Fundamentals of Object Oriented Programming &Java Evolution

1. INTRODUCTION

Programming languages can be classified into 3 primary types

1) **Unstructured Programming Languages:**
   - has sequentially flow of control
   - code is repeated throughout the program
2) **Structured Programming Languages:**
   - Has non-sequentially flow of control.
   - Use of functions allows for re-use of code.
3) **Object Oriented Programming**:
   - Combines Data& Action Together.

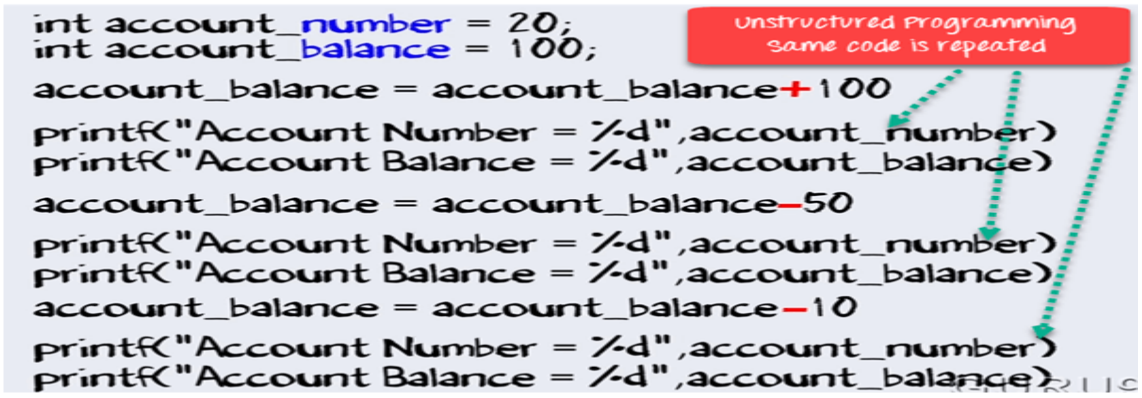
Let's understand these 3 types with an example.Suppose you want to create a Banking Software with functions like

- ➢ Deposit
- ➢ Withdraw
- ➢ Show Balance

Unstructured Programming Languages

- ➢ The earliest of all programming language were unstructured programming language.

- A very elementary code of banking application in unstructured Programming language will have two variables of one account number and another for account balance

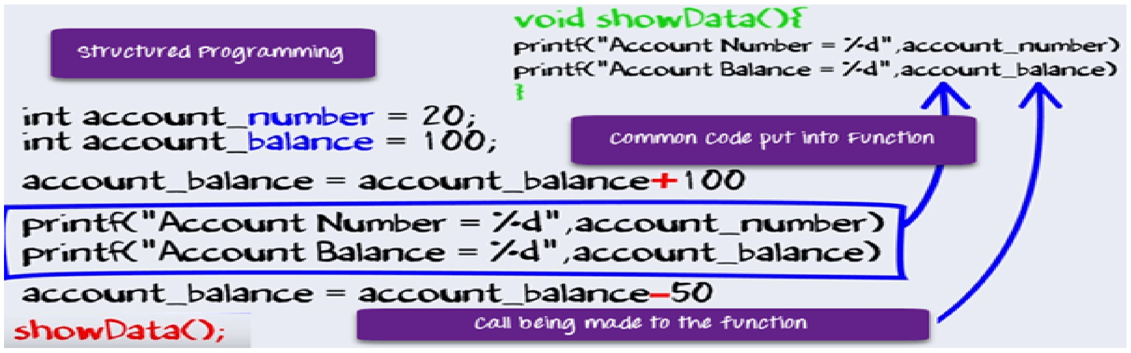


```
int account_number = 20;
int account_balance = 100;
account_balance = account_balance+100
printf("Account Number = %d",account_number)
printf("Account Balance = %d",account_balance)
account_balance = account_balance-50
printf("Account Number = %d",account_number)
printf("Account Balance = %d",account_balance)
account_balance = account_balance-10
printf("Account Number = %d",account_number)
printf("Account Balance = %d",account_balance)
```

For any further deposit or withdrawal operation – you will code repeat the same lines again and again.

Structured Programming

- With the arrival of Structured programming repeated lines on the code were put into structures such as functions or methods.
- Whenever needed, a simple call to the function is made.



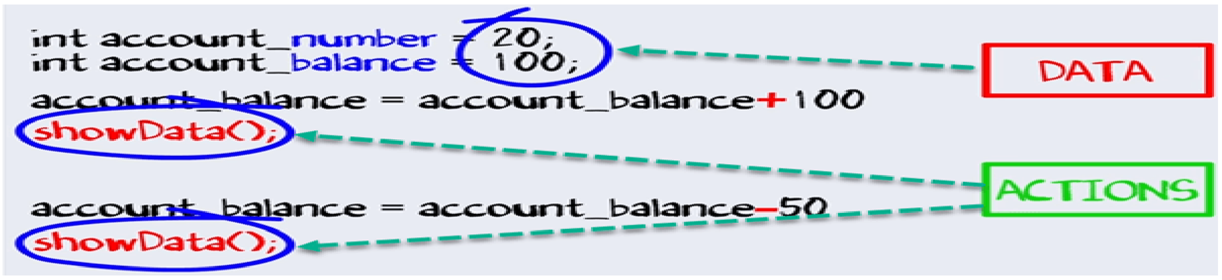
```
void showData(){
printf("Account Number = %d",account_number)
printf("Account Balance = %d",account_balance)
}

int account_number = 20;
int account_balance = 100;
account_balance = account_balance+100
printf("Account Number = %d",account_number)
printf("Account Balance = %d",account_balance)
account_balance = account_balance-50
showData();
```

Structured Programming
Common code put into Function
Call being made to the function

Object-Oriented Programming

- In fact, having data and performing certain operation on that data is very basic characteristic in any software program.
- Experts in Software Programming thought of combining the Data and Operations.
- Therefore, the birth of Object Oriented Programming which is commonly called OOPS.



```
int account_number = 20;
int account_balance = 100;
account_balance = account_balance+100
showData();

account_balance = account_balance-50
showData();
```

2. OBJECT ORIENTED PARADIGM

- The major objective of object-oriented approach is to eliminate some of the flaws encountered in the procedural approach.
- OOP treats data as a critical element in the program development and does not allow it to flow freely around the system.
- It ties data more closely to the to the functions that operate on it and protects it from unintentional modification by other functions.
- OOP allows us to decompose a problem into a number of entities called Objects and then build data and functions (known as methods in Java) around these entities.
- The combination of the data and the methods make up an object

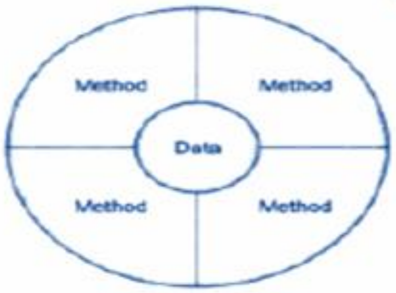


Fig. 1.1 Object = Data + Methods

The data of an object can be accessed only by the methods associated with that object. However, methods of one object can access the methods of other objects. Some of the features of object-oriented paradigm are:

1) Emphasis is on data rather than procedure.
2) Programs are divided into what are known as Objects.
3) Data Structures are designed such that they characterize the objects.
4) Methods that operate on the data of an object are tied together in the data structure.
5) Data is hidden and cannot be accessed by external functions.
6) Objects may communicate with each other through methods.

7) New data and methods can be easily added wherever necessary.
8) Follow bottom-up approach in program design.

3. BASIC CONCEPTS OF OBJECT ORIENTED PROGRAMMING

- Object Oriented Programming is a programming concept that works on the principle that objects are the most important part of your program.
- It allows users create the objects that they want and then create methods to handle those objects.
- Manipulating these objects to get results is the goal of Object Oriented Programming. Object Oriented Programming popularly known as OOP, is used in a modern programming language like Java

1) Object

Any Real world entity that has state and behavior is called Object. (0r) **Objects have states and behaviors**. Example: A dog has states - color, name, breed as well as behaviors – wagging the tail, barking, eating. Other Examples of Objects are Apple, Orange, Table, Fan etc..In Java, An Object is an instance of a Class.

2) Class

Collection of similar objects is called class. For Example Apple, Orange, Mango objects are grouped into a class called "Fruits" where as apple, table, fan objects cannot be grouped as a class because they are not similar objects. It is only an logical component and not the physical entity. Other example, if you had a class called "Expensive Cars" it could have objects like Mercedes, BMW, Toyota, etc. Its properties (data) can be price or speed of these cars. While the methods may be performed with these cars are driving, reverse, braking etc.In Java, Class defines data and methods that manipulate the data.

3) Inheritance

Inheritance is an OOPS concept in ***which one object acquires the properties and behaviors of the parent object***. It's creating a parent-child relationship between two classes. It offers robust and natural mechanism for organizing and structure of any software.

4) Polymorphism

Polymorphism refers to the ability of a variable, object or function to take on multiple forms. Or it refers as "one interface and many forms " (or) For example, in English, the verb "run" has a different meaning if you use it with "a laptop," and "a foot race". Here, we understand the meaning of "run" based on the other words used along with it. The same also applied to Polymorphism.

5) Abstraction

Abstraction is a process of hiding the implementation details from the user, only the functionality will be provided to the user For example, while driving a car, you do not have to be concerned with its internal working. Here you just need to concern about parts like steering wheel, Gears, accelerator, etc. Abstraction can be achieved using Abstract Class and Abstract Method in Java.

6) Encapsulation

Encapsulation is a principle of wrapping data (variables) and code together as a single unit. In this OOPS concept, the variables of a class are always hidden from other classes. It can only be accessed using the methods of their current class. For example - in school, a student cannot exist without a class.

4) BENEFITS OF OOPs

OOP offers several benefits to both the program designer and the user. The principal advantages are:

- OOP offers easy to understand and a clear modular structure for programs.
- Objects created for Object-Oriented Programs can be reused in other programs. Thus it saves significant development cost.
- Large programs are difficult to write, but if the development and designing team follow OOPS concept then they can better design with minimum flaws.
- It also enhances program modularity because every object exists independently.
- Through inheritance, we can eliminate redundant code and extend the use of existing classes.
- We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch. This leads to saving of development time and higher productivity.
- The principle of data hiding helps the programmer to build secure programs that cannot be invaded by code in other parts of the program.
- It is possible to have multiple objects to coexist without any interference.
- It is possible to map objects in the problem domain to those objects in the program.
- It is easy to partition the work in a project based on objects.

- The data- centered design approach enables us to capture more details of a model in an implementable form.
- Object-oriented systems can be easily upgraded from small to large systems.
- Message passing techniques for communication between objects make the interface descriptions with external systems much simpler.
- Software complexity can be easily managed.

5) Applications_of OOP

OOP is one of the programming buzzword today. There appears to be a great deal of excitement and interest among software engineers in using OOP. Applications of OOP are beginning to gain has been in the area of user interface design such as windows. There are hundreds of windowing systems developed using OOP techniques.

Real business system are often much more complex and contain many more objects with complicated attributes and methods. OOP is useful in this type of applications because it can simply a complex problem. The promising areas for application of OOP includes:

1) Real-time systems
2) Simulation and modeling
3) Object-oriented databases
4) Hypertext, hypermedia and expertext.
5) AI and expert systems
6) Neural networks and parallel programming
7) Decision support and office automation systems.
8) CIM/CAD/CAD system

It is believed that the richness of OOP environment will enable the software industry to improve not only the quality of software systems but also its productivity. Object-oriented technology is certainly changing the wy software engineers think,analyse,design and implement systems today.

