UNIT I

INTRODUCTION TO OOP AND JAVA FUNDAMENTALS

OBJECT-ORIENTED PROGRAMMING

Object-Oriented Programming (OOP) is a programming language model organized around objects rather than actions and data. An object-oriented program can be characterized as data controlling access to code.

Concepts of OOPS

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

OBJECT

- Object means a **real word entity** such as pen, chair, table etc.
- Any entity that has **state and behavior** is known as an object.
- Object can be defined as an **instance of a class.**
- An object contains an address and takes up some space in memory.
- **Identity:** Object identity is typically implemented via a unique ID.
- The value of the ID is not visible to the external user. But, it is used internally by the JVM to identify each object uniquely.

CLASS

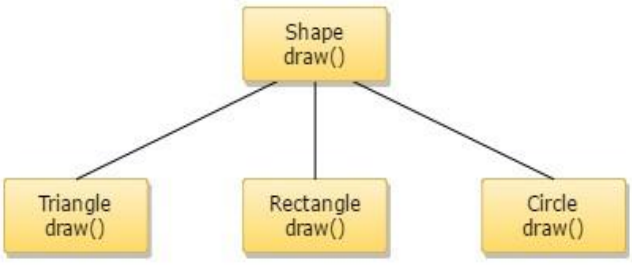
- **Collection of objects** is called class. It is a logical entity.
- A class can also be defined as a blueprint from which you can create an individual object.
- A class consists of Data members and methods.
- The primary purpose of a class is to hold data/information.
- The member functions determine the behavior of the class, i.e. provide a definition for supporting various operations on data held in the form of an object. Class doesn't store any space.

INHERITANCE

Inheritance can be defined as the procedure or mechanism of acquiring all the properties and behavior of one class to another, i.e., acquiring the properties and behavior of child class from the parent class.

POLYMORPHISM

When one task is performed by different ways i.e. known as polymorphism. For example: to convince the customer differently, to draw something e.g. shape or rectangle etc.



Polymorphism is classified into two ways:

Method Overloading (Compile time Polymorphism)

Method Overloading is a feature that allows a class to have two or more methods having the same name but the arguments passed to the methods are different.

Compile time polymorphism refers to a process in which a call to an overloaded method is resolved at compile time rather than at run time.

Method Overriding (Run time Polymorphism)

If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in java. In other words, If subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as method overriding.

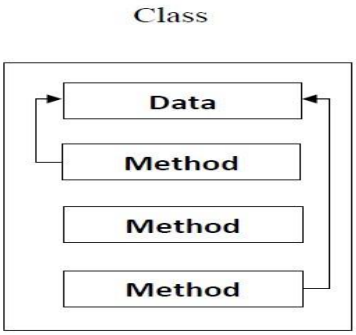
ABSTRACTION

Abstraction is a process of hiding the implementation details and showing only functionality to the user. For example: phone call, we don't know the internal processing. In java, we use abstract class and interface to achieve abstraction.

ENCAPSULATION

Encapsulation in java is a process of wrapping code and data together into a single unit, for example capsule i.e. mixed of several medicines.

A java class is the example of encapsulation.



DIFFERENCE BETWEEN PROCEDURE-ORIENTED AND OBJECT-ORIENTED PROGRAMMING

Procedure-Oriented Programming	Object-Oriented Programming
In POP, program is divided into small parts called **functions**	In OOP, program is divided into parts called **objects**.
In POP,Importance is not given to **data** but to	In OOP, Importance is given to the data rather
functions as well as **sequence** of actions to be done.	than procedures or functions because it works as a **real world**.
POP follows **Top Down approach**.	OOP follows **Bottom Up approach**.
POP does not have any access specifier.	OOP has access specifiers named Public, Private, Protected, etc.
In POP, Data can move freely from function to function in the system.	In OOP, objects can move and communicate with each other through member functions.
To add new data and function in POP is not so easy.	OOP provides an easy way to add new data and function.
In POP, Most function uses Global data for sharing that can be accessed freely from function to function in the system.	In OOP, data can not move easily from function to function,it can be kept public or private so we can control the access of data.
POP does not have any proper way for hiding data so it is **less secure**.	OOP provides Data Hiding so provides **more security**.
In POP, Overloading is not possible.	In OOP, overloading is possible in the form of Function Overloading and Operator Overloading.

