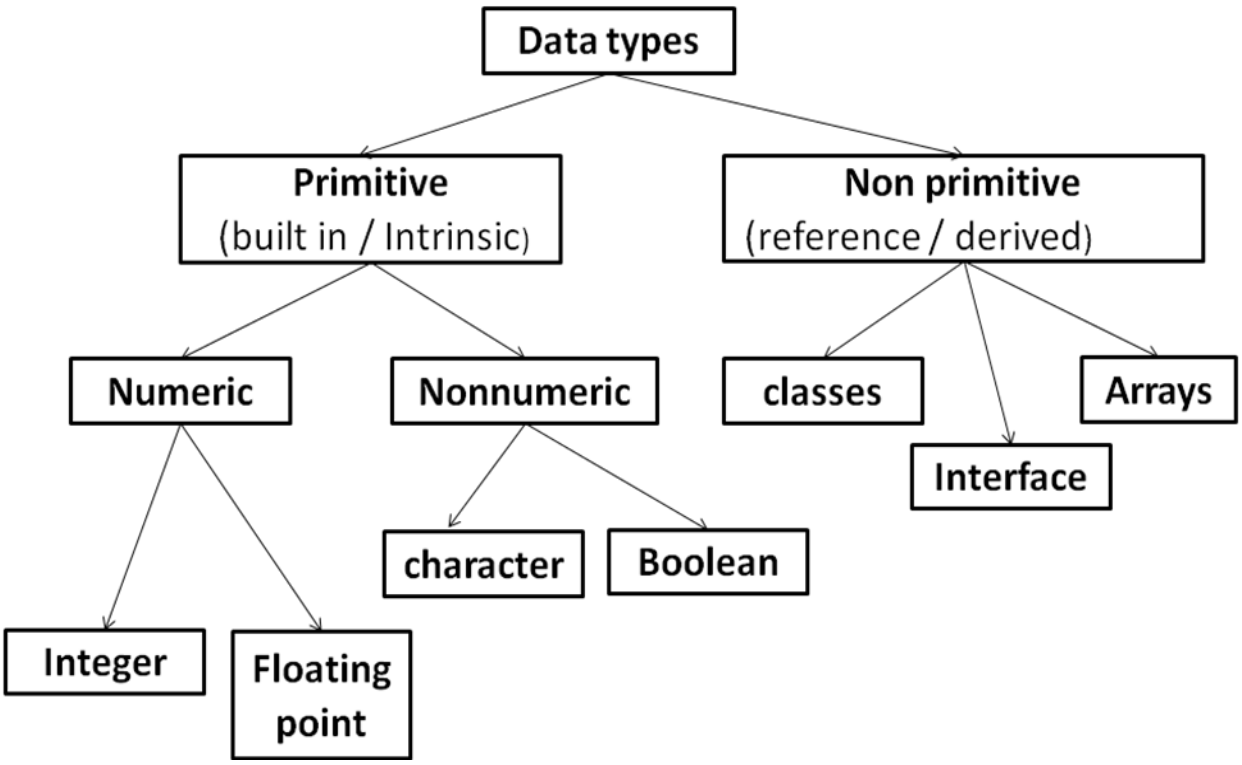
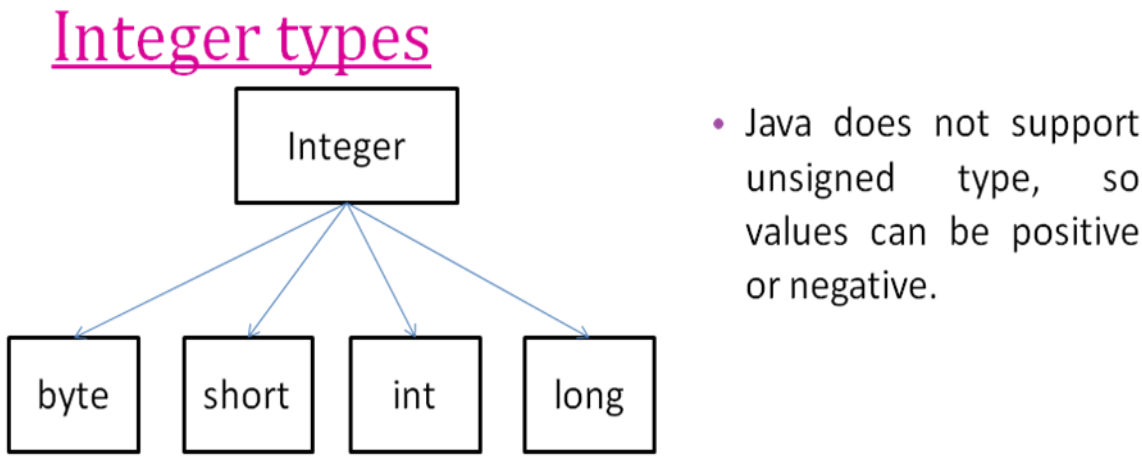
Unit-2 building block of language

2.1 Primitive Data Types: Integers, Floating Point type, Characters, Booleans etc
2.2 User Defined Data Type
2.3 Identifiers & Literals
2.4 Declarations of constants & variables
2.5 Type Conversion and Casting
2.6 Scope of variables & default values of variables declared
2.7 Wrapper classes
2.8 Comment Syntax
2.9 Garbage Collection
2.10 Arrays of Primitive Data Types
2.11 Types of Arrays
2.12 Creation, concatenation and conversion of a string, changing case of string, character extraction, String Comparison, String Buffer
2.13DifferentOperators:Arithmetic,bitwise, Relational,Logical,Assignment,Conditional,ternary,Increment/Decrement, Mathematical Functions
2.14 Decision and control statements: Selection Statement (if, if...else, switch),Loops while, do-while, for), Jump statements (break, continue, return & exit)

2.1 Data types in Java

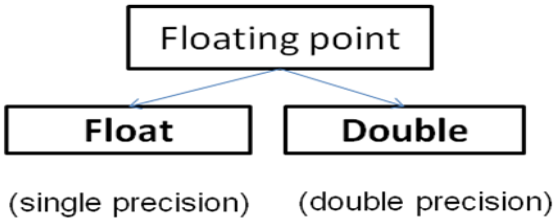




- Java does not support unsigned type, so values can be positive or negative.

Type	Size	Minimum value	Maximum value
byte	1 byte	-128	127
short	2 byte	-32678	32767
int	4 byte	-2,147,483,648	2,147,483,647
long	8 byte	-9,223,372,036,854,754,808	9,223,372,036,854,754,808

Floating types



Type	Size	Minimum value	Maximum value
Float	4 byte	3.4 e -038	3.4 e +038
double	8 byte	1.7 e -308	1.7 e +308

- Floating point no. are treated as **double precision** , so to force them to be in single precision mode ,we must append **f or F**.
  Ex: 1.23f , 7.6512F
- **NaN: Not a Number (**special value supported by floating point data type**)**
  It is used to represent result of operation such as dividing by zero ,where actual number is not produced.

Character type

- **char** is used ,Size : 2 byte
- It holds single character only.

Boolean type

- It is used to test a particular condition during execution of program.
- It has two values : **true** or **false**
- **boolean** keyword is used
- Size: 1 bit (0 or 1).All comparison operators return boolean type variable.

So, there are **8 basic(primitive)** data type in java.