6. Java History

In 1990, sun Microsystems has conceived a project to develop software for consumer electronic devices like TVS and VCRs that could be controlled by a remote. This project was called *Stealth Project* but later its name was changed to *Green project*.

In January of 1991, Bill Joy, James Gosling, Mike Sheradin , Patrick Naughton, and several others met in Colorado to discuss this project. Mike sheradin was to focus on business development, Patrick naughton was to begin work on graphic system, and James Gosling was to identify the proper programming language for the project. Gosling thought c and c++ could be used to develop the project. But the problem he faced with them is that they were system dependent languages and hence could not be used on various processors, which the electronic devices might use. So he started developing a new language, which was completely system independent. This language was initially called Oak. Since this name was registered by some other company, later it was changed to Java.

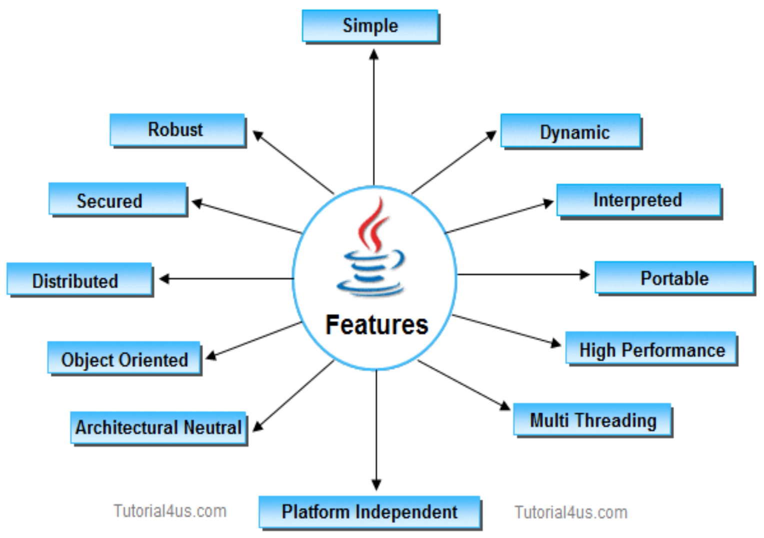
Why the name Java ? James Gosling and his team members were consuming a lot of coffee while developing this language. They felt that they were able to develop a better language because of the good quality coffee they consumed. So the coffee had its own role in developing this language and good quality was exported to the entire world from a place called "Java Island". Hence they fixed the name of the place for the language as Java. And the symbol for Java Language is Coffee cup and saucer.

Java is related to c++, which is direct decendent of C. Much of the characteristics of java is inherited from c and c++. From c, Java derives its syntax. Many of java's object oriented features were influenced by c++.

Year	Development
1990	Sun Microsystems decided to develop special software that could be used to manipulate consumer electronic devices. A team of Sun Microsystems programmers headed by James Gosling was formed to undertake this task.
1991	After exploring the possibility of using the most popular object-oriented language C++, the team announced a new language named "Oak"
1992	The team demonstrated the application of their new language to control a list of home appliances using a hand-held device with a tiny touch – sensitive screen
1993	With the advent of Internet and Web, the team came up with the idea of developing Applets that could run on all computers connected to the Internet
1994	The team developed a web browser called "HotJava" to locate and run applet programs on Internet
1995	Oak was renamed "Java" , due to some legal issues
1996	Java Established itself not only as a leader for internet programming but also as a general purpose, Object-Oriented programming language

7. Java Features

Features of a language are nothing but the set of services or facilities provided by the language vendors to the industry programmers. Some important **features of java** are;

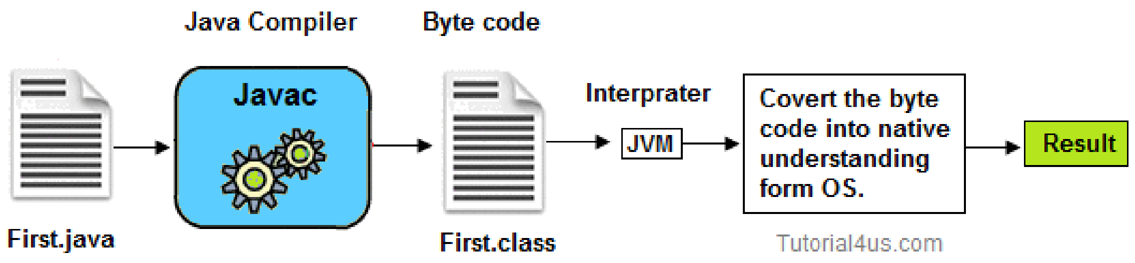


a) Simple
b) Platform Independent
c) Architectural Neutral
d) Portable
e) Multi Threading
f) Distributed
g) Networked
h) Robust
i) Dynamic
j) Secured
k) Interpreted & High Performance
l) Object Oriented

a). Simple
According to sun, Java language is Simple because

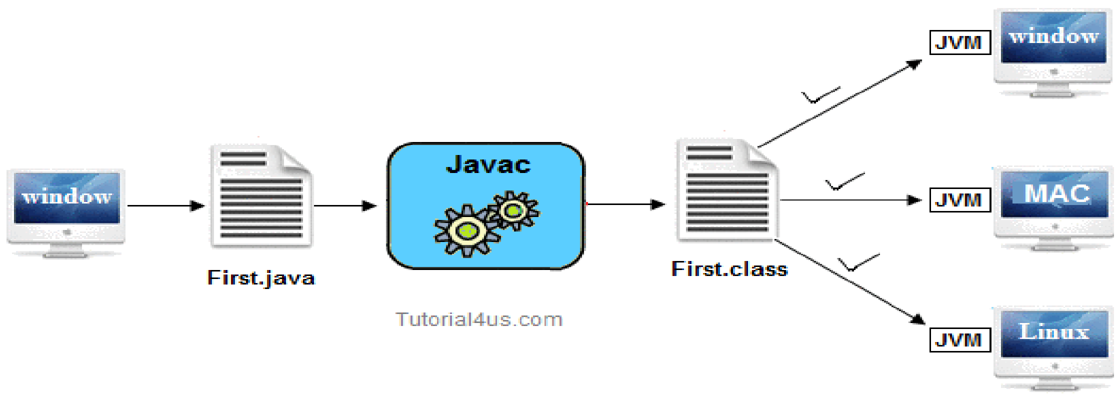
- Most of the concepts are drew from c++ thus making Java Learning Simple i.e Because java inherits the C++ syntax and many of the Object-Oriented features of C++, most programmers have trouble learning java.

- It is **free from pointer** due to this execution time of application is improve. [whenever we write a Java program without pointers then internally it is converted into the equivalent pointer program].
- It have **Rich set of API** (application protocol interface).
- It have **Garbage Collector** which is always used to collect un-Referenced (unused) Memory location for improving performance of a Java program.
- It contains user friendly syntax for developing any applications.



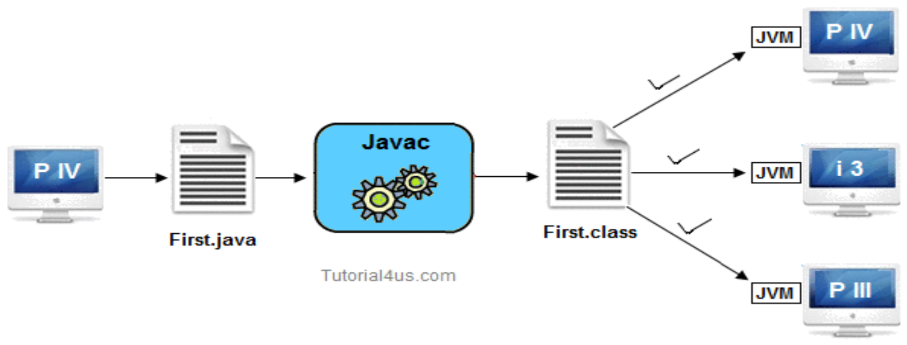
b) Platform Independent

- A Language or technology is said to be platform independent if and only if which can run on all available operating systems with respect to its development and compilation. (Platform represents O.S).



c) Architectural Neutral

Architecture represents processor. A Language or Technology is said to be Architectural neutral which can run on any available processors in the real world without considering there architecture and vendor (providers) irrespect to its development and compilation. The languages like C, CPP are treated as architectural dependent.



d) Portable

- If any language supports platform independent and architectural neutral feature known as portable.
- The languages like C, CPP, Pascal are treated as non-portable language where as Java is a portable language, According to SUN microsystem.
- Java Programs can be easily moved from one computer system to another, anywhere and anytime.
- Changes and Upgrades in Operating Systems, Processors and System resources will not force any changes in Java Programs
- This is the reason why Java has become a popular language for programming on Internet which interconnects different kinds of systems worldwide.

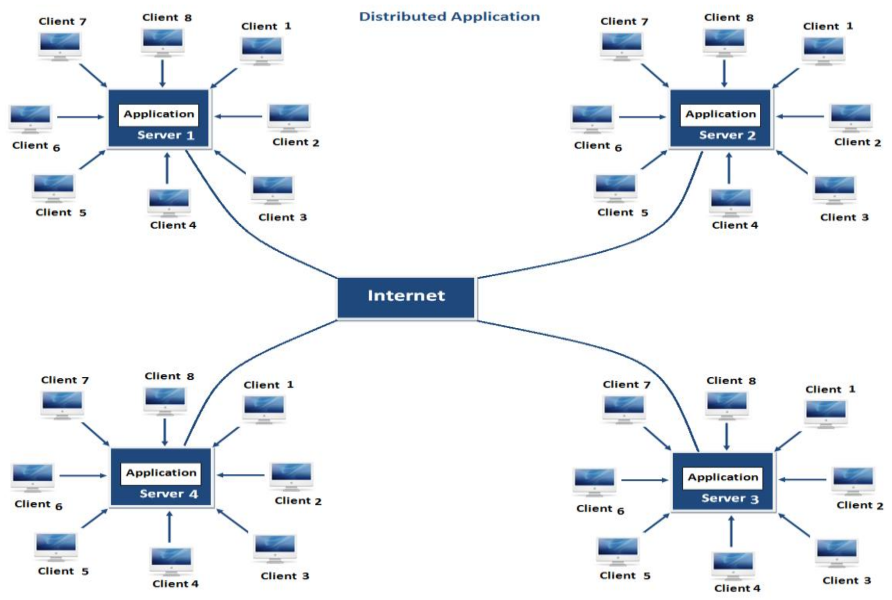
Portability = platform independent + architecture

e) Multithreaded

Multithreaded means handling multiple tasks simultaneously. This means that we need not wait for the application to finish one task before beginning another. This feature greatly improves the interactive performance of graphical applications. When any Language execute multiple thread at a time that language is known as multithreaded Language. Java supports multithreaded programs.

f) Distributed

Using this language we can create distributed application. RMI and EJB are used for creating distributed applications. In distributed application multiple client system are depends on multiple server systems so that even problem occurred in one server will never be reflected on any client system.



