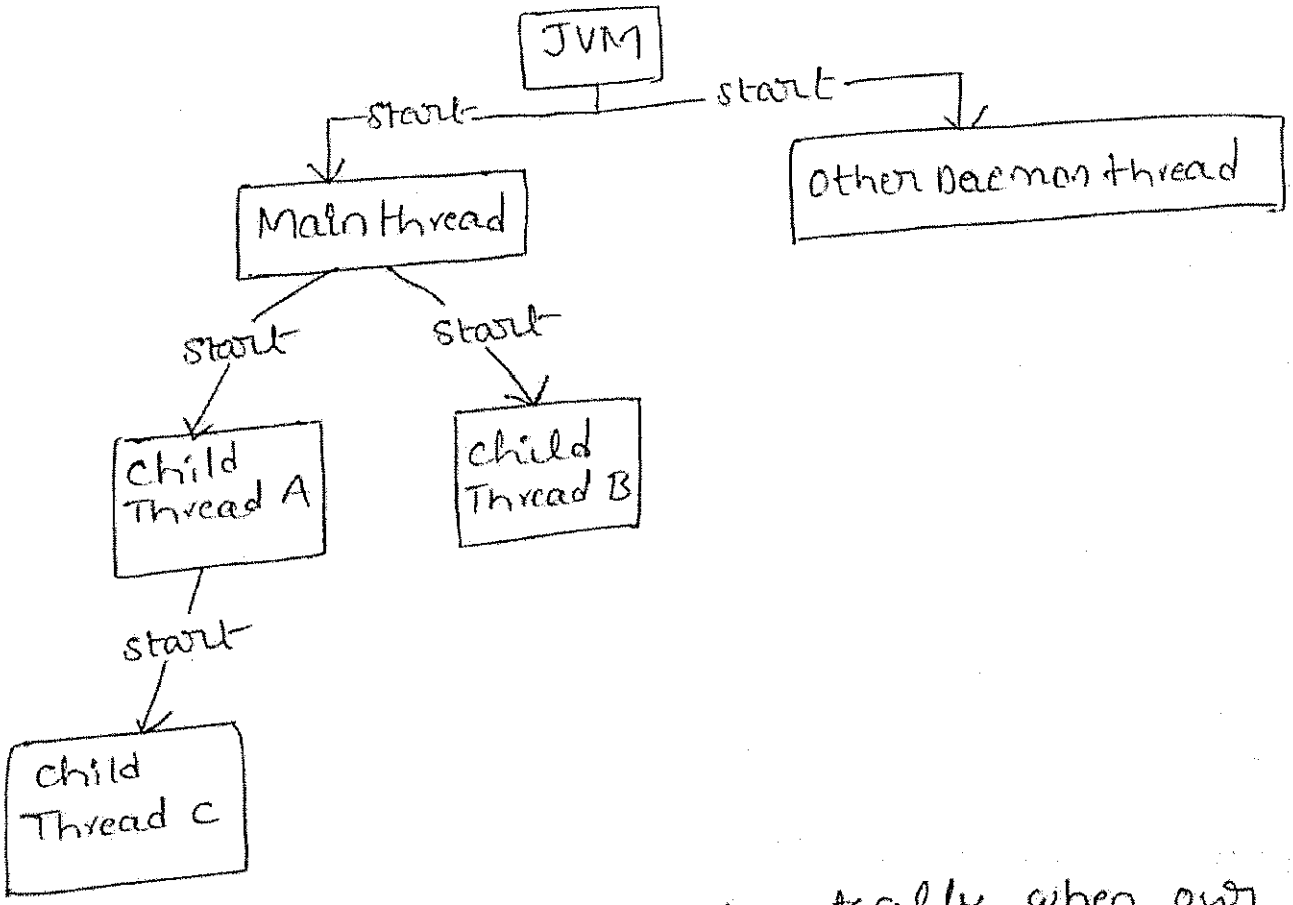
Main Thread

When a java program starts up one thread begins running immediatly. This is usually called the main thread of our program.

properties

* It is the thread from which other child thread will be spawned.
* often it must be the last thread to finish execution because it performs various shutdown actions

Flow diagram main thread



The main thread is created automatically when our program is started.