Sr. No.	Type	Size (in bytes)
1	boolean	1 **bit**
2	byte	1 byte
3	character	2 bytes
4	short	2 bytes
5	int	4 bytes
6	float	4 bytes
7	long	8 bytes
8	double	8 bytes

2.2 User Defined Data Type

- **Class, interface and array are user defined data type.**
- Class encapsulates various variables and methods
- Interface is same as class except it contains methods without body. It is used to implement multiple inheritance.
- An array contains same type of elements and provides sequential access.

2.3 Identifiers & Literals (Java tokens)

- Java program is a collection of classes.
- Class is defined by a set of declaration statements and methods containing executable statements.
- Smallest individual unit in program is tokens.
- Compiler recognize them for building up expressions and statement

Java program=tokens + comments + white space

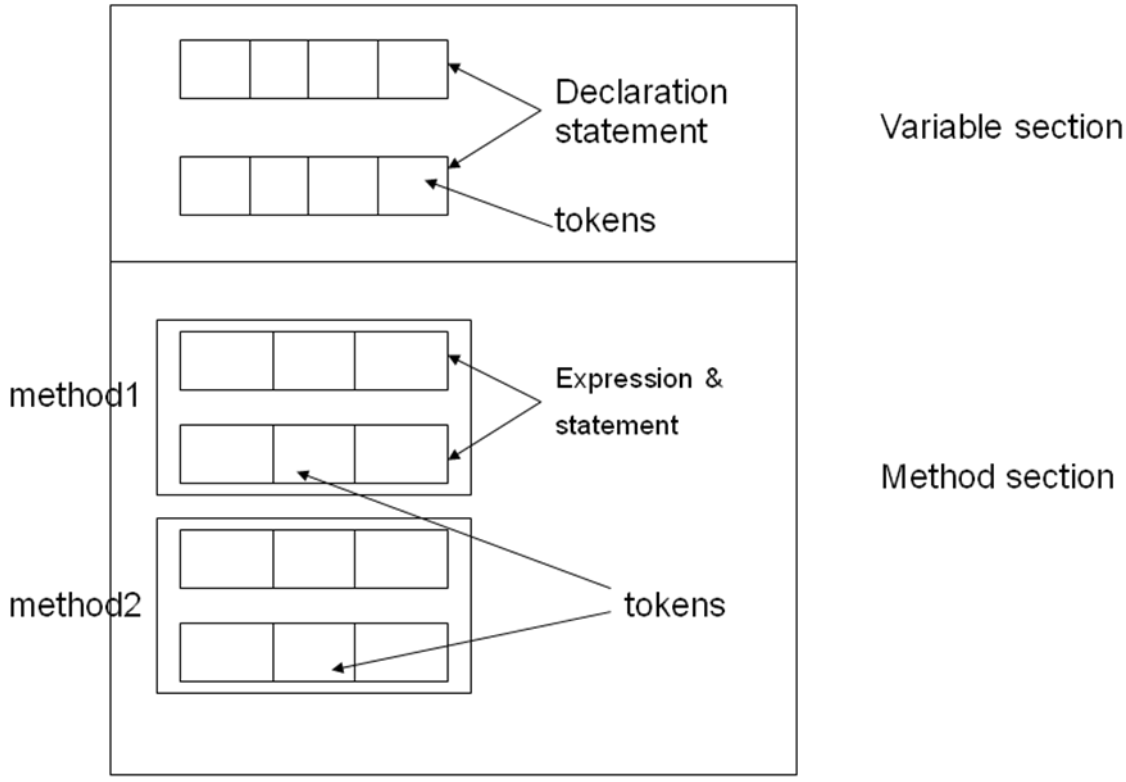


Fig. Elements of java class

There are Five types of tokens :

1) reserved keywords
2) identifiers
3) literals
4) operators
5) separators

Java character set

- Smallest unit of java language are characters.
- Characters are used to write java tokens, this characters are defined by Unicode character set.
- Unicode is 16-bit character coding system.

1) Keywords

- Java has 60 reserved keywords.
- We can not use keywords as names for variables, classes, methods and so on.
- Keywords should be written in small letters.

abstract	Boolean	Break	byte	byvalue*	case	cast*
catch	char	Class	const*	continue	default	do
double	else	Extends	false**	final	finally	float
for	future**	generic*	goto*	if	implements	import
inner*	instanceof*	Int	interface	long	native	new
null**	operator*	outer*	package	private	protected	public
rest*	return	Short	static	super	switch	syncronized
this	threadsafe*	Throw	throws	transient	true**	Try
var*	void	Volatile	while			

* Reserved for future use

** These are values defined by java

2) Identifiers

- Identifiers are programmer designed tokens.
- It is used for naming classes ,methods , variables , objects , label , package and interface in program.

Example:

- average , sum ,dayTemprature , totalMarks

3) Literals

- Sequence of characters (digits , letters ,and other character) that represent constant values to be stored in variables.
- Five types:
  - Integer literals
  - Floating_point literals
  - Character literals
  - String literals
  - Boolean literals

4) Operators

- All operator is a symbol that takes one or more arguments and operates on them to produce a result.

5) Seperators

- They are symbols used to indicate where groups of code are divided and arranged.

Sr. No.	Name
1	Parentheses **( )**
2	Braces **{ }**
3	Brackets **[ ]**
4	Semicolon **;**
5	Comma **,**
6	Period **.**

2.4 Declarations of constants & variables

Constant

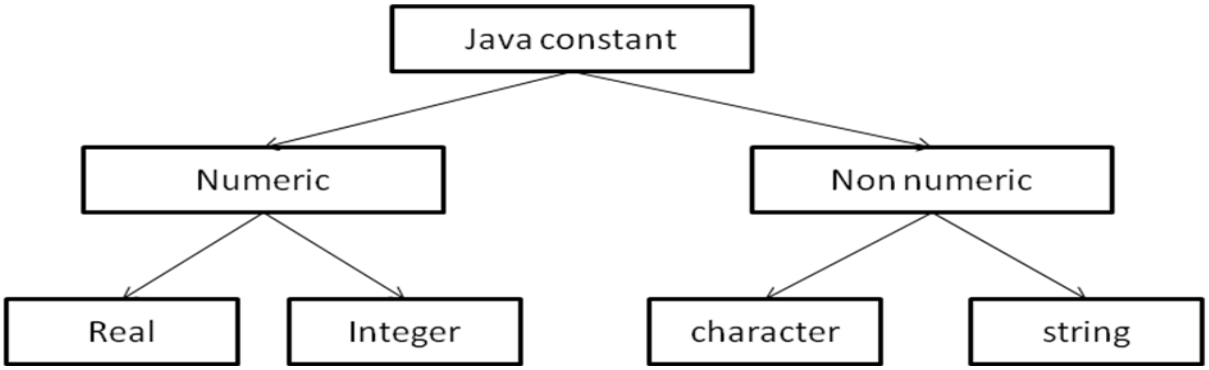
- Constant in java refers to fixed values do not change during execution of program.

1) Integer Constant
2) Real constant
3) Single constant
4) String constant

1) Integer constant:

Integers:

a. Decimal : 0 to 9 [ex. 123, 012 , 16789]
b. Octal : [ex. 037 , o ,0435]
c. Hexadecimal : [0x2 , 0x9F , 0x ,oxbcd]



2) Real constant

- Floating point / Real point constants

Ex. 0.083 , -0.75 , 215. , .95 , -.71

syntax : mantissa e Exponent

- Ex.
  - .65e4 , 12e-4 , 1.5 e+5 , 3.18E3
- Floating point contain 4 parts:
  - A whole number
  - A decimal number
  - A fractional part
  - An exponent

3) Single character constant

Ex.

‘5’ , ‘Y’ , ‘;‘ , ‘ ‘

4) String constant

Sequence of character enclosed between **double quotes.**

Ex.