Note: In this architecture same application is distributed in multiple server system.

g) Networked

It is mainly design for web based applications, J2EE is used for developing network based applications.

h) Robust

Simply means of Robust is strong. It is robust or strong Programming Language because of its capability to handle Run-time Error, automatic garbage collection, lack of pointer concept, Exception Handling. All these points makes It robust Language. To better understand how java is robust, consider two of the main reasons for program failure

Memory management can be difficult, tedious task in traditional programming environments. For Example, in C/C++, the programmer must manually allocate and free all dynamic memory. This sometimes leads to problems, because programmers will either forget to free memory that has been previously allocated or, worse, try to free some memory that another part of their code is still using. Java virtually eliminates these problems by managing memory allocation and deallocation for you.(deallocation is completely automatic, because java provides garbage collection for unused objects).

2. Exceptional Conditions in traditional environments often arises in situations such as division by zero or file not found and they must be managed

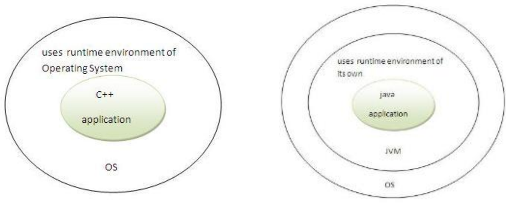
with clumsy and hard-to-read constructs. Java helps in this area by providing object-oriented exception handling.

I) Dynamic

It support Dynamic memory allocation due to this memory wastage is reduce and improve performance of application. The process of allocating the memory space to the input of the program at a run-time is known as dynamic memory allocation, To programming to allocate memory space by dynamically we use an operator called 'new'. 'new' operator is known as dynamic memory allocation operator.

J) Secure

Java is more secured language compare to other language. Security becomes an important issue for a language that is used for programming on Internet. Threat of viruses and abuse of resources is everywhere. Security in Java is achieved by running Java programs inside the Java Virtual machine i.e not allowing it to access to other parts of the computer.


uses runtime environment of Operating System
C++
application
OS
uses runtime environment of its own
java
application
JVM
OS

K) Interpreted & High performance

The Just-In-Time (JIT) compiler is a component of the Java™ Runtime Environment that improves the performance of Java applications at run time. The **JIT compiler** is enabled by default, and is activated when a Java method is called. The **JIT compiler** compiles the bytecodes of that method into native machine code, **compiling** it "**just in time**" to run. When a method has been compiled, the JVM calls the compiled code of that method directly instead of interpreting it.

L) Object Oriented

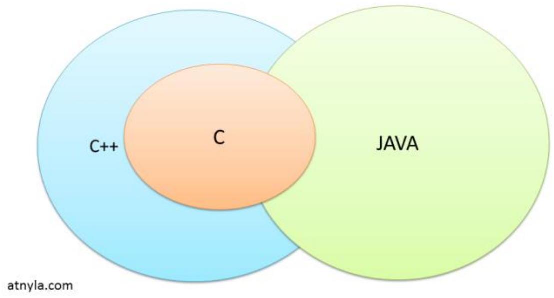
Object means a real word entity such as pen, chair, table etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

- Object

- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

8. How Java Differs from C and C++

Although Java was modeled after C and C++ languages, it differs from C and C++ in many ways. Java does not incorporate a number of features available in C and C++. For the benefit of C and C++ programmers, we point out here a few major differences between C/C++ and Java language.



Java also adds some new features, while C++ is a superset of C, Java is neither a superset nor a subset of C or C++.

Java and C

Java is not lot like C but the major difference between Java and C is that Java is an object-oriented language and has a mechanism to define classes and objects. In an effort to build a simple and safe language, the Java team did not include some of the C features in Java.

C Programming	Java Programming
It does include the unique statement keywords sizeof, and typedef.	It does not include the C unique statement keywords sizeof, and typedef.
It contain the data type struct and	It does not contain the data type struct and

C Programming	Java Programming
union.	union.
It define the type modifiers keywords auto, extern, register, signed, and unsigned.	It does not define the type modifiers keywords auto, extern, register, signed, and unsigned.
It supports an explicit pointer type.	It does not support an explicit pointer type.
It has a preprocessor and therefore we can use # define, # include, and # ifdef statements.	It does not have a preprocessor and therefore we cannot use # define, # include, and # ifdef statements.
It requires that the functions with no arguments, with the void keyword	It requires that the functions with no arguments must be declared with empty parenthesis, not with the void keyword
C has no operators such as instanceof and >>>.	Java adds new operators such as instanceof and >>>.
C adds have a break and continue statements.	Java adds labeled break and continue statements.
C has no object-oriented programming features.	Java adds many features required for object-oriented programming.

Java and C++

Java is a true object-oriented language while C++ is basically C with object-oriented extension. That is what exactly the increment operator ++ indicates. C++ has maintained backward compatibility with C. Is is, therefore, possible to write an old style C program and run it successfully under C++. Java appears to be similar to C++ when we consider only the “extensions” part of C++. However, some object -

oriented features of C++ make the C++ code extremely difficult to follow and maintain.

Listed below are some major C++ features that were intentionally omitted from Java or significantly modified.

C++ Programming	**Java Programming**
It support operator overloading.	It does not support operator overloading.
It support has template classes.	It does not have template classes as in C++.
It supports multiple inheritances of classes.	It does not support multiple inheritances of classes. This is accomplished using a new feature called "Interface".
It supports global variables.	It does not support global variables. Every variable and method is declared within classes and forms part of that class.
It supports pointers.	It does not use pointers.
It does not support destructor function with a finalize() function.	It has replaced the destructor function with a finalize() function.
There are header files in Java.	There are no header files in Java.

9. Java Environment

Java Environment include a Large number of

Development tools and

Hundreds of Classes and Methods.

1) The Development Tools are part of the system known as Java Development Kit(JDK)
2) And the classes and methods are part of Java Standard Library(JSL) also known as Application Programming Interface.

1) Java Development Kit

The Java Development Kit comes with a collection of tools that are used for developing and running java programs. They include

- appletviewer (for viewing java applets
- javac (java compiler)
- java (java interpreter)
- javap (java dissembler)
- javah (for c header files)
- javadoc(for creating html documents)
- jdb (java debugger)

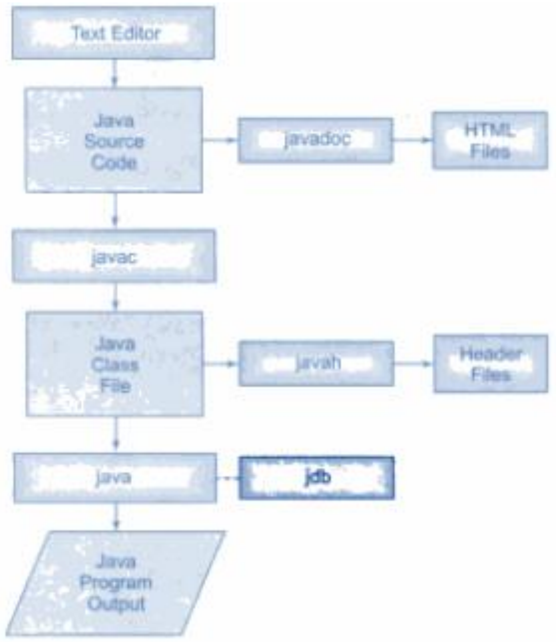
Tool	**Description**
appletviewer	Enables us to run java applets
java	Java Interpreter, runs application
javac	Java compiler, which translates java source code into byte code
javadoc	Creates HTML_format documentation from java source code files
javah	Produces header files for use with native methods
javap	Java dissembler, which enables us to convert bytecode files into a program description
jdb	Java debugger, which helps us to find errors in our program

2) API

The Java Standard Library (or API) includes hundreds of classes and methods grouped into several functional packages . Most commonly used packages are:

- Language Support Package : A collection of classes and methods required for implementing basic features of java.
- Utilities Package : A Collection of classes to provide utility functions such as date and time functions.
- Input/output package: A collection of classes required for input/output manipulation.
- Networking Package: A collection of classes for communicating with other computers via internet.
- AWT Package: The Abstract Window Tool Kit Package contains classes that implements platform-independent graphical user interface.

- Applet Package: This includes a set of classes that allows us to create Java Applets.



Process of building and running Java application programs

10. CONSTANTS

- Entities that do not change their values in a program are called constants or literals.
- Java literals are classified into 5 types:
  1. Integer Literals
  2. Floating Point Literals
  3. Character Literals
  4. Boolean Literals
  5. String Literals

1. Integer Literals

- A Whole number is called an integer. Eg: 25,27 etc… are integers
- Java supports 3 types of integer literals **decimal, octal, hexadecimal.**
- 25,27 are example of decimal integer literals.
- Octal integer literals start with 0 and are followed by octal digits 0 to 7. Eg: 0, 037, 032374 are octal integer literals
- Hexadecimal integer literals start with OX and are followed by hexadecimals digits 0 to 9, A to F. Eg: 0*29, 0*2AB9 are hexadecimal integers literals.

2. Floating Point Literals

- Numbers with decimal point and fractional values are called floating point literals.
- They can be expressed in either standard or scientific notation
- Standard notation consists of a whole number component followed by a decimal point followed by a fractional component.
- Scientific Notation uses a standard notation, floating point numbers plus a suffix that specifies a power of 10 by which the number is to be multiplied. (OR)
- A Floating point number followed by letter E(or e) and a signed integer. Eg: 6.237E-35 stands for 6.237×10^-35.
- Floating point literals in java defaults to double precision.

3. Boolean Literals

- In java, Boolean literals take two values false or true.
- These two values are not related to any numeric value as in C or C++.
- The Boolean value true is not equal to 1 and false is not equal to 0.

4. Character Literals

- Single characters in java are called character literals
- In java characters belong to 16-bit character set called Unicode.
- Java characters literals are written within a pair of single quote. EG: ‘a’, ‘z’ represent character literals.
- Further in certain occasions, a character like single quote(‘) itself is to be written as character literal.
- To represent such characters, java provides a set of character literals called escape sequence.

Escape Sequence Characters

Escape Sequence	**Description**
\ddd	Octal character represented in add
\uxxx	Hexadecimal unicode character
\’	Single quote
\”	Double quote
\\	Back slash
\r	Carriage return
\n	New line
\f	Form feed
\t	Tab
\b	Backspace

i.e there is also a mechanism for directly entering the value of a character in octal or hexadecimal. For octal notation, use the backslash followed by the 3 digit number. Eg: '\144' is the letter 'a'.

- For Hexadecimal you enter a backslah is (\u) then exactly four hexadecimals digits . Eg: '\40061' is the ISO-latin-1 'a'

5. String Literals

- A Sequence of characters written within a pair of double quote is called String Literal. Eg: " This is String"
- String Literals are to be started and ended in one line only.

11. Data Types

Every variable in java has a Data type. Data types specify the size and type of values that can be stored. Data types in java under various categories are shown below.