For each program a main thread is created by JVM [Java virtual machine]. The main thread first varifies the existence of the main() method.

program

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

```
System.out.println (" Entering to deadlock");

    Thread.currentThread().join();

    System.out.println ("This statement will never execute");

    }
  catch ( InterruptedException e)

  {
    e.printStackTrace();

    }

  }
}
```

output

Entering into deadlock

Create new thread

There are two ways to create a thread.

1. By Extending Thread class
2. By Implementing Runnable interface

Commonly used constructor thread class

* Thread()
* Thread(String name)
* Thread(Runnable r)
* Thread(Runnable r, String name)

Thread class

Thread class provide constructors and methods to create to create and perform operations on thread. Thread class Extends object class and implements Runnable Interface

program Extending thread class

```
class Multi Extends Thread
{
 public void run()
  {
  System.out.println("thread is running");
  }
 public static void main (String[] args)
  {
   Multi t1=new Multi();
```

```
    t1.start();
  }
}
```

output

thread is running.

Runnable interface

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread.

Runnable interface have only one method i.e

1. public void run() is used to perform action for a thread.

program for Runnable interface

```
class multi3 implements Runnable
{
public void run()
{
System.out.println(" thread is running");
}
public static void main (String[] args)
{
multi3 m1 = new Multi3();
```

```
Thread t1 = new Thread(m1);
 t1.start();
 }
}
```

<u>output</u>

thread is running

<u>Start()</u> Thread class is used to start a newly created thread

* A new thread starts
* The thread moves from new state to the Runnable state

*<u>Thread priorities Synchronization</u>

* Each thread have a priority. priorities are represented by a number between 1 and 10.
* In most cases the priority is given by the JVM or by the programmer itself
* The processor assigns a priority to thread scheduler

1. <u>public static int MIN_PRIORITY</u>

It holds the minimum priority that can be given to a thread. The value of this is 1.

2. public static int NORM_PRIORITY

It is the default priority that is given to a thread if it is not defined. The value for this is '0'.

3. public static int MAX_PRIORITY

It is the maximum priority that can be given to a thread. The value for this is 10.

Get and Set methods in priority

1. public final integer getPriority()

getPriority() is in java.lang.Thread package.

It is used to get the priority of a thread.

2. public final void setPriority(int newPriority)

setPriority(int newPriority) is in java.lang package.

It is used to set the priority of a thread.

program

```
class multiPriority Extends Thread
{
public void run()
 {
System.out.println("running thread name is:" + Thread.current-
    Thread().getName());
```

```
System.out.println(" running thread priority is:" +Thread.
    currentThread().getPriority());
  }
public static void main (String[] args)
{
MultiPriority m1=new MultiPriority();
MultiPriority m2=new MultiPriority();
m1.setPriority(Thread.MIN_PRIORITY);
m2.setPriority (Thread.MAX_PRIORITY);
m1.start();
m2.start();
}
}
```

<u>output</u>

```
running thread name is: Thread-0
running thread priority is: 10
running thread name is: Thread-1
running thread priority is: 1.
```

Inter-thread communication

Inter thread communication is a mechanism in which a thread is paused running in its critical section and other thread is allowed to enter in the same critical section to be executed.

It is implemented by following object class.

1. wait()
2. notify()
3. notifyAll()

1. wait()

It tells calling thread to give up monitor and go to sleep until some other thread enters the same monitor and call notify.

2. notify()

wakes up a thread that called wait() on same object

3. notifyAll()

wakes up all the thread that called wait() on same object.

Polling

The process of testing a condition repeatedly till it becomes true is known as polling so this waste CPU time.

To avoid polling Java uses three methods namely wait(), notify(), notifyAll().