Example of POP are : C, VB, FORTRAN, Pascal.	Example of OOP are : C++, JAVA, VB.NET, C#.NET.

FEATURES OF JAVA

- The main objective of Java programming language creation was to make it portable, simple and secure programming language.
- Apart from this, there are also some awesome features which play important role in the popularity of this language.
- The features of Java are also known as java *buzzwords*. A list of most important features of Java language are given below.

Simple

Java is very easy to learn and its syntax is simple, clean and easy to understand. According to Sun, Java language is a simple programming language because:

- ✓ Java syntax is based on C++ (so easier for programmers to learn it after C++).
- ✓ Java has removed many confusing and rarely-used features e.g. explicit pointers, operator overloading etc.
- ✓ There is no need to remove unreferenced objects because there is Automatic Garbage Collection in java.

Object-oriented

- ✓ Java is object-oriented programming language.
- ✓ Everything in Java is an object.

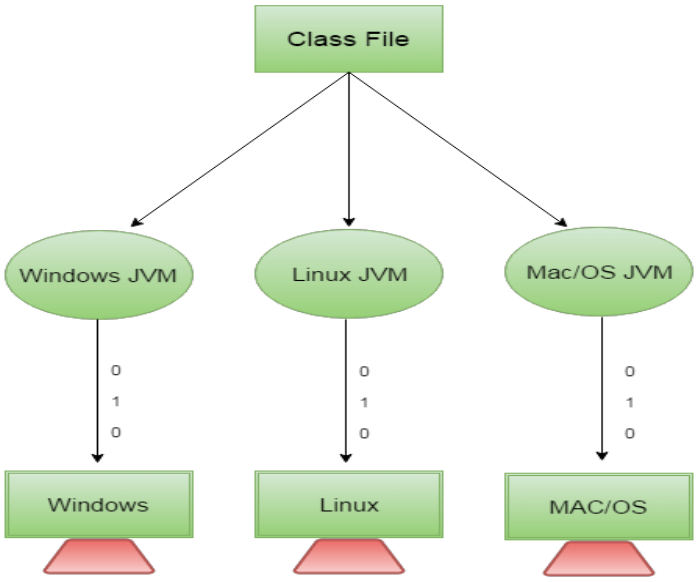
- ✓ Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and Behavior.
- ✓ Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

  Basic concepts of OOPs are:

  1. Object
  2. Class
  3. Inheritance
  4. Polymorphism
  5. Abstraction
  6. Encapsulation

Platform Independent

- ✓ Java is platform independent because it is different from other languages like C, C++ etc.
- ✓ which are compiled into platform specific machines while Java is a write once, run anywhere language.
- ✓ A platform is the hardware or software environment in which a program runs.
- ✓ There are two types of platforms software-based and hardware-based.
- ✓ Java provides software-based platform.



The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms.

It has two components:

1. Runtime Environment
2. API(Application Programming Interface)

Java code can be run on multiple platforms

e.g. Windows, Linux, Sun Solaris, Mac/OS etc.

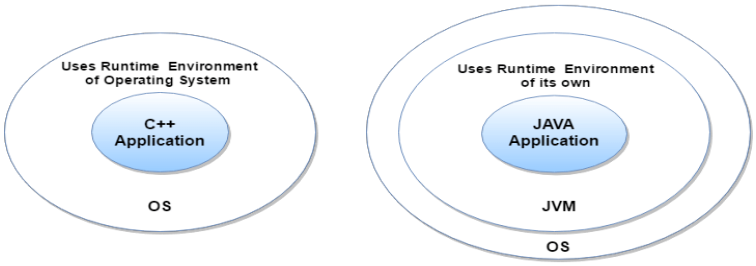
- ✓ Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere(WORA).

Secured

Java is best known for its security. With Java, we can develop virus-free systems.

Java is secured because:

- No explicit pointer
- Java Programs run inside virtual machine sandbox



Class loader:

- ✓ Class loader in Java is a part of the Java Runtime Environment (JRE) which is used to dynamically load Java classes into the Java Virtual Machine. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.

Bytecode Verifier:

- ✓ It checks the code fragments for illegal code that can violate access right to objects. **Security Manager**: It determines what resources a class can access such as reading and writing to the local disk.

- ✓ These security are provided by java language. Some security can also be provided by application developer through SSL, JAAS, and Cryptography etc.