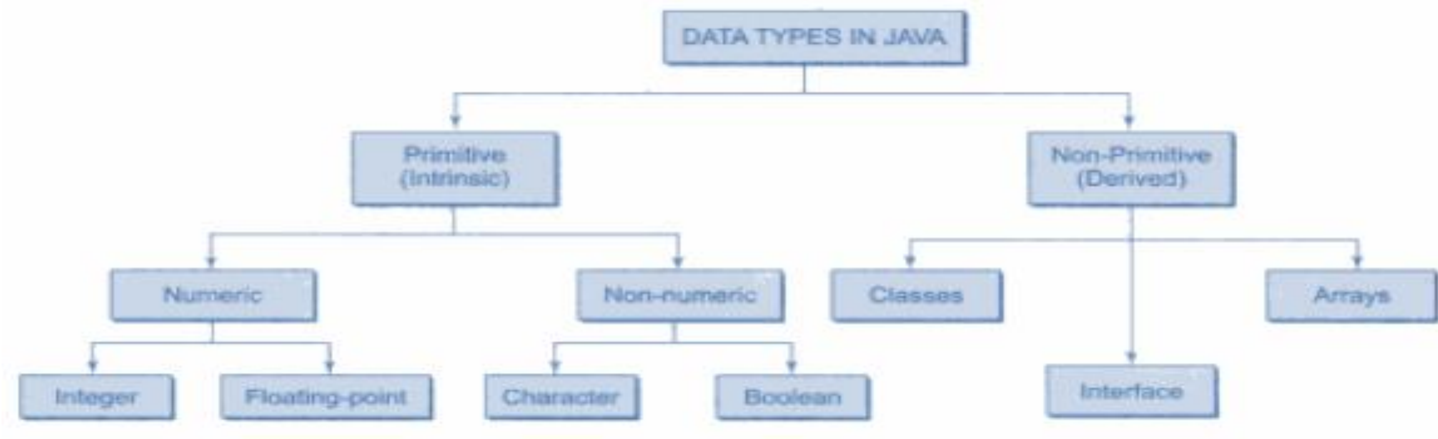


Fig. 4.2 *Data types in Java*

A) Primitive data types

Primitive data types are those whose variables allows us to store only one value but they never allows us to store multiple values of same type. This is a data type whose variable can hold maximum one value at a time.

```
Example
int a; // valid
a=10; // valid
a=10,20,30; // invalid
```

Here "a" store only one value at a time because it is primitive type variable.

B) Non Primitive Data Types or Derived

Derived data types are those which are developed by programmers by making use of appropriate features of the language. User defined data types related variables allows us to store multiple values either of same type or different type or both. This is a data type whose variable can hold more than one value of dissimilar type, in java it is achieved using class concept.

Note: In java derived or user defined data type combined name as reference data type.

Example
Student s = new Student();

Java defines Eight Primitive types of data.They are:

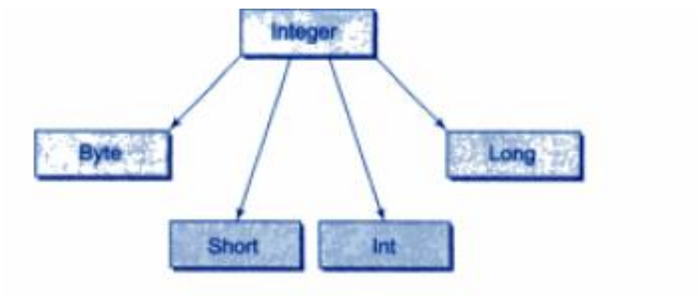
Byte
Short
Int
Long
Char
Float
Double
Boolean

The primitive types are also commonly referred to as simple types.These eight primitive types put in flow groups. They are;

1. Integer types
2. Floating point types
3. Character type
4. Boolean type

1. Integer Types

- This group includes byte, short, int and long, which are for whole-valued signed numbers.



- The width of each type is defined by Java language and do not depend on the machine in which the program is executed.
- The width and ranges of these integer types vary widely as shown in the below table.

Name	Width	Range
Long	64	-9,223,372,036,854,775,808 TO 9,223,372,036,854,775,807

Int	32	-2.147,483,648 to 2.147,483,647
Short	16	-32,768 to 32,767
Byte	8	-128 to 127

Byte

- The smallest integer type is byte
- This is a signed 8-bit type that has a range from -128 to 127
- Byte variables are declared by use of the byte keyword
- Eg: byte a,b;

Note: Languages such as a c and c++ allow the size of an integer to vary based on execution environment. However java is different because of java's portability, requirement all data types have a strictly defined range. For example an int is always 32 bits, regardless of particular platform

Short

- Short is a signed 16 bit type
- It has a range from -32,768 to 32,767
- It is probably the least used java types
- The short type integers are declared by the keyword short
- Eg: short s, t;

Int

- The most commonly used integer type is int.
- It is a signed 32-bit type that has a Range from -2.147,483,648 to 2.147,483,647
- Although we might think that using a byte or short would be more efficient than using an int in situations in which the larger range of an int is not needed this may not be the case

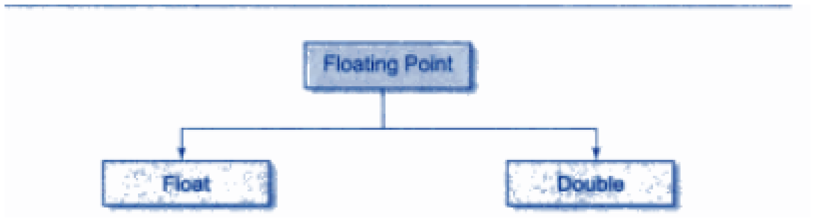
  The reason is that when byte and short values are used in an expression they are promoted to int when the expression is evaluated.
- Therefore, int is often the best choice when an integer is needed

Long:

- Long is a signed 64-bit type and is useful for those occasions where an Int type is not large enough to hold the desired values.
- The range of a long is Quite large
- This makes it useful when big, whole numbers are needed

2. Floating Point Types

- This group includes float and double which represent numbers with fractional precision.

- 

- There are two kinds of floating point types, float and double, which represent single and double precision numbers, respectively.
- Their width and ranges are shown below.

Name	Width in bits	Approximate range
Double	64	4.9e-324 to 1.8e+308
'//[Float	32	1.4e-045 to 3.4e+038

Float

- The type float specifies a single precision value that uses 32 bits of storage.
- Single precision is faster on some processors and takes half as much space as double precision, but will become imprecise when the values are either very large or very small.
- Variables of type float are useful when you need a fractional component but don't require a large degree of precision.

```
float lowtemp,  hightemp
```

Double

- Double precision, as denoted by the double keyword, uses 64 bits to store a value.
- When you need to maintain accuracy over many iterative calculations, or manipulating large valued numbers, double is the best choice.

```
double pi, r, a
```

Note: Single precision called "float" this is a binary format that occupies 32 bits(4 bytes) and its significant has a precision of 24 bits(about 7 decimal digits).
Double precision called "double". This is a binary format that occupies 4 bits (8 bytes) and its significant has a precision of 53 bits about 16 decimal digits.
Note: By default, all floating point numbers are treated as double.
Note: The range of data types is defined by Java Language and does not depend on the computer on which the data are generated.

3.Characters

- In Java, the data type used to
- store characters in char.
- Char in java is not the same as char in C or C++.
- C/C++ char is a 8-bit type whereas java char is a 16-bit type.

Why Java take 2 byte of memory for store character ?

Java support more than 18 international languages so java take 2 byte for characters, because for 18 international language 1 byte of memory is not sufficient for storing all characters and symbols present in 18 languages. Java supports Unicode but c support ascii code. In ascii code only English language are present, so for storing all English latter and symbols 1 byte is sufficient. Unicode character set is one which contains all the characters which are available in 18 international languages and it contains 65536 characters

5. BooleanTypes

- Java has a primitive type called Booleans, for logical values.
- It can have only one of two possible values true or false.

12. Variables

A Variable is an identifier that denotes a storage location used to store a data value. (or) Variables are the names of storage locations. Unlike constants that remain unchanged during the execution of a program, a variable may take different values at different tmes during the execution of the program.

Variable names may consists of alphabets, digits, the underscore and dollar characters, subject to the following conditions.

1. They must not begin with a digit
2. Uppercase and Lowercase are distinct. This means that the variable Total is not the same as total or TOTAL.
3. It should not be a keyword
4. White space is not allowed
5. Variable names can be any length

Declaration of Variables

A variable must be declared before it is used in the program. The general form of declaration of a variable is

Type variable1, variable2, …., variable

Variables are separated by commas. A declaration statement must end with a semicolon. Some valid declarations are

```
int count;
float x,y
```

Giving values to variables

A variable must be given a value after it has been declared but before it is used in an expression.this can be achieved in two ways.

1. By using an Assignment statement
2. By using a read statement

1. Assignment statement

A simple method of giving value to a variable is through the assignment statement as follows

variableName = value

For Example

initialValue = 1;
finalValue = 100;

2. Read Statement

We may also give values to variables interactively through the keyboard using the readLine().

Scope of the Variable

- The scope refers to the validity across the java program.
- The scope of a variable is limited to the block defined within the braces { and }
- It means a variable cannot be accessed outside the scope (or) The scope or a particular variable is the range within a program's source code in which that variable is recognized by the compiler.
- When scope rules are violated, errors will be generated during the compilation step.

Example:

```
class Scope
{
        Public static void main(String args[])
        {
                int x;
                x=10;
                if(x==10)
                {               // start new scope
                int y=20;       //know only to this block
                System.out.println("x and y" +x+" "+y);
                x=y+2;
                }
                // y=100// error ! y not known here
                // x is still known here
                System.out.println("x is  "+x);
          }
    }
```

- If we remove the comment symbol on the line y=100, a compile time error will occur because y is not visible outside of its block.

13. Type Conversion and Casting

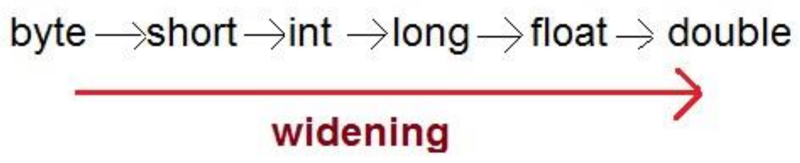
Assigning a value of one type to a variable of another type is known as **Type Casting**.

Example :

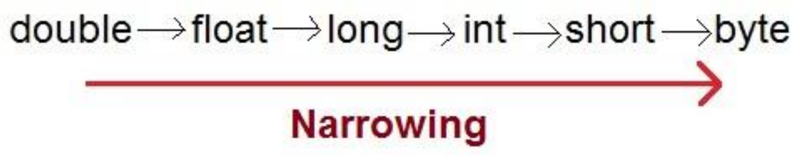
```
int x = 10;
byte y = (byte)x;
```

In Java, type casting is classified into two types,

- Widening Casting(Implicit) : Process of Converting lower data type into higher Data type



- Narrowing Casting(Explicitly done) : Process of converting Higher Data type into Lower Data Type



Widening or Automatic type converion

Automatic Type casting take place when,

- the two types are compatible
- the target type is larger than the source type

Example :

```
public class Test
{
   public static void main(String[] args)
   {
     int i = 100;
     long l = i;      //no explicit type casting required
     float f = l;     //no explicit type casting required
     System.out.println("Int value "+i);
     System.out.println("Long value "+l);
     System.out.println("Float value "+f);
   }

}
```

```
Int value 100
Long value 100
Float value 100.0
```

Narrowing or Explicit type conversion

When you are assigning a larger type value to a variable of smaller type, then you need to perform explicit type casting.

Example :

```
public class Test
{
   public static void main(String[] args)
   {
     double d = 100.04;
     long l = (long)d;  //explicit type casting required
     int i = (int)l; //explicit type casting required

     System.out.println("Double value "+d);
     System.out.println("Long value "+l);
     System.out.println("Int value "+i);
   }

}
```

```
Double value 100.04
Long value 100
Int value 100
```

14. Automatic Type Promotion Expression

Java permits mixing of constants and variables of different types in an expression, but during evaluation it adheres to very strict rules of type conversion. We know that the computer, considers one operator at a time, involving two operands. If the operands are of different types, the 'lower' type is automatically converted to the 'higher' type before the operation proceeds .the result is of the higher type.