Difference between wait() and sleep()

wait()	sleep()
1. wait() release lock	1. Sleep() doesnt release lock.
2. It is the method of object class	2. It is the method of thread class
3. It is non static method	3. It is static method
4. It should be notified by notify() or notifyAll() methods	4. After the specified amount of time, sleep is completed.

program

```
class Customer
{
int amount=10,000;
synchronized void withdraw(int amount)
{
```

```
System.out.println(" going to withdraw");
 if (this.amount < amount)
 {
System.out.println(" less balence; waiting for deposit");
  try
  {
 wait();
  }
 catch (Exception e)
 {

   }
}
this.amount -= amount;
System.out.println(" withdraw completed");
 }
Synchronized void deposit(int amount)
 {
System.out.println(" going to deposit");
 this.amount += amount;
System.out.println(" deposit completed");

 notify();

}
}
class Test
 {
```

```
public static void main (String[] args)
{
final Customer c = new Customer();
 new Thread()
 {
public void run()
{
 c.withdraw (15000);
}
}
 start();
 new Thread()
 {
 public void run()
 {
  c.deposit (10,000);
  }.start();
  }
}
```

<u>output</u>

```
going to withdraw
less balance; waiting for deposit
  going to deposit
  deposit completed
  withdraw completed.
```

Suspending thread

* The suspend() of thread class puts the thread from running to waiting state.
* This method is used if you want to stop the thread execution and start it again when a certain event occurs
* The suspended thread can be resumed using the resume()

Syntax

```
public final void suspend()
```

program

```
class Suspend Extends Thread
{
 public void run()
 {
  for (int i=1, i<5; i++)
   {
    try
    {
    sleep(500);
System.out.println(Thread.currentThread().getName());
  }
catch (InterruptedException e)
  {
  System.out.println(e);
  }
```

```
System.out.println(i);
}
}
public static void main(String[] args)
{
Suspend t1 = new Suspend();
Suspend t2 = new suspend();
Suspend t3 = new suspend();
t1.start();
t2.start();
t2.suspend();
t3.start();
}
}
```

<u>output</u>

```
Thread -0
 1
Thread -2
 1
Thread -0
 2
Thread -2
 2
Thread -0
 3
Thread -2
 3
Thread -0
 4
Thread -2
 4.
```

Resume()

The resume() of thread class is only used with suspend() method.

This method is used to resume a thread which was suspended using suspend()

Syntax

```
public final void resume()
```

program

```
class Resume Extends Thread
{
 public void run()
 {
  for (int i=1; i<5; i++)
   {
    try
     {
      sleep(500);
System.out.println(Thread.currentThread().getName());
     }
    catch (InterruptedException e)
    {
    System.out.println(e);
    }
    System.out.println(i);
   }
 }
 public static void main (String[] args)
 {
```

```
Resume t1 = new Resume();
Resume t2 = new Resume();
Resume t3 = new Resume();
  t1.start();
  t2.start();
  t2.suspend();
  t3.start();
  t2.resume();
 }
}
```

<u>Output</u>

```
Thread - 0
  1
Thread - 2
  1
Thread - 1
  1
Thread - 0
  2
Thread - 2
  2
Thread - 1
  2
Thread - 0
  3
Thread - 2
  3
Thread - 1
  3
Thread - 0
  4
Thread - 2
  4
Thread - 1
  4.
```

Stopping thread

The stop() method of thread class terminates the thread execution. once a thread is stopped it can not be restarted by start()