Robust

- ✓ Robust simply means strong. Java is robust because:
- ✓ It uses strong memory management.
- ✓ There are lack of pointers that avoids security problems.

- ✓ There is automatic garbage collection in java which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.

- ✓ There is exception handling and type checking mechanism in java. All these points makes java robust.

Architecture-neutral

- ✓ Java is architecture neutral because there is no implementation dependent features e.g. size of primitive types is fixed.

- ✓ In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. But in java, it occupies 4 bytes of memory for both 32 and 64 bit architectures.

Portable

- ✓ Java is portable because it facilitates you to carry the java bytecode to any platform. It doesn't require any type of implementation.

High-performance

- ✓ Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower

than a compiled language (e.g. C++). Java is an interpreted language that is why it is slower than compiled languages e.g. C, C++ etc.

Distributed

- ✓ Java is distributed because it facilitates users to create distributed applications in java. RMI and EJB are used for creating distributed applications.
- ✓ This feature of Java makes us able to access files by calling the methods from any machine on the internet.

Multi-threaded

- ✓ A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications etc.

Dynamic

- ✓ Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand.
- ✓ It also supports functions from its native languages i.e. C and C++.
- ✓ Java supports dynamic compilation and automatic memory management (garbage collection).

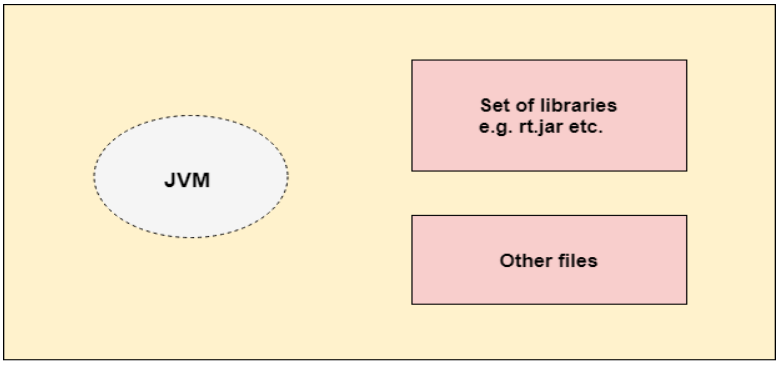
GARBAGE COLLECTION

- ✓ Objects are dynamically allocated by using the **new** operator, dynamically allocated objects must be manually released by use of a **delete** operator. Java takes a different approach; it handles deallocation automatically this is called garbage collection.

THE JAVA ENVIRONMENT

JRE

- ✓ JRE is an acronym for Java Runtime Environment.
- ✓ It is also written as Java RTE.
- ✓ The Java Runtime Environment is a set of software tools which are used for developing java applications.
- ✓ It is used to provide runtime environment. It is the implementation of JVM. It physically exists. It contains set of libraries + other files that JVM uses at runtime.
- ✓ Implementation of JVMs are also actively released by other companies besides Sun Micro Systems.



JRE

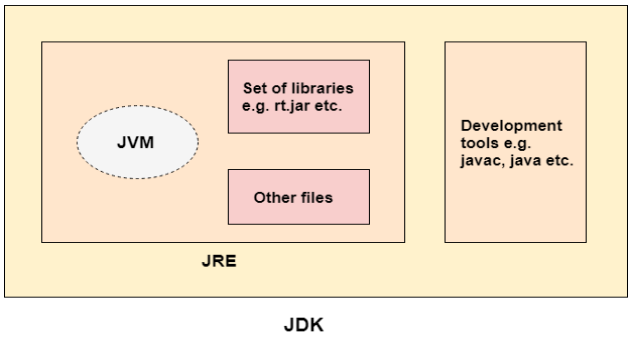
JDK

- ✓ JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop java applications and applets. It physically exists.
- ✓ It **contains JRE + development tools.**

- ✓ JDK is an implementation of any one of the below given Java Platforms released by Oracle corporation:

1. **Standard Edition Java Platform**
2. **Enterprise Edition Java Platform**
3. **Micro Edition Java Platform**

The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc) etc. to complete the development of a Java Application.



JVM (Java Virtual Machine)

JVM (Java Virtual Machine) is an abstract machine.