If byte, short and int variables are used in an a expression, the result is always promoted to int, to avoid overflow. If a single long is used in the expression , the whole expression is promoted to long. If a expression contains a float operand, the entire expression is promoted to float. If any operand id double, result is double.

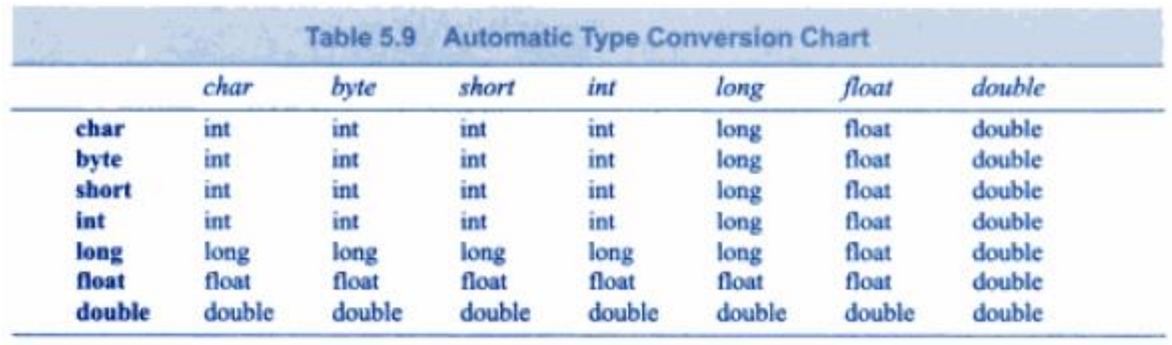


Table 5.9 Automatic Type Conversion Chart

	char	*byte*	*short*	*int*	*long*	*float*	*double*
char	int	int	int	int	long	float	double
byte	int	int	int	int	long	float	double
short	int	int	int	int	long	float	double
int	int	int	int	int	long	float	double
long	long	long	long	long	long	float	double
float	float	float	float	float	float	float	double
double	double	double	double	double	double	double	double

The final result of an expression is converted to the type of the variable on the left of the assignment sign before assigning the value to it. However, the following changes are introduced during the final assignment.

- float to int causes truncation of the fractional part
- double to float causes rounding of digits
- long to int causes dropping the excess higher order bits.

15. -Arrays

An Array is a group of continuous or related data items that share a common name. For instance, we can define an array name salary to represent a set of salaries of a group of employees. A particular value is indicated by writing a number called ***index*** number or ***subscript*** in brackets after the array name.

For Example ,

salary[10]

represents the salary of the 10th employee. While the complete set of values is referred to as an array, the individual values are called ***elements.***

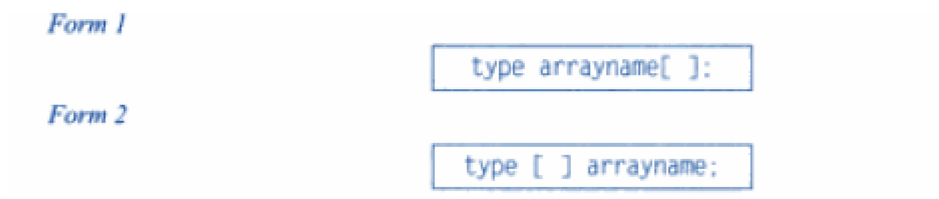
The ability to use a single name to represent a collection of items and to refer to an item by specifying the item number enables us to develop ***concise and efficient programs.***

One – Dimensional Arrays

A list of items can be given one variable name using only one subscript and such a variable is called a ***single-subscripted*** variable or a ***one-dimensional array.***

Declaring Array Variables

Arrays in java may be declared in two foms



Note − The style **type[] arrayname** is preferred. The style **type arrayname[]** comes from the C/C++ language and was adopted in Java to accommodate C/C++ programmers.

Example

The following code snippets are examples of this syntax −

```
double[] myList;   // preferred way.
or
double myList[];   // works but not preferred way.
```

Creating Arrays

You can create an array by using the new operator with the following syntax −

Syntax

```
arrayname = new type[arraySize];
```

The above statement does two things −

- It creates an array using new type[arraySize].
- It assigns the reference of the newly created array to the variable arrayname.

Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement, as shown below −

```
type[] arrayname = new type[arraySize];
```

Alternatively you can create arrays as follows −

```
type[] arrayname = {value0, value1, ..., valuek};
```

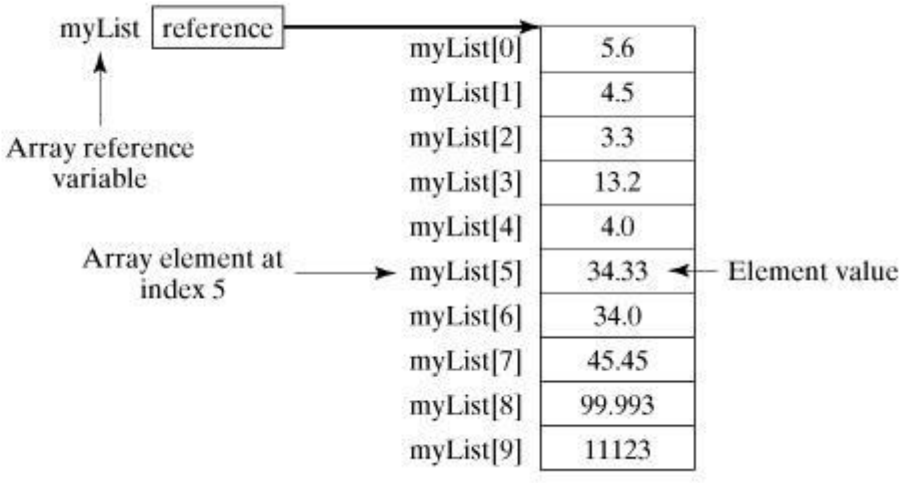
The array elements are accessed through the **index**. Array indices are 0-based; that is, they start from 0 to **arrayname.length-1**.

Example

Following statement declares an array variable, myList, creates an array of 10 elements of double type and assigns its reference to myList −

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

Following picture represents array myList. Here, myList holds ten double values and the indices are from 0 to 9.



Two-Dimensional Array

Two-dimensional Array is used to store two dimensional data. Two dimensional array's are used to store table of data, which contains rows and columns. . The Two Dimensional Array in Java programming language is nothing but an Array of Arrays. If the data is linear we can use the One Dimensional Array but to work with multi-level data we have to use Multi-Dimensional Array.

Creating an Two-Dimensional array

Data_Type[][] Array_Name = new int[Row_Size][Column_Size];

Initialization of Two Dimensional Array in Java

We can initialize the Two Dimensional Array in multiple ways

First Approach

Declaring and Creating an Array

int[][] Student_Marks = new int[2][3];

Initializing Array elements in more traditional way

```
        Student_Marks[0][0] = 15; // Initializing Array elements at position [0][0]
        Student_Marks[1][1] = 45; // Initializing Array elements at position [1][1]
```

second Approach

int[][] Employees = { {10, 20, 30}, {15, 25, 35}, {22, 44, 66}, {33, 55, 77} };

Here, We did not mention the row size and column size but the compiler is intelligent enough to calculate the size by checking the number of elements inside the row and column

third Approach

Above 3 ways are good to store small number of elements into the array, What if we want to store 100 rows or 50 column values. It will be a nightmare to add all of them using any of the above mentioned approaches. To resolve this, we can use the Nested For Loop in Java concept here:

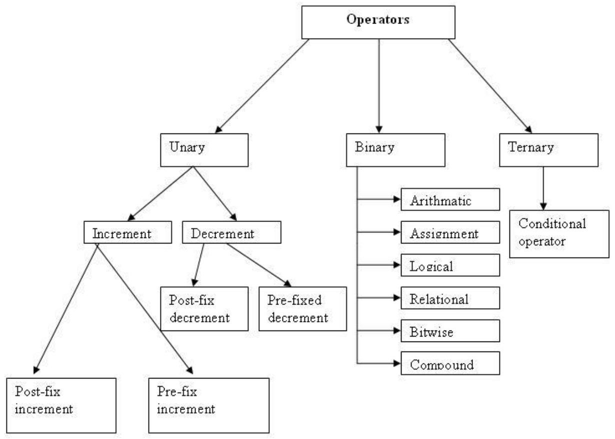
```
int rows, columns;
int[][] Employees = new int[100][50];

for (rows = 0; rows < 100 ; rows++) {
        for (columns = 0; columns < 50; columns++) {
                Employees[rows][columns] = rows + columns;
        }
}
```

16. Operators and Expressions

In Java Operators are symbols that are used to perform some operations on the operands.They are used to manipulate primitive Data types.Combination of operands and operators are known as expression.Java provides a rich set of operators to manipulate the variables.There are three types of operators in Java.

1. Unary operators
2. Binary operators
3. Ternary operators



1. **UNARY OPERATOR:**

In which we use one operand is called unary operator.It has two types.

1.1 Increment Unary operator

1.2 Decrement Unary operator

1.1 Increment Unary operator:

This is used to increase the value by one. It has two types.

- Post-fix Increment operator
- Pre-fix Increment operator

- ***Post-fix Increment operator:***

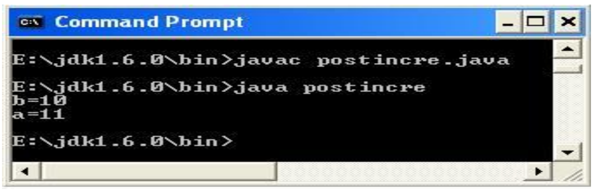
"++" symbol is used to represent Post-fix Increment operator. This symbol is used after the operand.

In this operator , value is first assign to a variable and then incremented the value.

Example

```
class postincre
{
public static void main(String aa[])
{
int a,b;
a=10;
b=a++;
System.out.println("b="+b);
System.out.println("a="+a);
}
}
```

output:



```
E:\jdk1.6.0\bin>javac postincre.java

E:\jdk1.6.0\bin>java postincre
b=10
a=11

E:\jdk1.6.0\bin>
```

In above example first the value of **"a"** is assign to the variable **"b"** .Then increment the value.So the value of b variable is **"10"**.

- ***Pre-fix Increment operator:***

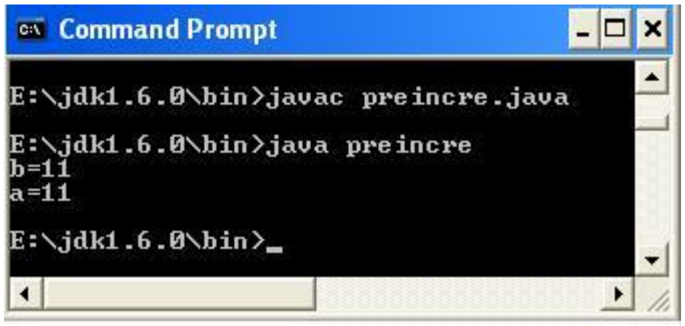
"++" symbol is used to represent Pre-Fix operator.This symbol is used after the operand.

In this operator value is incremented first and then issigned to a variable.