Syntax

```
public final void stop()
public final void stop(Throwable obj)
```

program

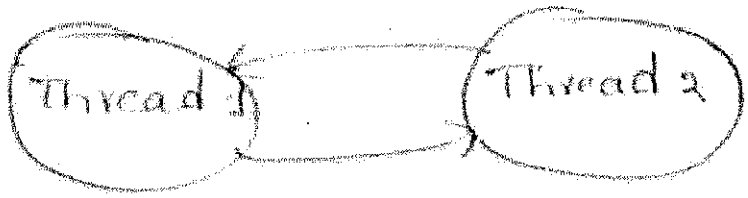
```
class stop Extends Thread
{
public void run()
 {
 for (int i=1; i<5; i++)
   {
    try
     {
      Sleep(500);
System.out.println(Thread.currentThread.getName());
 }
catch (InterruptedException e)
{
System.out.println(e);
}
System.out.println(i);
}
```

```
}
public static void main (String[] args)
{
 Stop t1 = new Stop();
 Stop t2 = new Stop();
 Stop t3 = new Stop();
 t1.start();
 t2.start();
 t3.stop();
 System.out.println("Thread t3 is stopped");
}
}
```

Dead lock in java

Deadlock in java is a part of multithreading. Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock

Both threads are waiting for each other to release the lock the condition is called dead lock



Race conditions

Race condition in Java is a type of concurrency bug or issue which is introduced in your program because parallel execution of your program by multiple threads at the same time

Race condition occur when two thread operate on the same object without proper synchonization and their operation interleave on each other

Race condition in java occurs in a multithreading environment when more than one thread try to access a shared resource [modify, write] at the same time

Java Data Base Connectivity

* Java data base connectivity is a Java API to connect and execute the query with the database
* It is the part of JavaSE [Standard Edition].
* JDBC API uses JDBC drivers to connect with the database. [API - Application programming interface]
* We can use JDBC API to access tabular data stored in any relational database
* By the help of JDBC API we can save, update, delete and fetch data from the database
* We can use JDBC API to handle database using Java program and perform the following activities

(i) connect to the database

(ii) Execute queries and update statements to the database

(iii) Retrieve the result received from the database

Architecture of JDBC

1. Application

Application in JDBC is a Java applet that communicates with a data source

2. JDBC API

JDBC API provides classes, methods and interfaces that allow Java programs to execute SQL statements and retrieve results from the database

3. Driver manager The driver manager plays an important role in the JDBC architecture. The driver manager uses some database specific drivers that effectively connect enterprise applications to database