“hello java” , “1997” ,”?----” , “5+3” , “x”

Back slash character sequence (Escape sequence)

‘\b’	Back space
‘\f’	Form feed
‘\n’	New line
‘\r’	Carriage return
‘\t’	Horizontal tab
‘\”	Single quote
‘\\’	Double quote

Variable

- It is a data name used to store a data value.
- Variable may take different values at different times during execution of program.

- Rules for declaration
  1. Not start with digit
  2. Upper case and lower case letters are distinct
  3. Not should be a keyword
  4. No white spaces
  5. Variable name can be of any length.

Declaration of variable

Syntax:

type variable1, variable2,……;

Ex:
```
int count;
float  x , y ;
double p1;
byte b;
char  ch;
```

Giving values to varaible

1) By using assignment statement

Syntax : **variable name=value**;

Ex: i=0,x=0.0f;

int i=0; char ch='X';

2) By using read statement: (**readLine()** method)

readLine() method reads input from keyboard as a string which is then converted to corresponding data type using wrapper classes.

3) By passing parameter in command line as its argument

2.5 Type Conversion and Casting

- **Process of converting one data type to another is called type casting**.
- When we need casting?

It is used when there is need to store a value of one type into a variable of another type.

- **Syntax:**

type variable1 = (type) variable2 ;

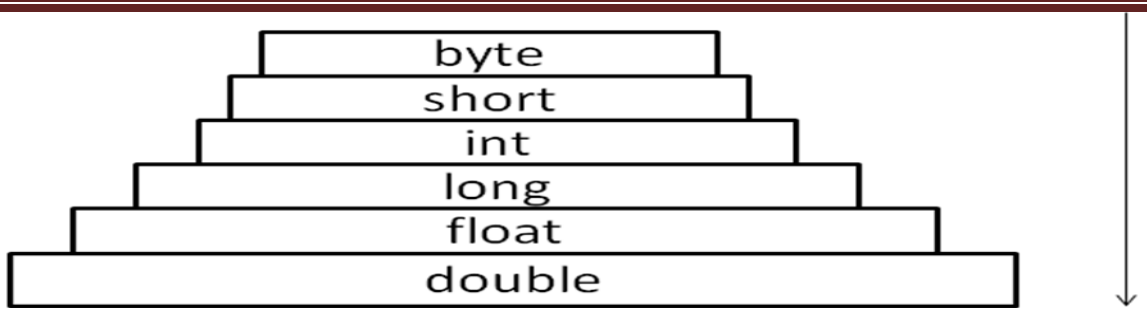
Example:

```
int m=5;

byte n=(byte) m;    // converts int m into byte type

long count=(long) m; // converts int m into long type
```

- float and double type can be converted any except boolean.
- byte ,short , int and long can be converted in any type except boolean .
- Casting in smaller type may result in loss of data.
- Casting floating point into integer will loss of fractional part.



Casts that result in No Loss of information

From	To
byte	short , char , int , long , float , double
short	int , long , float , double
char	int , long , float , double
int	long , float , double
long	float , double
float	double

Automatic type conversion

- It is possible to assign value of one type to a other type without casting, is known as automatic type conversion.
- It is possible **only if destination type has enough precision to store source type.**
- Example:

```
byte b=75;
int a=b;
```

Two operations :

1) Widening(promotion)
2) narrowing

- **Widening:** process of assigning smaller type to larger type is widening or promotion.

```
Ex:   byte b=5;
      int a=b;
```

- **Narrowing:** process of assigning larger type to smaller type is known as narrowing.

```
Ex:   float c=25.43f;
      int a=(int) c;
```

- Narrowing may result in loss of information.

2.6 Scope of variables & default values of variables declared

Variable classified as

1. instance variable
2. Class variable
3. Local variable

1) Instance variable:

- Declared inside class
- They are creating when object are instantiated and therefore they are associated with objects.
- It takes different values for different objects.

2) Class variable

- Declared inside class
- They are global to class and belongs to the entire set of object that class creates.
- Only one memory location is created for each class variable

3) Local variable

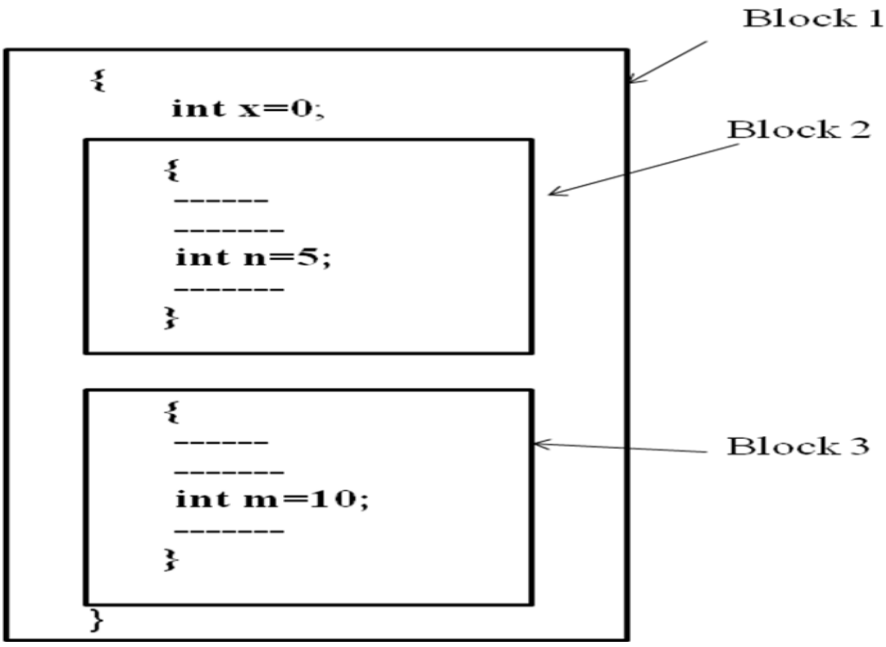
- Variable declared and used inside methods.
- They work only inside that specific method.

```
class A
{
        int a;

        sum(int x, int y)
        {
        int z=x + y;
        }
}
```

Global or class variable

Here, x ,y and z are local variable



Standard default values

Type	value
byte	**Zero:(byte) 0**
short	Zero**:(short)** 0
int	Zero : 0
long	Zero: 0L or 0l
double	0.0d
float	0.0f
char	Null character or '\u0000'
boolean	false
Reference	null

2.7 Wrapper classes

- Vector class cannot handle primitive data types like int, float, long, char and double.

Use:

- Primitive data types may be converted into objects types by using wrapper classes contained in **java.lang;**
- **Wrapper classes for converting simple types**

No.	Primitive data type	Wrapper class
1	boolean	Boolean
2	char	Character
3	double	Double
4	float	Float
5	int	Integer
6	long	Long

- Wrapper classes have number of unique methods for handling primitive data type.