Example

```
class preincre
{
public static void main(String aa[])
{
int a,b;
a=10;
b=++a;
System.out.println("b="+b);
System.out.println("a="+a);
}
}
```

output:



```
E:\jdk1.6.0\bin>javac preincre.java

E:\jdk1.6.0\bin>java preincre
b=11
a=11

E:\jdk1.6.0\bin>_
```

In above example first the increment is done then the value of **"a"** variable is assigned to the variable **"b"**.So the value of **"b"** variable is **11.**

a. Decrement Unary operator

This is used to decrease the value by one. It has two types.

- Post-fix decrement operator
- Pre-fix decrement operator

- **Post-fix decrement operator:**

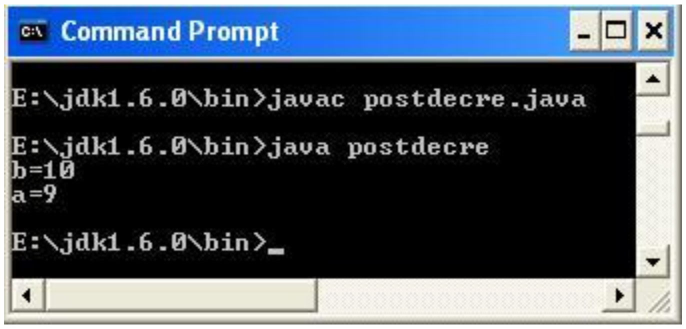
"–" symbol is used to represent post-fix decrement opearator.This symbol is used after the operand.

In this operator , value is first assigned to a variable and then decremented the value.

Example

```
class postdecre
{
public static void main(String aa[])
{
int a,b;
a=10;
b=a–;
System.out.println("b="+b);
System.out.println("a="+a);
}
}
```

output:



```
E:\jdk1.6.0\bin>javac postdecre.java

E:\jdk1.6.0\bin>java postdecre
b=10
a=9

E:\jdk1.6.0\bin>_
```

In above example first the value of **"a"** is assign to the variable **"b"** .Then decrement the value.So the value of b variable is **"10"**.

- ***Pre-fix decrement operator:***

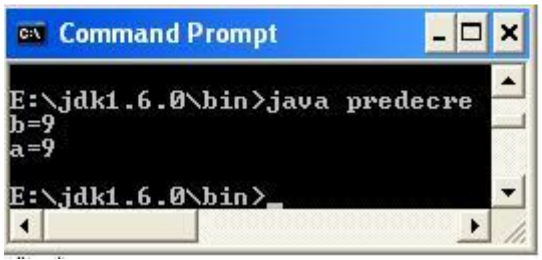
"–" symbol is used to represent the pre-fix decrement operator.This symbol is used after the operand.

In this operator, value is decremented first and then decremented value is used in expression.

Example

```
class predecre
{
public static void main(String aa[])
{
int a,b;
a=10;
b=–a;
System.out.println("b="+b);
System.out.println("a="+a);
}
}
```

output:



```
E:\jdk1.6.0\bin>java predecre
b=9
a=9

E:\jdk1.6.0\bin>_
```

In above example first the value of **"a"** is decrement then assign to the variable **"b"**.So the value of b variable is **"9"**.

2.<u>Binary operators :</u>

In which we use two operand is called Binary operator. Java supports many types ofBinary

Operator.
1. Assignment Operator
2. Arithmetic Operator
3. Logical Operator
4. Comparison Operator

1. Assignment Operator :

This Operator is used to assign the value. This symbol "=" is used to assign the value . e.g **int a=12;**

2. Arithmetic Operator :

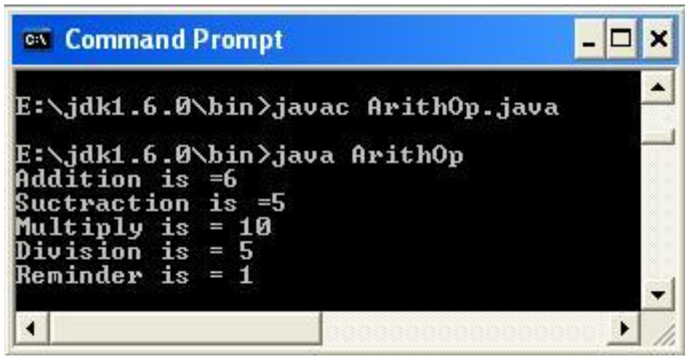
This Operator is used to perform mathematical operation on operand .Arithmetic operator are

	Operators	Description	Use
1.	Additional operator ("+") :	Used to add the value of two operand.	a+b
2.	Subtract operator ("-") :	Used to subtract the value of two operand.	a-b
3.	Multiply operator ("*") :	Used to multiply the value of two operand.	a*b
4.	Division operator ("/") :	Used to divide the value of two operand.	a/b
5.	Modulus operator("%") :	Used returns the remainder of a division operation.	a%b

Example :

```
class ArithOp
{
public static void main (String[] args)
{
  // answer is now 6
int answer = 4 + 2;
System.out.println("Addition is =" +answer);
      // answer is now 5
answer = answer – 1;
System.out.println("Subtraction is =" +answer);
  // answer is now 10
answer = answer * 2;
System.out.println("Multiply is = " +answer);
// answer is now 5
answer = answer / 2;
System.out.println("Division is = " +answer);
  // answer is now 1
answer = answer % 2;
System.out.println("Reminder is = " +answer);
}
}
```

output :



```
E:\jdk1.6.0\bin>javac ArithOp.java

E:\jdk1.6.0\bin>java ArithOp
Addition is =6
Suctraction is =5
Multiply is = 10
Division is = 5
Reminder is = 1
```

Logical operators:

The logical operators || (conditional-OR) , && (conditional-AND) and ! (conditional-NOT)operates on boolean expressions. Here's how they work.

Java Logical Operators

Operator	Description	Example
||	conditional-OR; true if either of the boolean expression is true	false || true is evaluated to true
&&	conditional-AND; true if all boolean expressions are true	false && true is evaluated to false
!	Conditional-NOT; true if exp is false	!false is evaluated to true

Example 8: Logical Operators

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

          int number1 = 1, number2 = 2, number3 = 9;
          boolean result;
```

```
        // At least one expression needs to be true for result to be true
        result = (number1 > number2) || (number3 > number1);
        // result will be true because (number1 > number2) is true
        System.out.println(result);

        // All expression must be true from result to be true
        result = (number1 > number2) && (number3 > number1);
        // result will be false because     (number3 > number1) is false
        System.out.println(result);
    }
}
```

When you run the program, the output will be:

```
true
false
```

3. Logical NOT Operator :

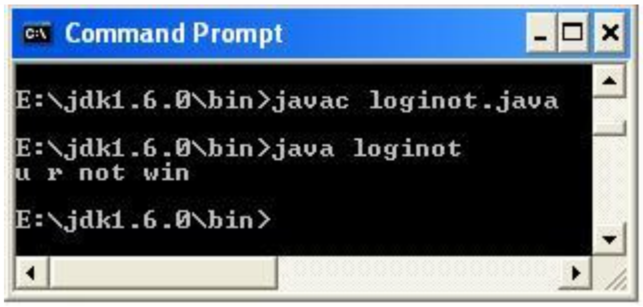
Logical NOT operator is used to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.If a condition is false then Logical NOT operator will make True.

The not operator is probably the easiest to understand. It is simply the opposite of what the condition says.

Example

```
class loginot
{
public static void main(String aa[])
{
boolean a=true;
if (!a)
System.out.println("u r win");
else
System.out.println("u r not win");
}
}
```

output:



```
E:\jdk1.6.0\bin>javac loginot.java

E:\jdk1.6.0\bin>java loginot
u r not win

E:\jdk1.6.0\bin>
```

In above example "if not true" is asking if the variable "a" variable is not true, otherwise known as false. If "a" variable is false, Java will display "u r win". "a"variable is set to true, so that code will not execute. then the else part is execute shown in output.

Relational operator:

This operator is used to compare the two values ,so this operator is also known as"comparison operator."

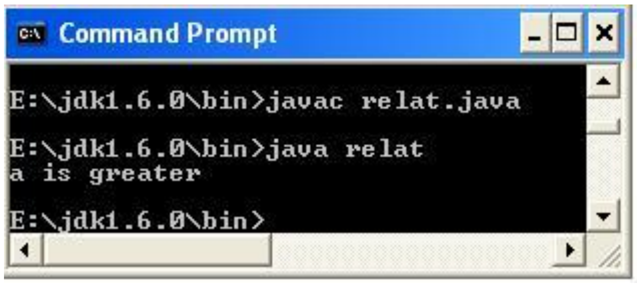
Conditional symbols and their meanings for comparison operator are below.

Operator	**Condition**	**Description**	**Example**
==	is equal to	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(a == b) is not true.
!=	is not equal to	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
>	is greater than	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
<	is less than	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.

Example:

```
class relat
{
public static void main(String aa[])
{
int a=10;
int b=5;
if (a>b)        // “>” relational operator
system.out.println(“a is greater”);
else if (a==b)    //”==” relational operator
System.out.println(“a is equal to b”);
else
System.out.println(“enter 1 to 10 number”);
}
}
```

output:



```
E:\jdk1.6.0\bin>javac relat.java

E:\jdk1.6.0\bin>java relat
a is greater

E:\jdk1.6.0\bin>
```

Ternary operator:

In Ternary operator use three operands. It is also called **conditional assignment statement** because the value assigned to a variable depends upon a logical expression.

syntax is :

variable=(test expression) ? Expression 1 : Expression 2

Example:

c=(a>b)?a:b;

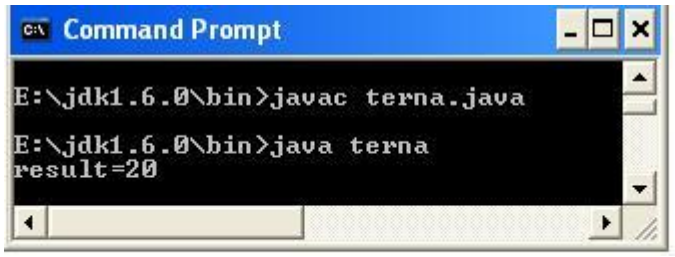
c= (a>b) ? a : b ;

Test Condition Expression1 Expression2

Example of Ternary operator:

```
class terna
{
public static void main(String aa[])
{
int a,b,result;
a=10;
b=20;
result=(a>b)?a:b;
System.out.println(“result=”+c);
}
}
```

output:



```
E:\jdk1.6.0\bin>javac terna.java

E:\jdk1.6.0\bin>java terna
result=20
```

Bitwise Operators

Java provides 4 bitwise and 3 bit shift operators to perform bit operations.

- | Bitwise OR
- & Bitwise AND
- ~ Bitwise Complement
- ^ Bitwise XOR
- << Left Shift
- >> Right Shift
- >>> Unsigned Right Shift

Bitwise and bit shift operators are used on integral types (byte, short, int and long) to perform bit-level operations.

Java Bitwise and Bit Shift Operators

Operator	Description
|	Bitwise OR
&	Bitwise AND
~	Bitwise Complement
^	Bitwise XOR

`<<`	Left Shift
`>>`	Right Shift
`>>>`	Unsigned Right Shift

Bitwise OR

Bitwise OR is a binary operator (operates on two operands). It's denoted by `|`.
The `|` operator compares corresponding bits of two operands. If either of the bits is 1, it gives 1. If not, it gives 0. For example,

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)
Bitwise OR Operation of 12 and 25
  00001100
| 00011001
  ________
  00011101  = 29 (In decimal)
```

Example 1: Bitwise OR

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

            int number1 = 12, number2 = 25, result;

            result = number1 | number2;
            System.out.println(result);
    }
}
```

When you run the program, the output will be:

```
29
```

Bitwise AND

Bitwise AND is a binary operator (operates on two operands). It's denoted by `&`.
The `&` operator compares corresponding bits of two operands. If both bits are 1, it gives 1. If either of the bits is not 1, it gives 0. For example,

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bit Operation of 12 and 25
  00001100
& 00011001
  ________
  00001000  = 8 (In decimal)
```

Example 2: Bitwise AND

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

              int number1 = 12, number2 = 25, result;

              result = number1 & number2;
              System.out.println(result);
    }
}
```

When you run the program, the output will be:

```
8
```

Bitwise Complement

Bitwise complement is an unary operator (works on only one operand). It is denoted by ~.
The ~ operator inverts the bit pattern. It makes every 0 to 1, and every 1 to 0.