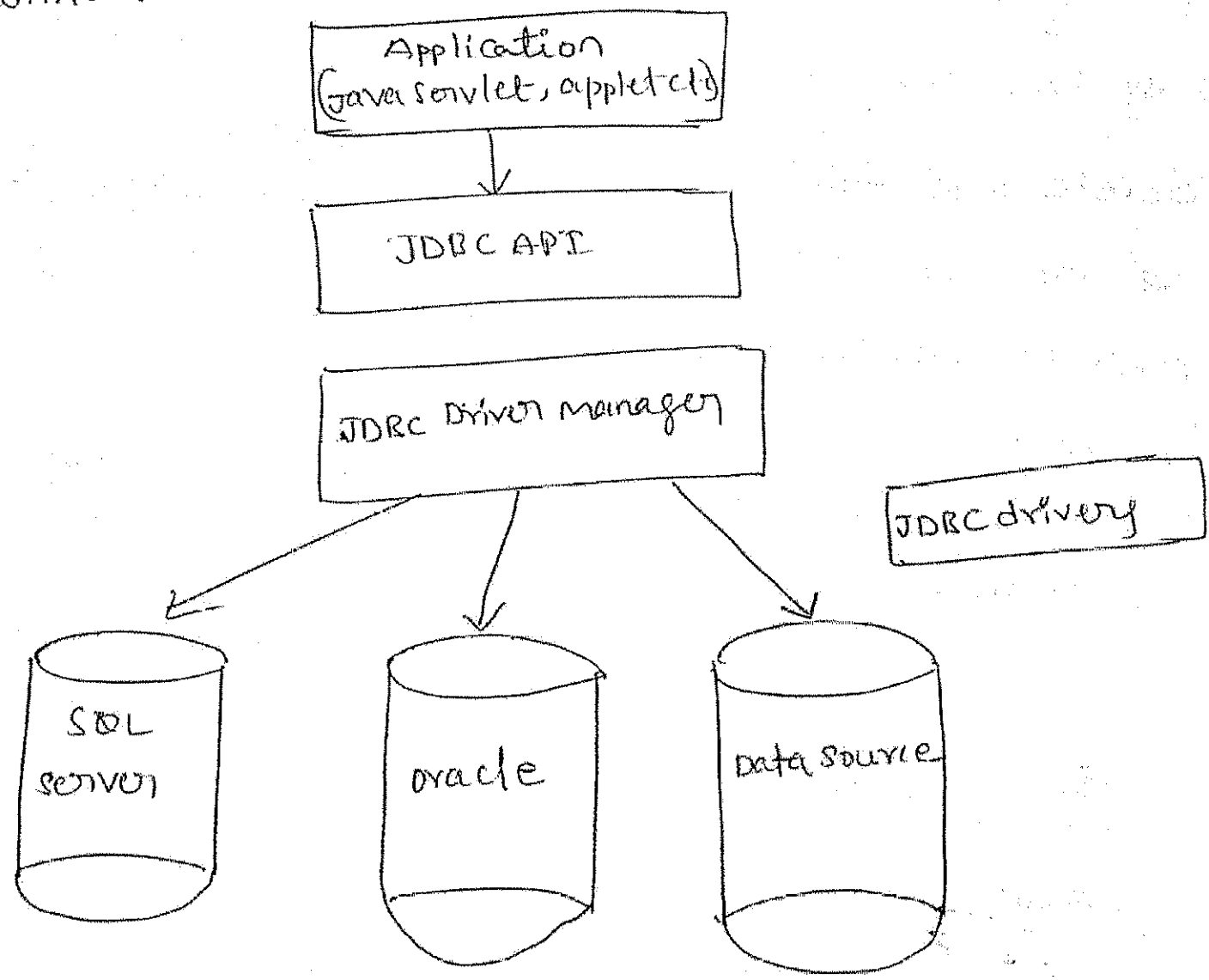


figure JDBC ARCHITECTURE

4. JDBC drivers (bridge)

The JDBC-ODBC bridge driver connects the database drivers to the database.
This bridge driver translates the JDBC method call to the ODBC method call [OR]

JDBC drivers helps us to communicate with a data source through JDBC.

There are two types of JDBC architecture

1. Two tier model
2. Three tier model.

1. Two-tier model

* In two tier model java application talks directly to the data source
* JDBC driver provides communication between the application and data source
* when a user sends a query to the data source the answers to those queries are given to the user in the form of results
* we can locate the data source on a different machine on a network to which a user is connected.

Q. Three-tier model

In the three-tier model the query of the user queries goes to the middle tier service.

From the middle tier service the command again reach the data source the results of the query go back to the middle tier.

It finally goes to the user

This type of model is beneficial for management information system.

Creating JDBC Application

The following steps are required to create a new Database using JDBC application.

1. Import the package

Requires that you include the packages containing the JDBC classes needed for database programming using import java.sql.*

2. open a connection

Requires using the DriverManager.getConnection() to create a connection object which represents a physical connection with the database server

To create a new database you need not give any database name while preparing database URL

3. Execute a query

Requires using an object of type statement for building and submitting an SQL statement to the database

4. Clean up the environment

try with resources automatically close the resources

program

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
public class JDBCExample
{
static final String DB_URL="jdbc:mysql://localhost/";
static final String USER="guest";
static final String PASS="guest123";
public static void main (String[] args)
{
try (Connection conn=DriverManager.getConnection(DB_URL,
                                USER,PASS));

  Statement stmt= Conn.CreateStatement();
  {
  String sql="CREATE DATABASE STUDENTS";
   stmt.executeUpdate(sql);
  System.out.println("Database created successfully...");
  }
```