Converting **primitive numbers to object number** Using constructor methods :

1	**Integer** intval= new **Integer**(i);	Primitive integer to Integer object
2	**Float** floatval=new **Float**(f);	Primitive float to Float object
3	**Double** doubleval=new **Double** (d);	Primitive double to Double object
4	**Long** longVal=new **Long**(l);	Primitive long to Long object

Converting **object numbers to primitive number** Using **typeValue ( ):**

1	int i = intval.**intValue( );**	Object to Primitive integer number
2	float f = floatval.**floatValue( );**	Object to Primitive float number
3	double d= doubleval.**doubleValue( );**	Object to Primitive double number
4	Longl = longVal.**longValue ( );**	Object to Primitive long number

Converting numbers to String using **String ( ) methods:**

1	str= **Integer**.toString (i);	Primitive int to String
2	str= **Float**.toString (f);	Primitive float to String
3	str= **Double**.toString (d);	Primitive double to String
4	str= **Long.**toString (l);	Primitive long to String

Converting String objects to numeric objects using static ( ) methods **valueOf( ):**

1	Intval=Integer.**valueOf(** str **);**	Convert string to int object
2	Floatval=Float.**valueOf** (str);	Convert string to float object
3	Doubleval=Double.**valueOf(** str);	Convert string to double object
4	Longval=Long .**valueOf(** str);	Convert string to long object

Converting numeric String to primitive number using parsing methods:

1	int i=Integer.**parseInt (** str);	Convert string to primitive integers
2	long l=Long.**parseLong** (str);	Convert string to primitive long

Example

```
class Wrap1
{
public static void main(String args[])
{
        String      s=new String("123");
        int i=Integer.valueOf(s).intValue(); //typeValue() and valueOf()
method
        i=i+10;


        float f=Float.valueOf(s).floatValue();
        System.out.println("s="+s);

        System.out.println("i="+i);
        //f=100.0f;
```

```
        System.out.println("f="+f);

        Float f1=new Float(12.5f);  //constructor method
        Float f2=f1;
        float f3=f1+f2;
        System.out.println(f1.doubleValue());
        System.out.println(f2.doubleValue());
        System.out.println("sumoftwofloatobject="+f3);

        int i1=5;
        //Integer si=i1;
        Integer si=new Integer(i1);
        String s2=si.toString();
        String s1=new String(s2);
        System.out.println(s2.length());

}
}
```

2.8 Comment Syntax

Java has three types of comment lines:

1) Single line comments: start with // and extend to the end of the current line. This comment type is also present in C++ and in modern C

Syntax: // This is an end-of-line comment

2) Multiline comments: start with /* and end with */, they may cover multiple lines. This type of comment was derived from C and C++.

Syntax : /* This is a multi-line comment.
It may occupy more than one line. */

3) Documentation comments are processed by Javadoc tool to generate documentation from source files.

- it starts with /** and follows conventions defined by the Javadoc tool. Technically these comments are a special kind of traditional comment and they are not specifically defined in the language specification.

Syntax :
```
/**
 * This is a documentation comment.
 *
 * @author John Doe
 */
```

2.9 Garbage Collection

- Java runtime system performs memory management.
- Java uses constructor to initialize object (give space in memory).
- Java does not have destructor as in C++.
- Java system has garbage collector and finalizer method for memory de allocation for any object.

Finalizer method

- Finalizer releases the objects special memory.
- Constructor method is used to initialize an object when it is declared, this is called initialization. Java also supports finalization, which is opposite to initialization.
- Java run time is an automatic garbage collecting system, which automatically frees up the memory resources used by objects.
- But if objects may hold other non-object resources (such as file descriptor or windows character system).
- Garbage collector can't free these resources, to free these resources we must use Finalizer methods.
- **finalize( )** can be added to any class.
- Java calls that method whenever it is about reclaim space for that object.
- **finalize( )** method should explicitly define tasks to be performed.
- **Before an object gets garbage collected ,the garbage collector gives the object an opportunities to clean up itself through call to the object's finalize ( ) method , this is called finalization.**
- When there is no more reference to object, object is finalized and is then garbage collected.
- finalize ( ) method is member of object class.
- finalize ( ) is only called just prior to garbage collector.

System . runFinalization ( );

this method force object's finalization on it finalize all objects that are waiting to be garbage collected.

Syntax:

```
protected void finalize( )
{
```

```
        //finalization code
    }
```

Protected prevents access to finalize ( ) by code defined outside its class.

Garbage collector

- A Java runtime environment deletes objects periodically when it determines that they are no longer being used, this process is called ***garbage collection*.**
- It is done by **garbage collector** of java.
- Garbage collector only knows how to release memory allocated with new (means dynamically).
- An object is eligible for garbage collection when there are no more reference to that object.
- When garbage collector is ready to release the storage used for your object, it will first call finalize ( ) and only on the next pass garbage collection pass will reclaim the objects memory.
- Before an object gets garbage collected ,garbage collector gives the object an opportunity to clean up itself through a call to objects finalize( ) method.
- Running of garbage collector:

We can ask garbage collector to run at any time by calling gc( ) method of System class.

System.gc( );

2.10 Arrays of Primitive Data Types

- Java has eight basic (primitive) data type. We can create array of any primitive data type.

  Syntax: datatype Arrayname[ ] =new datatype[size];

  Where ,size is any decimal number, it defines no.. of element you want to store in that array.

Ex:
```
int a[ ]=new int [ 100 ];
byte  val [ ]=new byte [10] ;
float  weight[ ]=new float[10];
```

2.11 Types of Arrays

- Array is a fixed size sequential collection of elements having same data type.
- We can declare array without giving its size.(we do not have to provide array size at the time of declaration of array.)
- Three types of array :
  1) one dimensional
  2) two dimensional
  3) variable sized array(multi dimensional)

1) One dimensional array

- **Declaration of array:**

  type arrayname[ ] ; Ex: int a[ ];

  or

  type[ ] arrayname; Ex: int[ ] a;

- **Creation of array:**

  Arrayname=new type [size] ;

  Ex: a=new int [10] ;

- **Initialization of array:**

Compile time :

1) Arrayname[subscript]=value**;**

2) Type arrayname[ ]={ list of values};

Ex: int a [ ] ={1,2,3,4,5,6};

int a[0]=1;

int a[1]=2; and so on

Run time :

Using command line argument,readLine( )method,using Scanner class's methods

2) Two dimensional array

int a[ ] [ ] = new int [3] [4] ;

- **a is 3X4 array.**

int a[2][3]={0,0,0,1,1,1};

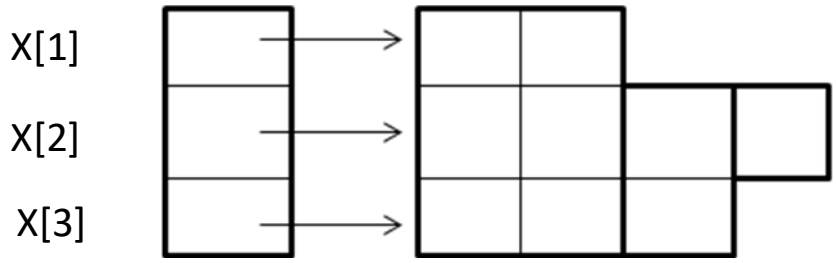
int a[2][3]={ {0,0,0} , {1,1,1} };

3) Variable sized array

- Java treats multidimensional array as "arrays of arrays"
- It is possible to declare a two dimensional.
- int a[ ][ ]= new int [3][ ]

```
X[0] = new int[2];
X[1] = new int[4];
X[2] = new int[3];
```

These statements create a two dimensional array as having different length for each row as follows:



Array length

- In java all arrays store the allocated size in variable named length.
- We can access length of array using length variable.