- ✓ It is a specification that provides runtime environment in which java bytecode can be executed.
- ✓ JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).
- ✓ **The JVM performs following operation:**
- ✓ Loads code,
- ✓ verifies code,
- ✓ executes code,
- ✓ provides runtime environments
- ✓

STRUCTURE OF JAVA PROGRAM

A first Simple Java Program

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

To compile:

javac Simple.java

To execute:

java Simple

class keyword is used to declare a class in java.

public keyword is an access modifier which represents visibility, it means it is visible to all.

static is a keyword, if we declare any method as static, it is known as static method.

- ✓ The core **advantage of static method** is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create object to invoke the main method. So it saves memory.

void is the return type of the method, it means it doesn't return any value.

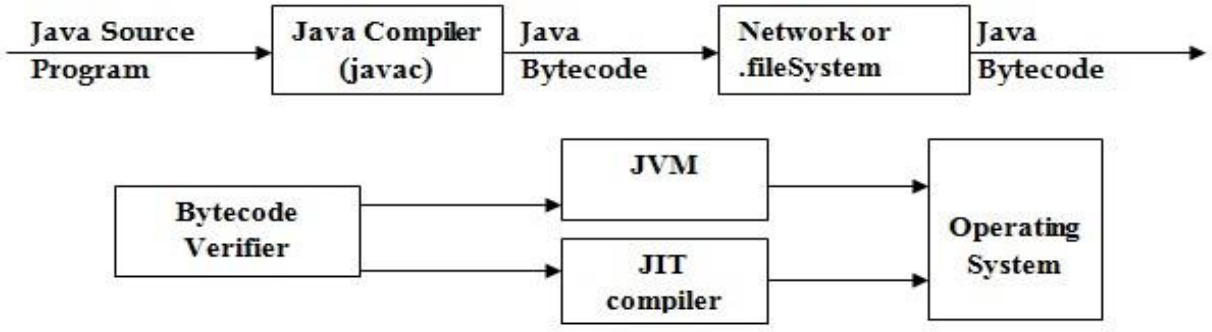
main represents the starting point of the program.

String[] args is used for command line argument.

System.out.println() is used print statement.

- ❖ A program is written in JAVA, the javac compiles it. The result of the JAVA compiler is the .class file or the bytecode and not the machine native code (unlike C compiler).
- ❖ The bytecode generated is a non-executable code and needs an interpreter to execute on a machine.
- ❖ This interpreter is the JVM and thus the Bytecode is executed by the JVM.

And finally program runs to give the desired output.



DEFINING CLASSES IN JAVA

- The class is at the core of Java .
- A class is a *template* for an object, and an object is an *instance* of a class. A class is declared by use of the **class** keyword

Syntax:

class *classname*

{

type instance-variable1; type instance-variable2;

// ...

type instance-variableN;

type methodname1(*parameter-ist*)

{

// body of method

}

...

type methodnameN(parameter-list)

{

// body of method

}

- The data, or variables, defined within a **class** are called *instance variables*. The code is contained within *methods*.

- The methods and variables defined within a class are called *members* of the class. In most classes, the instance variables are acted upon and accessed by the methods defined for that class.

Example

A Simple Class class called **Box** that defines three instance variables: **width**, **height**, and **depth**.

```
class Box
{
double width;
 double height;
double depth;
}
```

To create a Box object

```
Box mybox = new Box(); // create a Box object called mybox
```

mybox will be an instance of **Box**.

Example1:

```
/* A program that uses the Box class. Call this file BoxDemo.java */

class Box
{
double width;
```

```
 double height;
 double depth;
}
// This class declares an object of type Box.
class BoxDemo
{
public static void main(String args[])
{
Box mybox = new Box();
double vol;    // assign values to mybox's instance variables
mybox.width = 10;
mybox.height = 20;
mybox.depth = 15; // compute volume of box
vol = mybox.width * mybox.height * mybox.depth;
System.out.println("Volume is " + vol);
}
}
```

Output:

Volume is 3000.0

Declaring Objects

- First, declare a variable of the class type.
- This variable does not define an object. Instead, it is simply a variable that can *refer* to an object.

Syntax:

```
Box mybox = new Box();
Box mybox; // declare reference to object
mybox = new Box(); // allocate a Box object
```

Assigning Object Reference Variables Syntax:

```
Box b1 = new Box();
Box b2 = b1;
```

- **b2** is being assigned a reference to a copy of the object referred to by **b1**.
- **b1** and **b2** will both refer to the *same* object.