```
Catch (SQLException e)
{
  e.printStackTrace();
  }
 }
}
```

C:\> JAVAC JDBCExample.Java

C:\> JAVA JDBCExample

Database Created successfully.--

Establishing JDBC connections

Before establishing a connection between front end i.e your Java program and back end ie database

Steps for connectivity between Java program and database

1. Loading the Driver

To begin with you first need load the driver or register it before using it in the program.

you can register a driver in one of two ways

(1) Class.forName()

Here we load the drivers class file into memory at the runtime. No need of using new or creation of object

Ex Class.forName("oracle.jdbc.driver.OracleDriver");

(ii) DriverManager.registerDriver();

DriverManager is a java inbuilt class with a static member register.

Ex

DriverManager.registerDriver(new Oracle.jdbc.driver.OracleDriver())

2. Create the connection

After loading the driver, establish connections using

Connection con = DriverManager.getConnection(url, user, password)

user - username from which your sql command prompt can be accessed

password password from which your sql command prompt can be accessed.

con is a reference to connection interface

url uniform Resource Locator.

3. Create a statement

once a connection is established you can interact with the database. The JDBCStatement, callableStatements, and preparedStatement interface define the methods that enable you to send SQL commands and receive data from your database

Use of JDBC Statement is as follows.

Statement st = con.createStatement();

Here con is a reference to connection interface used in previous step.

4. Execute the query

We can have multiple types of queries

* Query for updating / inserting table in a database
* Query for retrieving data

The executeQuery() of statement interface is used to execute queries of retrieving values from the database

5. close the connections

So finally we have sent the data to the specified location and now we are at the verge of completion of our task.

By closing connection, objects of statement and Resultset will be closed automatically.

The close() method of connection interface is used to close the connection

Example

```
con.close();
```

program

```
import java.sql.*;
import java.util.*;
class main
{
public static void main(String[] args)
{
String url = "jdbc:oracle:thin:@localhost:1521:xe";

String user= " System");
String pass = " 12345";
Scanner k= new Scanner(System.in);
System.out.println(" enter name");
String name = k.next();
System.out.println("enter roll no");

int roll = k.nextInt();
System.out.println(" enter class");

String cls = k.next();
String sql = " insert into Student1 values ("+name+" ,"+roll+",
                    " + cls");

    Connection  con = null;
    try
    {
     DriverManager.registerDriver(new oracle.jdbc.OracleDri-
                                  ver());
```

```
con = DriverManager.getConnection(url, user, pass);

Statement st = con.CreateStatement();
int m = st.executeUpdate(Sql);
if (m==1)
System.out.println("inserted successfully: "+Sql);
else
System.out.println("insertion failed");
  con.close();
 }
 catch (Exception ex)
 {
System.out.println(ex);
 }
}
```

Output