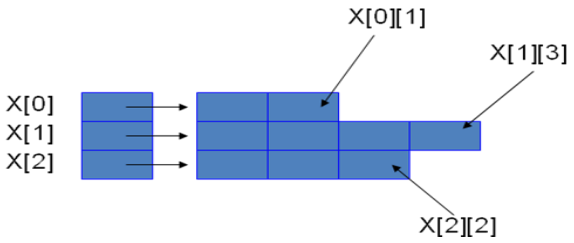
```
int size= a.length;
```

2.12 Creation, concatenation and conversion of a string, changing case of string, character extraction, String Comparison, String Buffer

- There is no string data type in Java. But Java has String class.
- We can create String class object for string handling.
- String represents a sequence of characters. The easiest way to represent a sequence of character in java is character array.
- E.g. :

```
char charArray  [ ]= new char[4];
    charArray [ 0 ]= 'J'
    charArray [ 1 ]='a' ,
    charArray [ 2 ]='v'
    charArray[ 3 ]='a'
```

- character array have the ability to query their length ,but they are not good enough to support the range of operations we want to perform on strings.(like to copy one character array into another array may require a lot of efforts)
- Java can handle these problem more efficiently using string.Charcater array is represented as follow:



String constructor

- In java strings are class objects which are implemented using two classes **String** and **String Buffer.**
- **Java string is an instantiated object of the String class.**
- All string literals in Java programs are implemented as instances of String class. Strings in Java are 16-bit Unicode.
- In JDK 1.5+ you can use String Builder, which works exactly like String Buffer, but it is faster and not thread-safe.

Characteristics / benefits of java's string:

- More reliable and predictable.
- No problem of bound checking same as in C.
- Java string is not character array and is not NULL terminated.

String declaration

We can declare and create string as follows:

1) String **stringname;**

stringname =**new** String("string");

Or

String str=new **String("** string**");**

E.g. : String **str** ;

str= new String ("VPMP"); or

String **str1**= new String ("Polytechnic");

2) The easiest way of creating a String object is using a string literal:

String **str1** = "Hello!"

3) using command line argument assign string.

String str=args[0];

- A **string literal** is a reference to a String object. Since a string literal is a reference, it can be manipulated like any other String referenceit can be used to invoke methods of String class.

Find string length:

- **length( )** method is used to count no. Of characters in any string.

int myLength = "Hello ".**length( );**

or

int myLength=str1.**length( );**

String concatenation:

1) Using (+) operator:

- The Java string can be concatenated using **(+) operator**, which has been overloaded for Strings objects.
- String concatenation is implemented through the StringBuffer class and its append method.

For example,

String **finalString** = "Hello" + "World;

Executed as,

String **finalString** =new StringBuffer ().**append** ("Hello").**append (**"World").**toString**();

2) concat ( ) method:

It concatenates two strings.

E.g. : String **str1**="Hello " ;
String str2="world";
str1.concat (str2); //output will be "HelloWorld"

String constructor:

String class provides three types of constructors to create String objects

1. **String ( ) :Creates** a new String object whose content is empty.**Str=" "**
2. **String (String s):** Creates a new String object whose content is same as the String object passed as an argument.

   Ex: str= **"str1"**
3. String also provides constructors that take byte and char array as argument and returns String object.

   String str=new String (charArray);

   Where charArray is array of type char which contains alphabets.

String array:

- We can create and use array that contain strings,

  String strArr [ ] =new **String** [3];
- It will create a strArr of size 3 to hold three string constants.
- Strings can be assigned to this array using element by element or more effectively using for loop.

String equality (comparison)

1) equals( ) method:

- String class overrides the **equals ( )** method of the Object class.
- It compares the content of the two string object and returns the boolean value accordingly.
- If both strings are **same** then returns "**true**" else "**false** ".
- equals ( ) method compares actual contents of two String objects.
- It differentiate Upper case and Lower case letters.

Ex:

String **str1**="Hello " ;
String **str2**="hello" ;
String **Str3** ="Hello";

Then

str1.eqauls (**str2**); // returns **false**
str1.eqauls (**str3**); // returns **true**

2) Using "==" operator:

- We can also compare two strings using "==".
- " == " compares the references not the actual contents of the String object;

Ex:

String **str1**="Hello " ;
String **str2**="hello" ;
String **Str3** ="Hello";

Then

If (**str1==str2**); // returns **true**
If (**str1 == str3**); // returns **true**

3) equalsIgnoreCase( ) :

- It compares strings which ignores the case of contents while comparing.
- It returns true if str1=str2, ignoring the case of characters.
- If both strings are **same** then returns "**true**" else "**false** ".

Ex:

String **str1**="Hello " ;
String **str2**="hello" ;
str1.eqaulsIgnoreCase (**str2**); //returns **true**

4) compareTo( ):

- It compares two Strings and returns an **int** value.
  - ✓ It returns value **0** (zero), if str1 is **equals** to **str2.**
  - ✓ It returns value **negative**, if str1 < **str2.**
  - ✓ It returns value **positive**, if str1 > str2**.**
- This method behaves exactly like the first method, if the argument object is actually a String object; otherwise, it throws a **ClassCastException.**

Ex:

```
String str1="Hello " ;
String str2="hello" ;
String Str3 ="Hello";

str1.compareTo(str2);          // returns -32
str1.compareTo(str3);          //returns  0
str2.compareTo(str1);          // returns  32
```

Searching a string

- String class provides **indexOf ( )** method which searches for the specified character inside the string object.
- This method has been overloaded.
- If the **search is successful**, then it **returns the index of the char** otherwise **-1** is returned.

1) **int indexOf(String str) :**
   - Finds the start index of the first occurrence of the substring argument in a String

   Ex: int position= str1.**indexOf** ("e"); // for "hello" it returns 1

2) **int indexOf(String str, int fromIndex)**
   - Finds the start index of the first occurrence of the substring argument in a String, starting at the index specified in the second argument

   Ex: int position= str1.**indexOf** ("e",3); // for "hello" it returns -1

3) **int indexOf(char c):**
   - Returns the index of first occurrence of the argument char

   Ex: int position= str1 .**indexOf(e); // for welcome it returns 1**

4) **int indexOf(int c, int fromIndex) :**
   - Returns the index of first occurrence of the argument char.

   Ex: int position= str1.**indexOf(e , 3); // for welcome it returns 6**

The String class also provides methods to search for a character or string in **backward direction**.

These methods are given below.

1. int **lastIndexOf**(int ch)
2. int **lastIndexOf**(int ch, int fromIndex)
3. int **lastIndexOf**(String str)
4. int **lastIndexOf**(String str, int fromIndex)

Replacing a character in string

- The replace method of String class can be used to replace all occurrences of the specified character with given character.

str. **replace** (char oldChar, int newchar)

Ex: **String** str1="Hello";
str1.**replace** ("H" ,"c"); // Gives string "cello" instead of "Hello"

Getting a string

- String class provides **substring ( )** method to extract specified portion of the given String. This method has been overloaded.
- If the index value is not valid, a **StringIndexOutOfBoundsException** is thrown.

Note:

A new String object containing the substring is created and returned. The original String won't be affected.

1) String **substring** (int startIndex)

- gives substring starting from **nth** character

Ex: String s1="welcome";
s1.substring(2) ; // gives **lcome**

2) String **substring** (int startIndex, int endIndex)

- gives substring starting from **nth** character up to mth character (not including **mth**)

Ex: String s1="welcome";
s1.substring(2,4) ; // gives **lc**

String conversion:

- String class provides set of **static** overloaded **valueOf ( )** method to convert **primitives types** and **object** into **strings.**

Object to string:

- static String **.valueOf**(Object obj)
- static String**. valueOf**(char[] character)

Primitive to string:

- static String .**valueO**f(boolean b)
- static String .**valueOf(**char c)
- static String .**valueOf**(int i)
- static String. **valueOf**(long l)
- static String .**valueOf**(float f)
- static String .**valueOf**(double d)

Manipulating a string

- String class provides following methods to manipulate character case in String.