```
35 = 00100011 (In Binary)

Bitwise complement Operation of 35
~ 00100011
  ________
  11011100  = 220 (In decimal)
```

Example 3: Bitwise Complement

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

            int number = 35, result;

            result = ~number;
            System.out.println(result);
    }
}
```

When you run the program, the output will be:

```
-36
```

Why are we getting output -36 instead of 220?
It's because the compiler is showing 2's complement of that number; negative notation of the binary number.
For any integer n, 2's complement of n will be -(n+1).

```
 Decimal         Binary                      2's complement
---------       ---------          ---------------------------------------
0            00000000         -(11111111+1) = -00000000 = -0(decimal)
1            00000001         -(11111110+1) = -11111111 = -256(decimal)
12            00001100         -(11110011+1) = -11110100 = -244(decimal)
220           11011100         -(00100011+1) = -00100100 = -36(decimal)

Note: Overflow is ignored while computing 2's complement.
```

The bitwise complement of 35 is 220 (in decimal). The 2's complement of 220 is -36. Hence, the output is -36 instead of 220.

Bitwise XOR
Bitwise XOR is a binary operator (operates on two operands). It's denoted by ^.
The ^ operator compares corresponding bits of two operands. If corresponding bits are different, it gives 1. If corresponding bits are same, it gives 0. For example,

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bitwise XOR Operation of 12 and 25
  00001100
| 00011001
```

```
 ________
 00010101  = 21 (In decimal)
```

Example 4: Bitwise XOR

```
class Xor {
```

The bitwise complement of 35 is 220 (in decimal). The 2's complement of 220 is -36. Hence, the output is -36 instead of 220.

Bitwise XOR

Bitwise XOR is a binary operator (operates on two operands). It's denoted by `^`.
The `^` operator compares corresponding bits of two operands. If corresponding bits are different, it gives 1. If corresponding bits are same, it gives 0. For example,

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bitwise XOR Operation of 12 and 25
 00001100
| 00011001
 ________
 00010101  = 21 (In decimal)
```

Example 4: Bitwise XOR

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

            int number1 = 12, number2 = 25, result;

            result = number1 ^ number2;
            System.out.println(result);
   }
}
```

When you run the program, the output will be:

```
21
```

Signed Left Shift

The left shift operator `<<` shifts a bit pattern to the left by certain number of specified bits, and zero bits are shifted into the low-order positions.

```
212 (In binary: 11010100)

212 << 1 evaluates to 424 (In binary: 110101000)
212 << 0 evaluates to 212 (In binary: 11010100)
212 << 4 evaluates to 3392 (In binary: 110101000000)
```

Example 5: Signed Left Shift

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

           int number = 212, result;

           System.out.println(number << 1);
           System.out.println(number << 0);
           System.out.println(number << 4);
   }
}
```

When you run the program, the output will be:

```
424
212
3392
```

Signed Right Shift

The right shift operator `>>` shifts a bit pattern to the right by certain number of specified bits.

```
212 (In binary: 11010100)

212 >> 1 evaluates to 106 (In binary: 01101010)
212 >> 0 evaluates to 212 (In binary: 11010100)
212 >> 8 evaluates to 0 (In binary: 00000000)
```

If the number is a 2's complement signed number, the sign bit is shifted into the high-order positions.

Example 6: Signed Right Shift

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

           int number = 212, result;

           System.out.println(number >> 1);
           System.out.println(number >> 0);
           System.out.println(number >> 8);
   }
}
```

When you run the program, the output will be:

```
public class Test {

  public static void main(String args[]) {
    int a = 60;    /* 60 = 0011 1100 */
    int b = 13;    /* 13 = 0000 1101 */
    int c = 0;

    c = a & b;       /* 12 = 0000 1100 */
    System.out.println("a & b = " + c );

    c = a | b;       /* 61 = 0011 1101 */
    System.out.println("a | b = " + c );

    c = a ^ b;       /* 49 = 0011 0001 */
    System.out.println("a ^ b = " + c );

    c = ~a;          /*-61 = 1100 0011 */
    System.out.println("~a = " + c );

    c = a << 2;      /* 240 = 1111 0000 */
    System.out.println("a << 2 = " + c );

    c = a >> 2;      /* 15 = 1111 */
    System.out.println("a >> 2  = " + c );

    c = a >>> 2;     /* 15 = 0000 1111 */
    System.out.println("a >>> 2 = " + c );
  }
}
```

This will produce the following result −

Output

```
a & b = 12
a | b = 61
a ^ b = 49
~a = -61
a << 2 = 240
a >> 2  = 15
a >>> 2 = 15
```

17. Control Statements

Causes the flow of execution to advance and branch based on changes to the state of a program.

In Java, control statements can be divided into the following three categories:

- Selection Statements
- Iteration Statements
- Jump Statements

Selection Statements

Selection statements allow you to control the flow of program execution on the basis of the outcome of an expression or state of a variable known during runtime.

Selection statements can be divided into the following categories:

- The if and if-else statements
- The if-else statements
- The if-else-if statements
- The switch statements

The if statements

The first contained statement (that can be a block) of an if statement only executes when the specified condition is true. If the condition is false and there is not else keyword then the first contained statement will be skipped and execution continues with the rest of the program. The condition is an expression that returns a boolean value. General form of simple if statement is

```
if<expression>
{
Statement-block;
}
```

- The statement-block may be single statement or a group of statements .
- If the expression is true, the statement block will be executed, otherwise the statement block will be skipped to the statement-x.
- Example: Print even numbers from 1 to 10

```
class Even
{
        public static void main(string args[])
        {
            for(int i=1;i<=10;i++)
            {
                If(i%2 ==0)
                System.out.println(i+"is a even number");
            }
        }
    }
```

Output:

```
o    is even number
4    is even number
6    is even number
8    is even number
10   is even number
```

<u>if else statement:-</u>

- if else statement is an extension of the simple if statement. The general form is

```
if(expression)
{
```

```
            True-block statements
        }
        Else
        {
            False-block statements
        }
```

- if the test expression is true, then the true-block statements immediately following the if statement are executed. Otherwise, the false-block statements are executed .
- In either case, Either true-block or false-block will be executed, not both.
- In both the cases, the control is transferred subsequently to the statement-x Diagram

```
Class IfesleTest
{
        Public static void main(Stringargs[])
        {
                Int number[]={50,65,71,81};
                Int even=0,odd=0;
                For(int i=0;i<number.length;i++)
                {
                        If((number[i]%2)==0)
                        even+=1;
                        else
                        odd+=1;
                }
                s.o.p("even nos:"+even+"odd nos:"+odd);
        }
}
o/p:
even nos:2 odd nos:3.
```

<u>ested if else statement:-</u>

- A nested if is an if statement that is the target of another if or else.
- Nested ifs are very common in programming
- General form of Nested if looks like
- Nested if else statement is made by placing one if else in another if else statement.
- Nested if else statement helps to select one out of many chooses.
- General form of Nested if else is

```
if<cond1>
```

```
{
        if<cond2>
        {
                if<cond3>
                        stmt 4
                else
                        stmt3
        }
        else
                stmt2
}
else
        stmt 1
```

- In the nested if else statement, the outermost if is evaluated first.
  If the condition1 is false, the statement is the outermost else is evaluated and if else ends.
  If the conditon1 is true, the control goes to execute the next inner if statement.
- If conditon2 is false, statement2 is executed otherwise conditon3 is evaluated
- If condition3 is false statement3 is executed. Otherwise statement is executed.

```
Class Nested ifelse
{
        public static void main(String args[])
        {
                Int a=325,b=715,c=478;
                s.o.p("largest value is:");
                if(a>b)
                {
                        If(a>c)
                        s.o.p(a);
        else
                        s.o.p(c);
                }
                Else
                {
                        If(c>b)
                        s.o.p(c);
```

```
                    else
                    s.o.p(b);
                }
            }
        }
        Output:-
        Largest value is:712
```

se if ladder:-

- A common programming construct that is based a sequence of nested is based upon a sequence of nested ifs is the if else if ladder.
- General form of if else ladder

```
        if<condition>
            stmt
        else if<condition>
            stmt;
        else if<condition>
            stmt;
        else
            stmt;
```

- The if statements are Executed from the top down. As soon as one of the conditions controlling the if is true, the stmt associated with that if is Executed, and the rest of the ladder is bypassed.
- If none of the condition is true, then the final else stmt will be executed.
- The final else acts as a default condition; i.e if all other conditional tests fail, then the last else stmt is performed.
- If there is no final else and all other condition are false.

switch statement:-

- The switch statement helps to select one out of many chooses.
- It often provides a better alternative than a large d=series of if else if statements
- General form of switch statement is

```
        Switch(expression)
        {
            Case value 1:stmt1;
                        Break;
            Case value 2:stmt2;
                        Break;
            .
```

```
                        .
                        .
                        Case value N: stamt N;
                                     Break;
                        Default:stmt;
                  }
```

- The expression must be of type byte,short,int or char.
- Each of the values specified in the case stmts must be of a type compatible with the expression.
- Each case value must be unique literal.
- Duplicate case value are not allowed.
- The switch stmt works like this
  - The value of the expression is compared with each of the literal values in the case stmt.
  - It a match is found, the code sequence following that case stmt is executed.
  - If none of the constants matches the value of the expression, then the default stmt is executed.
  - However, the default stmt is optional, if no case matches and no default is present, then no further action is taken.
  - The break stmt is used inside the switch to terminate a stmt sequence.
  - When a break stmt is encountered, execution branches to the first case of code that follows the entire switch stmt.
- A switch stmt is usually more efficient than a set of nested ifs faster than the equivalent logic coded using a sequence of if else.

<u>ample of switch compared to if else:-</u>

```
ass Ifelse

public static void main(String args[])
{
        Int month=4;
                String season;
        If(month==12||month==1||month==2)
                Season=”winter”;
        Else if(mont==3||month==4||month==5)
                Season=”spring”;
        Else if(month==6||month==7||month==8)
                Season=”summer”;
        Else if(month=9||month==10||month=11)
                Season=”autumn”;
        Else
```

```
                Season="Bogus month";
                s.o.p("April is in the"+season+"");
}

):-
pril is in the spring
```

ıme program using switch stmt

```
ass switch

Public static void main(String args[])
{
        Int mont=4;
        String season;
        Switch(month)
        {
                Case12:
                Case1:
                Case2:
                Season="winter";
                Break;
                Case3:
                Case4:
                Case5:
                Season="spring";
                Break;
                Case6:
                Case7:
                Case8:
                Season="summer";
                Break;
                Case9:
                Case10:
                Case11:
                Season="autumn";
                Break;
                Default:season="Bogus month";
        }
s.o.p("April is in the season"+season);
}
```

conditional operator stmt (or) Ternary operator:-

- java includes a special ternary (three-way) operator that can replace certain types of if-then-else stmts.
- The conditional or ternary operation has general form as
  Expression1?expression2:expression3
- Here, expression1 can be any expression that to a Boolean value.
- If expression is true, then expression2 is evaluated; otherwise, expression3 is evaluated.