```
javac Main.java
java Main
enter name
abc
enter roll no
12
enter class
6a
inserted successfully: insert into student values ('abc', 12, '6a')
```

Batch processing in JDBC

Instead of Executing a single query, we can Execute a batch (group) of queries. It makes the performance fast.

The java.sql.statement and java.sql.prepared-statement interface provides methods for batch processing

Advantage

* Fast performance

Methods

* void addBatch(String query) – It adds query into batch.
* int[] ExecuteBatch() – It Executes the batch of queries

steps for batch processing in JDBC

* Load the driver class
* Create connections
* Create Statement
* Add query in the batch
* Execute Batch
* Close connections

Batch processing aims at improving performance and data consistency.

we can perform batch processing using the addbatch() and the executeBatch() methods which are avelable in both statement and preparedStatement classes

addBatch()

addBatch() method is used to construct a batch.

constructing a batch means storing the SQL commands in a buffer, maintained by statement object

executeBatch()

when executeBatch() called then the commands will be transferred at a time as a batch from buffer to database

Transaction management in JDBC

Transaction management represents a single unit of work.

The ACID properties describes the transaction management well.

ACID stands for.

Atomicity means either all successful or none

consistency It ensure bringing the database from one consistent state to another consistant state

Isolation

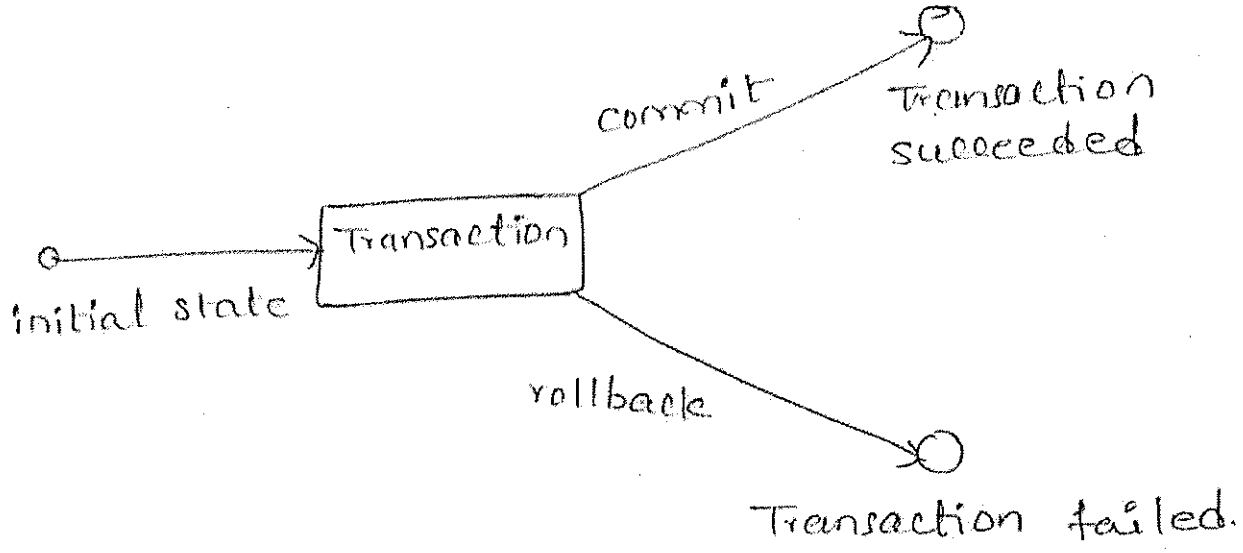
It ensures that transaction is isolated from other transaction

Durability

It means once a transaction has been committed it will remain so, even in the event of errors, power loss etc.

Advantage

It makes fast performance because database is hit at the time of commit.



Methods

* void setAutoCommit(boolean status) - It is true by default means each transaction is committed by default
* void commit()

commits the transaction

void rollback()

cancels the transaction

important functions in transaction management

1. commit

After the execution of SQL statements we want to make the changes permanant in the database

Rollback

Rollback undoes the changes till the last commit or mentioned savepoint.

Savepoint

Savepoint helps to create checkpoint in a transaction and it allows to perform a rollback to that particular savepoint.

Any savepoint that has been created for a transaction will be automatically destroyed and become invalid once the transaction is committed or rolled back.

Result set interface

The object of ResultSet maintains a cursor pointing to a row of a table. initially cursor points to before the first row but we can make this object to move forward and backward direction by passing either TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE in Create Statement (int int) method.

Methods in ResultSet

1. public boolean next()

It is used to move the cursor to the one row next from the current position.

2. public boolean previous()

It is used to move the cursor to the one row previous from the current position.

3. public boolean first()

It used to move the cursor to the first row in resultset object

4. public boolean last()

It is used to move the cursor to the last row in resultset object

5. public boolean absolute(int row)

It is used to move the cursor to the specified row number in the resulltset object

6. public boolean relative(int row)

It is used to move the cursor to the relative row number in the resultset object, it may be positive or negative

7. public int getInt(int columnIndex);

It is used to return the data of specified

column index of the current row as int.

8. public int getInt (String columnname):

It is used to return the data specified column name of current row as row as int

9. public String getString (int ColumnIndex)

It is used to return the data specified column index of the current row as String

10 public String getString (String ColumnName)

It is used return the data specified column name of the current row as String.