1) toUpperCase ( ) : converts the string **str1** to all upper case.

Ex: **String** str1=”Hello”;
str2=str1.**toUpperCase( )**; // string in str2 will be HELLO

2) toLowerCase ( ) : converts the string **str1** to all lower case .

Ex**:** **String** str1=”VPMP”;
str2=str1.**toLowerCase( )**; // string in str2 will be vpmp

Note: Original String object is returned if none of the characters changed; otherwise new String object is constructed and returned.

3) trim( ) : This method removes white space from the **front and the end of a String**.

Ex: **String** str=" wel come ";
System.out.println(str.trim()); // output will be **wel come ,**no space before and after as original string str

4) int length( ) : Returns length of the String.

Ex: **String** str="hello";
System.out.println(str.length ( ));

5) charAt( ): Gives **n$^{th\ character}$** of str1.

Ex: **String** str="hello";
System.out.println(str.charAt(1)); // ouptu will be **e**

6) p**.toString( ):** Creates a string representation of the object p.

Difference between String and StringBuffer class

	String class	**StringBuffer** class
1	It creates string of fixed length.	It creates strings of flexible (variable) length.
2	Strings cannot be changed.	String created from this class can be modified in terms of length and content.
3	We cannot insert characters and substring in the middle of strings.	We can insert characters, substrings in the middle of the string or append another string to the end.

StringBuffer class

- **StringBuffer** class is peer class of **String**.
- It enables us to create **flexible length string** allow us to **perform manipulation on string whenever we want.**

Commonly used StringBuffer Methods:

1) str1.**setCharAt** (n, char ch): It modifies the **nth** character to **ch.**

Ex:
```
StringBuffer str=new StringBuffer("Cello");
str .setCharAt (0 , 'H');                    //output string Hello
```

2) str1.**append** (str2): Appends the string **str2** at the end of **str1.**

```
StringBuffer str=new StringBuffer("Hello");
str . append (" GoodMorning ");        //output string Hello GoodMorning
```

3) str1.**insert** (n,s2) :it inserts the string **str2 at** the position **n** of the string **str1.**

```
StringBuffer str=new StringBuffer("Hello Good Morning");
str.insert (5, " class ");
```

4) str1.**setLength**(n) : sets the length of the string **str1** to **n.**

```
str.setLength(30);
```

If **n < str1. length( )** then **str1** is **truncated**.

If **n > str1 .length ( )** then **zeros** are **added to str1.**

Example-1: Using Methods of String Buffer class

```
class  StringBuf
{
 public static  void main(String args[ ])
{
        StringBuffer str=new StringBuffer("Cello");

        //      Modifying characters
        str .setCharAt( 0 , 'H');
        System.out.println( "New string : " + str );

        //      append string
        str . append ("Good Morning ");
        System.out.println( "Appended string :" + str );

        //      insert string
        str.insert (5, " class ");
        System.out.println( "after insert string :" + str );
```

```
        //find string length
        int l=str.length();
        System.out.println( "String length="+l);

        //set length   of string
        str.setLength(30);
        System.out.println("string="+str);

        str.setLength(14);
        System.out.println("string="+str);
}
}
```

Output:

```
New string: Hello
Appended string: HelloGood Morning
After insert string: Hello class Good Morning
String length=25
String =Hello class Good Morning
String = Hello class Go
```

Example-2:

```
class  StringManipulation
{
 public static  void main(String args[ ])
{
StringBuffer str=new StringBuffer("Hello class");
System.out.println("Original string :" + str );

//get string length
System.out.println( "length of string =" + str.length() );
//Accessing characters in astring

for( int i=0 ;i< str.length ( ) ; i++ )
{     int p = i + 1 ;
            System.out.println("character at position :"+ p + "is " + str.charAt(i) );
}

//Inserting a string in the middle

String  str1 =new  String (str . toString());
int pos = str1 . indexOf(" class");
str . insert (pos , "ce ");
System .out.println( "Modified string  : "+ str );
```

```
//Modifying characters

str .setCharAt( 7 , '-');
System.out.println( "string now : " + str );

//Appending a string at the end

str . append (" Good Morning");
System.out.println( "Appended string :" + str );
}
}
```

Output:

```
Original String : Hello class
Length of string  = 11
Character at position : 1 is H


Character at position : 2 is e
Character at position : 3 is l
Character at position : 4 is l
Character at position : 5 is o
Character at position : 6 is
Character at position : 7 is c
Character at position : 8 is l
Character at position : 9 is a
Character at position : 10 is s
Character at position : 11 is s
Modified string : Helloce class
String now : Helloce-class
Appended string : Helloce-class Good Morning
```

Exapmle-3:

Write a program which reads two strings from keyboard ,and perform following:

1) find string length of both strings
2) concat both strings
3) convert both strings into Upper case
4) convert both strings into Lower case
5) replace character

```
class string1
{
    public static void main(String args[])
    {
        String s1 = new String(args[0]);
```

```
            String s2 = new String(args[1]);
            System.out.println("String1 is = " +s1);
            System.out.println("String2 is = " +s2);

            int len = s1.length();
            int len1 = s2.length();
            System.out.println("Length of String1 is = " +len);
            System.out.println("Length of String2 is = " +len1);

    System.out.println("Concate of String1 and String2 is = " +s1.concat(s2));
    System.out.println("Convert String1 into Uppercase = " +s1.toUpperCase());




    System.out.println("Convert String2 into Uppercase = " +s2.toUpperCase());
    System.out.println("Convert String1 into Lowercase = " +s1.toLowerCase());
    System.out.println("Convert String2 into Lowercase = " +s2.toLowerCase());
    System.out.println("After replace Character in String1 = " +s1.replace('a','o'));
        }
}
```

Output:

```
String1 is = Hello
String2 is = class
Length of String1 is = 5
Length of String2 is = 5
Concate of String1 and String2 is= Helloclass
Convert String1 into Uppercase =HELLO
Convert String2 into Uppercase =CLASS
Convert String1 into Lowercase =hello
Convert String2 into Lowercase =class
After replace character in string1= closs
```

Exapmle-4:

Write a program which uses comparisons methods for strings.

```
class string2
{
        public static void main(String args[])
        {
        /*      read string from command line(from user)
                 String str1=args[0];
                String str2=args[1];                              */
```

```
        String str1="Hello";
        String str2="hello";
        String str3=new String("Hello");
        if(str1.equals(str2))
        {
            System.out.println("Equal");
        }
        else
       {
                System.out.println("Not Equal");
       }

  System.out.println("compare between String1 and String3 using equals method = "
 +str1.equals(str3));

 System.out.println("compare between String1 and String2 using equalsognorecase
method = " +str1.equalsIgnoreCase(str3));

System.out.println("compare between String1 and String2 using compareto method = "
+str1.compareTo(str2));
System.out.println("compare between String1 and String3 using compareto method = "
+str1.compareTo(str3));
        }
  }
```

Output:

```
Not Equal
Compare between String1 and string3 using equals method = true
Compare between String1 and string2 using equalsIgnoreCase method = true
Compare between String1 and string2 using compareTo method = true
Compare between String1 and string3 using compareTo method = 0
```

Example:5

```
class Compare
{
public static void main(String args[])
{
        int count=0;
        String str1=args[0].toString();
        String str2=args[1].toString();
        for(int i=1;i<args.length;i++)
        {
                if(str1.compareTo(args[i])==0)
                                System.out.println("Both strings are same");
                else
```

```
                System.out.println("Both strings are different");
        }
    }
    }
```