```
For example:
If(x<0)
        Flag=0;
Else
        Flag=1;
Can be written as
        Flag=(x<0)?0:1
```

ITERATION STMTS:-

Java's iteration stmts are
1. for
2. while
3. do-while.

These stmts creates what we commonly call loops

A loop repeatedly executes the same set of instruction until a termination condition is met

hile:-
The while loop is java's most fundamental loop stmt
It repeats a stmt or block while its controlling expression is true.
The general form of while stmt is

```
While <condition>
{
        Body of the loop
}
```

The condition can be any Boolean expression.

The body of the loop will be executed as long as the conditional expression is true
When condition becomes false, control passes to the next line of code immediately following the loop.
The curly braces are unnecessary if only a single stmt is being repeated.

demonstrate the while loop

```
Class while
{
        Public static void main(String args[])
        {
                Int n=10;
                While(n>0)
                {
                        s.o.p("tick"+n);
                        n--;
                }
        }
}
o/p:-
tick 10
tick 9
tick 8
tick 7
tick 6
tick 5
tick 4
tick 3
tick 2
tick 1
```

- Since the while loop evaluates its conditional expression at the top of the loop, the body of the loop will not execute even once if the condition is false to begin with.
- The body of the while can be empty. This is because a Null stmt(one that consists only of a semicolon)is syntactically valid in java.

For ex:-

```
Class nobody
{
        Public static void main(    )
 {
```

```
            Int I=100,j=200;
            While(++i<--j);//no body in the loop.
            s.o.p("mid point is"+i);
    }
    }
    o/p:-
    mid point is 150
```

- The value of I is incremented and j value is decremented
- These value compared with one another
- If the new value of is still less than the new value of j, then loop repeats
- If I is equals to or greater than j the loop stops

Do-while:-

- If the conditional expression controlling a while loop is initially false, then the body of the loop will not executed at all
- However, it is desirable to execute the of a loop at least once , even if condition expression is false to begin with
- Fortunately, java supplies a loop that does just that : the do while

The do while loop always execute its body at least once, because its conditional expression is at bottom of loop

- The general form of do while is

```
do
{
        Body of the loop
}
While<condition>
```

- **Example:**

```
Class dowhile
{
    public static void main( String args[])
    {
            int n=10;
            do
            {
                    s.o.p("tick"+n);
                    n--;
            }while(n>0);
```

o/p:-

```
tick 10
tick 9
tick 8
tick  7
tick 6
tick 5
tick 4
tick 3
tick 2
tick 1
```

- Each iteration of the do while loop first executes the body of loop and than evaluates the conditional expression
  If these expression is true these loop repeats, otherwise the loop terminates
  <u>Condition</u> must be **Boolean expression**
- the do while is useful when you process a menu slection,
- because you will usually want the body of a menu lop to execute atleast once.

For Example:

```
Class menu
{
        Psvm()
        {
                Char choice;
                do{
                        s.o.p("1:is");
                        s.o.p("2:switch");
                        s.o.p("3.do-while");
                s.o.p("4.for");
                }while(choice<'1'||choice>'5');

        Switch(choice)
        {
                Case 1:
                s.o.p("if is branching statement");
                break;
                Case 2:
```

```
                s.o.p("switch is also a branching statement");
                break;
                Case 3:
                s.o.p("do-while is a looping statement");
                break;
                Case 4:
                s.o.p("for is also a looping a statement");
                break;
            }
          }
     }
```

For statement

- in java i.e jdks,there are two forms of the for loop.
- The first is the traditional form the has been is use since the original version of java.
- The second is the new "for-each "form.
- General form of traditional for statement is

```
for(initialization:condition:iteration)
{
        Body of the loop
}
```

- The for loop operates as follows
- When the loop first starts ,the initialization portion of the loop is executed.
- It is important to understand that initialization expression is only executed once. Next, condition is evaluated.This must be a Boolean expression .i.e the loop control variable against a target value.
- If this expression is true, then the body of the loop is executed.
- If it is false, the loop terminates.
- Next, the iteration portion of the loop executed
- This is usually an expression that increments or decreaments the loop controls variable.
- This loop then **ITERATES**
- First evaluatin g the conditional expression,then executing the body of the loop,and then executing the iteration expression with each pass.
- This process repeats until the controlling expression is false.

```
//demonstrate the loop
Class forloop
```

```
{
                Psvm(string args[])
                {
                        Int n;
                        For(n=10;n>0;n--)
                        s.o.p("tick"+m);
            }

}
```

Initialization exp is evaluated only once,then condiotino is evaluated if it is true body of the loop ecutes and comes to iteration portion if it is false out executing body of the loop goes to iteration rtion.

;:

```
Class fortick

        Psvm()
        {
                For(int n=10;n>10;n--)
                s.o.p("tick"+n);
        }
```

Nested loop:-

Like all other programming languages, java allows loops to be nested.

i.e one loop may be inside another

eg:-

```
//loops may be nested
Class Nested
{
        P s v m( )
        {
                Int I, j;
                For(i=0;i<10;i++)
                {
                        For(j=I;j<10;j++)
                                s.o.p(". ");
                                s.o.p();
        }
}

 . . . . . . . .
. . .  . . . .
. . . . . .
. . . . .
. . . .
. . .
. .
.
```

2. <u>Jmp stmts:</u>

- Java supports 3 jump stmts

1. break
2. continue
3. return.

<u>break stmt:-</u>

it has 3 uses.

1. It terminates a stmnt sequence in a switch stmt.
2. If can be used to exit a loop.
3. If can be used as a “civilized” form of goto.

- When a break stmt is encountered inside a loop. The loop is terminated and program control resumes at the next stmt following the loop.
- i.e by using break, we can force immediate termination of a loop, by passing the conditional expression (eg:i<=10) and any Remaining code in the vody of the loop.

Eg:-

```
 class BreakLopp
{
        P s v m( )
        {
                For(int i=0;i<100;i++)
                {
                        If(i==0)break;//terminate loop it 1=10
                        s.o.p(“i=”+i);
                }
                s.o.p(“loop complete”);
```

```
  }
}
o/p:-
i=0
i=1
i=2
i=3
i=4
i=5
i=6
i=7
i=8
i=9
loop complete.
```

- Break stmnt can be used with any of java's loops (i.e while, do..while and for loops)
- When used inside a set of nested loops, the break stmt will only break out of the innermost loop.

Eg:

```
Class breakloop
{
  P s v m( )
  {
          For(int i=0;i<3;i++)
          {
                  s.o.p("pass"+i+":");
                  for(int i=0;j<100;j++)
                  {
                          If(j==10)break;
                          s.o.p(j+" ");
                  }
                  s.o.p("loops complete");
          }
  }
}
o/p:-
pass 0:0 1 2 3 4 5 6 7 8 9
pass 1:0 1 2 3 4 5 6 7 8 9
pass 2:0 1 2 3 4 5 6 7 8 9
loops complete
```

continue:-

- sometimes, you might want to continue running the loop but stop continue running the remainder of the code in its body for this particular iteration
  the continue stmt performs such as an action.

  Eg:-

```
Class continue.
{
 p s v m( )
 {
          For(int i=0;i<10;i++)
          {
                  s.o.p("i+" ");
                  if(i%2==0)continue;
                  s.o.p("")'
          }
 }
}
```

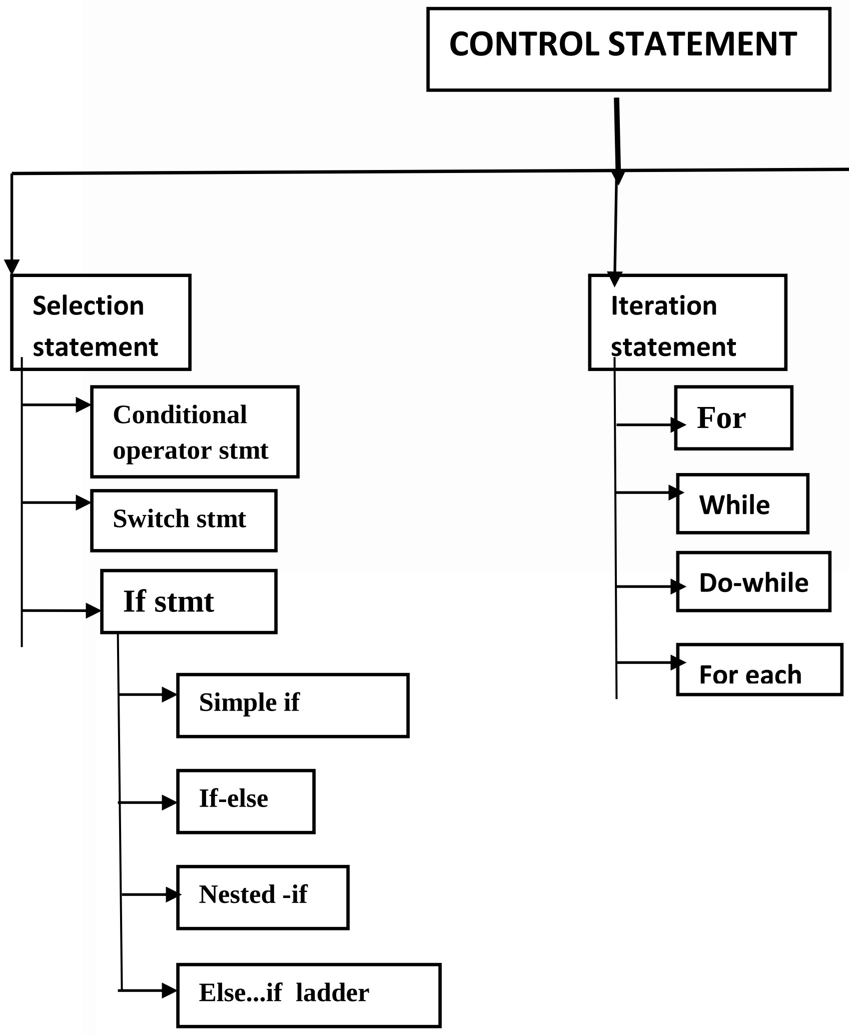
  Here if I is even, the loop continue without printing a newline.

  o/p:-

```
0 1
2 3
4 5
6 7
8 9
```

3.2 **Return:-**

- Return stmt is used to explicitly return from a method
- i.e it causes program control to transfer back to the caller of the method
- return stmt can be used to cause execution to branch back to the caller at the method.


CONTROL STATEMENT
Selection statement
Conditional operator stmt
Switch stmt
If stmt
Simple if
If-else
Nested -if
Else...if ladder
Iteration statement
For
While
Do-while
For each
Jump statements
break
continue
return