Output: **javac Compare.java**
java compare india indiA
Both strings are different

Example: 5

Write a program which accepts (reads) strings from user and sort them into ascending order.

```
class SortString
{
    public static void main(String args[])
    {
    int len=args.length;
    String city[ ]=new String[len];

    for(int i=0;i<len;i++)
    {
        city[i]=args[i];
    }
     String temp;
    for(int i=0;i<len;i++)
        {
          for(int j=0;j<len-1;j++)
          {
            if(city[j].compareTo(city[i])>0)
            {
              temp = city[i];
              city[i] = city[j];
              city[j] = temp;
            }
          }
        }
       for(int i=0;i<args.length;i++)
       {
          System.out.println(city[i]);
       }
    }
```

} Here,ASCII code are compared for sorting

Output: **javac SortString.java**
java SortString Mumbai delhi banglore Baroda

Baroda
Mumbai
Delhi
banglore

Most commonly used methods of **String** class

No.	Method call	Action performed
1	S2=s1.**toLowerCase( );**	Converts the string s1 to all lowercase
2	S2=s1.**toUpperCase( );**	Converts the string s1 to all uppercase
3	S2=s1.**replace('x', 'y');**	Replace all appearance of x with y
4	S2=s1.**trim( )**	Remove white spaces at the beginning and end of the string s1
5	s1.**equals( )**	Returns **true** if s1 is equal to s2(case sensitive)
6	S1.**equalsIgnoreCase( )**	Returns **true** if s1 is equal to s2,ignoring the case of characters(upper or lower)
7	int l=s1.**length( )**	Gives the length of string s1
8	S1.**charAt( )**	Gives nth character of s1
9	S1.**compareTo(** s2 **)**	Returns negative ,if s1 < s2 , Positive , if s1 >s2, Zero is s1 equals s2
10	S1.**concat**( )	Concatenates s1 and s2
11	S1.**substring**(n)	Give substring starting from n^{th} character
12	S1.**substring**(n , m)	Give substring starting from n^{th} character up to mth(not including m^{th})
13	String. **valueOf(** p **)**	Creates a string object of the parameter p
14	p. **toString( )**	Creates string representation of the object p
15	S1.**indexOf ('x')**	Gives the position of the first occurrence of 'x' in the string s1
16	S1.**indexOf('x' ,** n)	Gives the position of 'x' that occurs after n^{th} position in the string s1
17	String **.valueOf**(variable)	Converts the parameter value to string representation

Most commonly used methods of **StringBuffer** class

No.	Method call	Action performed
1	s1.**setCharAt( );**	Modifies the n^{th} character to x
2	s1.**append(s2);**	Appends string to s2 to s1at the end
3	s1.**insert(n, s2);**	Insert the string s2 at the position n of the string s1
4	s1.**setLength(n)**	Sets the length of the string s1 to n. If n < s1.length( ) then s1 is truncated. If n > s1.length( ) then zeros are added to s1

2.13 Different Operators and mathematical functions:

- An operator is some special characters (symbol) that tells the computer to perform certain mathematical or logical manipulations.

Types of Operators

1) Arithmetic Operators
2) Relational operators
3) Logical Operators
4) Assignment Operators
5) Increment & Decrement Operators
6) Conditional Operators
7) Bitwise Operators
8) Special Operators

1) Arithmetic Operators

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo Divisions

- Ex: a+b, a-b, a*b, a%b, a/b
- Where a & b variables & known as operand.
- **Integer Arithmetic**: Where all operands are integer.
- **Real Arithmetic:** Where all operands are real.
- **Mixed-Mode Arithmetic:** Where one operand is real & the other is integer.

  Ex. 15 / 10.0 produce result 1.5

  While 15/10 produce result 1

2) Relational Operators

- Compare two quantities & depending on their relation.
- Relational operators are used in decision statements such as while & if to decide the course of action of a running program.

Operator	Meaning
<	Is less than
<=	Is less than or equal to
>	Is greater than
>=	Is greater than or equal to
==	Is equal to
!=	Is not equal to

3) Logical Operators

- C has three logical operators.

Operator	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT

- The Logical operator && and || are used when we want to test more than one condition and make decisions.
- Ex: a>b && x==10
- In logical AND operator is used to perform more than one condition and take one decision.
- In logical OR operator is used to perform more than one condition and take one decision.
- Logical NOT operator is used to negate the answer of operation.
-

4) Assignment Operators

- Used to assign the result of an expression to a variable.
- C has a set of "shorthand" assignment operators.

a=a+1	a+=1
a=a-1	a-=1
a=a*(n+1)	a*=n+1
a=a/(n+1)	a/=n+1
a=a%b	a%=b

5) Increment & Decrement Operators

- Increment Operator ++
- Decrement operator --
- The increment operator ++ adds 1 to the operand, while decrement operator -- subtracts 1.
- We use the increment and decrement statements in for and while loops.

Prefix increment operator

- A prefix operator first adds 1 to the operand(variable) and then the result is assigned to the variable on the left.
- **Syntax:** ++variable name

Postfix increment operator

- A postfix operator first assigns the value to the variable on the left and then increment the operand(variable).
- **Syntax:** variablename++

Prefix decrement operator

- A prefix operator first subtracts 1 to the operand (variable) and then the result is assigned to the variable on the left.
- **Syntax:** --variablename

Postfix decrement operator

- A postfix operator first assigns the value to the variable on the left and then decrement the operand(variable).
- **Syntax:** variablename--

6) Conditional Operator / Turnery Operator

- Conditional operator can be used in place of if-else statement.
- It is a two-way branching statement.
- Syntax: exp1 ? exp2:exp3
- where, exp1,exp2 & exp3 are expressions.

- First exp1 is evaluated.
- If it is true then the expression exp2 is evaluated & becomes the value of an expression.
- If exp1 is false, then exp3 is evaluated and its value becomes the value of the expression.

7) Bitwise Operator

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise Exclusive OR
~	One's complement
<<	Shift left
>>	Shift right
>>>	Shift right with zero fill

- Bitwise operators are used for Manipulation of data at bit level.
- Used for testing the bits, or shifting them right or left.
- Bitwise operators are used only for integer datatype.
- Not to be applied to float or double

Bitwise AND operator

- **Symbol:** &
- **Syntax:** variablename1 & variablename2;
- The bitwise AND operator is represented by a single ampersand (&) and is surrounded on both sides by integer expressions.
- The result of ANDing operation is 1 if both the bits have a value of 1; otherwise it is 0.

Bitwise OR operator

- **Symbol:** |
- **Syntax:** variablename1 | variablename2;
- The bitwise OR operator is represented by vertical bar(|) and is surrounded on both sides by integer expressions.
- The result of OR operation is 1 if at least one of the bits has a value of 1; otherwise it is 0.
- Example:

Bitwise Exclusive OR operator

- **Symbol:** ^
- **Syntax:** variablename1 ^ variablename2;
- The result of exclusive OR is 1 if only one of the bits is 1; otherwise it is 0.

One's complement

- **Symbol : ~**
- It complements bits 1to 0 and 0 to 1.

Right Shift with zero fill

- The right shift operation causes all the bits in the operand op to be shifted to the right by n positions.
- The rightmost n bits in the original bit pattern will be lost and the leftmost n bit positions that are vacated will be filled with 0s.

8) Special operators

1) instanceOf operator
2) dot operator

1. instanceOf operator :

- It is an object reference operator.
- It returns true if object on left side is instance of class given on right side.
- It defines whether object belongs to particular class or not.
- Ex:

person **instanceOf** *Student*

If object *person* belongs to class Student then returns true , else returns false.

2. Dot (.) operator :

- It is used to access the instance variable and method of class objects.
- It is also access classes and sub package from package.
  - person1.age;
  - person1.salary();

Mathematical Functions

Function	Description
abs(int a)	Returns the absolute integer value of a
ceil(double a)	Returns the "ceiling," or smallest whole number greater than or equal to a
cos(double x)	Returns the cosine value of an angle in radians
exp(double x)	Returns the exponential number e^x raised to the power of a
floor(double a)	Returns the "floor," or largest whole number less than or equal to a
log(double a)	Returns the natural logarithm (base e) of a
max(a,b)	Takes two values, a and b, and returns the greater of the two
min(a, b)	Takes two values, a and b, and returns the smaller of the two
pow(a, b)	Returns the number a raised to the power of b (a^b)
sin(double)	Returns the trigonometric sine of an angle
sqrt(double a)	Returns the square root of a
tan(double a)	Returns the trigonometric tangent of an angle
toDegrees(double)	Translates radians to degrees
toRadians(double)	Translates degrees to radians
acos(double a)	Returns the arc cosine of a, in the range of 0.0 through pi
asin(double a)	Returns the arc sine of a, in the range of -pi/2 through pi/2
atan(double a)	Returns the arc tangent of a, in the range of -pi/2 through pi/2
rint(double)	Returns the closest integer to the argument, but as a floating-point number

2.14 Decision and control statements:

Java statement

- Empty statement: used as a place holder
- Labeled statement: used as a label
- Expression statement:

  Seven types

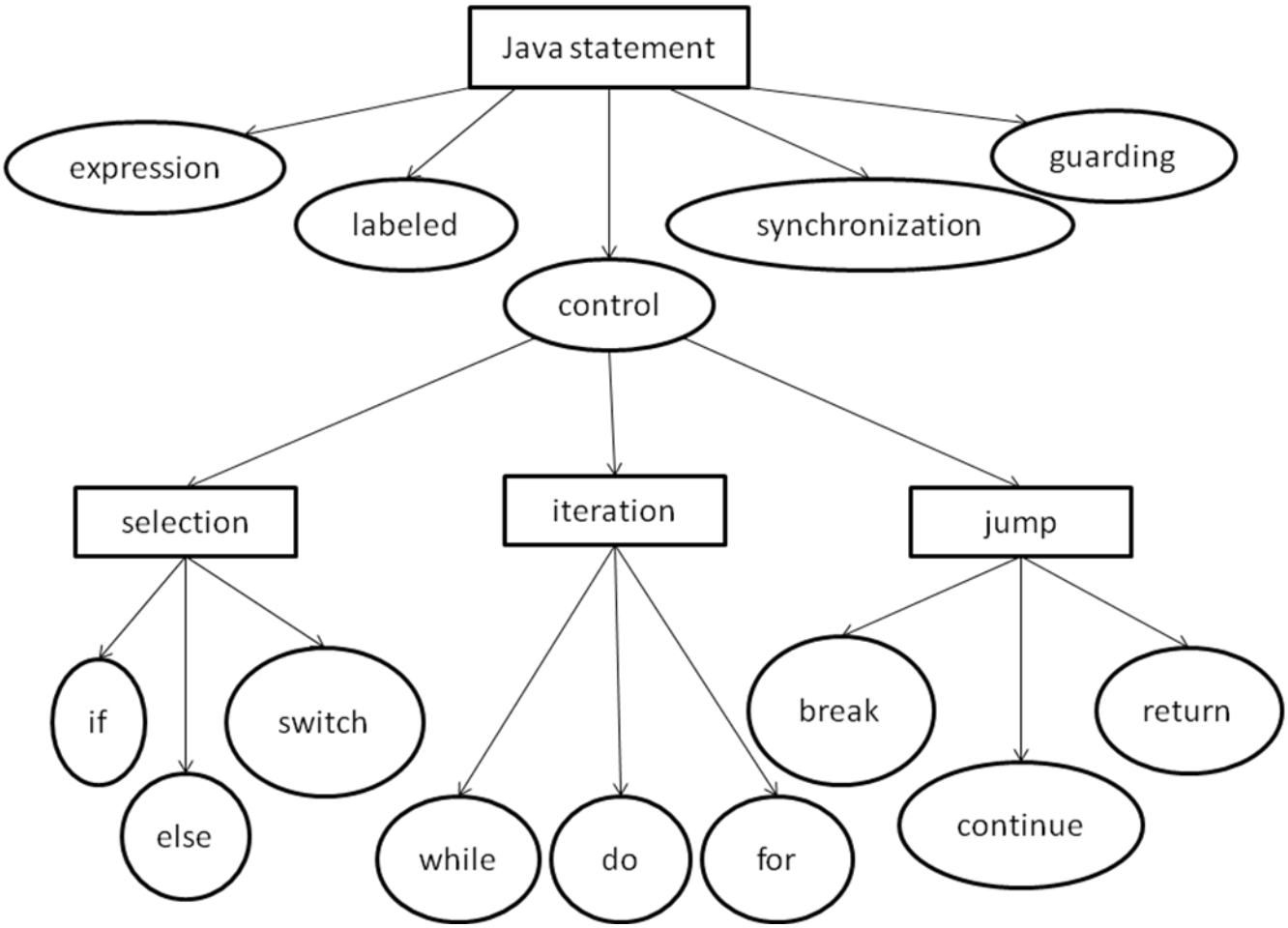
  2. Assignment
  3. Pre-Increment
  4. Pre-decrement
  5. Post-Increment
  6. Post-decrement

  7. Method call
  8. Allocation expression

- Selection statement:

  Three types : If , If-else , switch

- Iteration statement:
  Three types : While , do , For
- Jump statement:
  Four types: Break , Continue , Return , throw
- Synchronization statement: used for handling issues with multithreading
- Guarding statement: used for safe handling of code that may cause exceptions
  - Try, catch , finally
  -



Java control statements are of three type:

1) Selection Statement (if, if...else, switch)
2) Loops while, do-while, for)
3) Jump statements (break, continue, return & exit)

1) Selection statement:

1) simple if statement :

```
if ( test expression)
{
```

```
        //statement block;
    }
```

2) if...else statement :

```
    if ( test expression)
    {
        //true block statement block;
    }
    else
    {
        // false block statement;
    }
```

3) nesting of if.....else statement:

```
    if ( test condition-1)
    {
        if(condition-2)
        {
            statement-1;
        }
        else
        {
            Statement-2;
        }
    }
    else
    {
        Statement-3;
    }
```

5) else....if ladder

```
if ( condition-1)
{
    statement-1;
}
else if(condition 2)
{
    statement-2;
}
```

```
else if(condtion 3)
{
        statement-3;
}
.....
.....
.....
else
        default: default statement;
```

5) switch statement :

```
switch(integer or character value or expression)
{
case value-1:
                block-1
        break;
case value-2:
                block-2
        break;
.....
.....
default :
        default block;
        break;
}
```

2) Iteration statement:

1) while statement:

```
while(test condition)
{
        Body of the loop
}
```

2) do..while statement:

```
do
{
        Body of the loop
}while(test condition);
```

3) for loop:

```
for ( initialization  ;  test expression   ; increment  or decrement  )
{
        Body of the loop

}
```

Jump statement

1) break: It is used to exit from a loop or block

2) continue :it is used to skip some part of code and causes loop to be continued with the next iteration.

3) return :it is used to return values to calling function