CONSTRUCTORS

- ✓ Constructors are special member functions whose task is to initialize the objects of its class.
- ✓ It is a special member function, **it has the same as the class name.**
- ✓ Java constructors are invoked when their objects are created.
- ✓ It is named such because, it constructs the value, that is provides data for the object and are used to initialize objects.
  - ✓ Every class has a constructor when we don't explicitly declare a constructor for any java class the compiler creates a default constructor for that class which does not have any return type.
  - ✓ The constructor in Java cannot be abstract, static, final or synchronized and these modifiers are not allowed for the constructor.

There are two types of constructors:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Default constructor (no-arg constructor)

- ✓ A constructor having no parameter is known as default constructor and no-arg constructor.

Example:

```
/* Here, Box uses a constructor to initialize the dimensions of a box.

*/ class Box
 {
double width;
double height;
double depth;

// This is the constructor for Box.
Box()
{
System.out.println("Constructing Box");
width = 10;
height = 10;
depth = 10;
}
// compute and return volume
 double volume()
 {
```

```
return width * height * depth;
}
}
class test
 {
public static void main(String args[])
{
// declare, allocate, and initialize Box objects
Box mybox1 = new Box();
 Box mybox2 = new Box();
double vol;
// get volume of first box
vol = mybox1.volume();
System.out.println("Volume is " + vol);
// get volume of second box
vol = mybox2.volume();
System.out.println("Volume is " + vol);
}
}
```

Output:

```
Constructing Box
Constructing Box
Volume is 1000.0
Volume is 1000.0
```

Parameterized Constructors

- ✓ A constructor which has a specific number of parameters is called parameterized constructor. Parameterized constructor is used to provide different values to the distinct objects.

Example:

```
class Box
 {
double width;
double height;
double depth;
// This is the constructor for Box.

Box(double w, double h, double d)
 {
width = w;
height = h;
depth = d;
}
// compute and return volume
double volume()
{
return width * height * depth;
}
}
class BoxDemo7
{
```

```
public static void main(String args[])
{
// declare, allocate, and initialize Box objects
Box mybox1 = new Box(10, 20, 15);
Box mybox2 = new Box(3, 6, 9);
double vol;
// get volume of first box vol = mybox1.volume();
System.out.println("Volume is " + vol);
// get volume of second box vol = mybox2.volume();
System.out.println("Volume is " + vol);
}
}
```

Output:

Volume is 3000.0

Volume is 162.0

Box mybox1 = new Box(10, 20, 15);

The values 10, 20, and 15 are passed to the **Box( )** constructor when **new** creates the object.

Thus, **mybox1**'s copy of **width**, **height**, and **depth** will contain the values 10, 20, and 15 respectively.

Overloading Constructors

Example:

```
/* Here, Box defines three constructors to initialize the dimensions of a box
various ways.
*/
```

```
class Box
{
double width;
double height;
double depth;

// constructor used when all dimensions specified

Box(double w, double h, double d)
{
 width = w;
 height = h;
 depth = d;
}
// constructor used when no dimensions specified
Box()
{
width = -1; // use -1 to indicate
height = -1; // an uninitialized
depth = -1; // box
}
// constructor used when cube is created
Box(double len)
{
width = height = depth = len;
}
// compute and return volume
```

```
 double volume()
 {
return width * height * depth; }
}
class test
{
public static void main(String args[])
{
// create boxes using the various constructors
Box mybox1 = new Box(10, 20, 15);
Box mybox2 = new Box();
Box mycube = new Box(7);
 double vol;
// get volume of first box
 vol = mybox1.volume();
System.out.println("Volume of mybox1 is " + vol);
// get volume of second box
vol = mybox2.volume();
System.out.println("Volume of mybox2 is " + vol);
// get volume of cube
 vol = mycube.volume();
System.out.println("Volume of mycube is " + vol);
}
}
```

Output:

```
Volume of mybox1 is 3000.0
Volume of mybox2 is -1.0          Volume of mycube is 343.0
```

METHODS

Syntax:

type name(parameter-list)

{

// body of method

}

- ✓ *type* specifies the type of data returned by the method. This can be any valid type, including class types that you create.

The this Keyword

- ✓ **this keyword is used to to refer to the object that invoked it. this can be used inside any method to refer to the *current* object**. That is, this is always a reference to the object on which the method was invoked. this() can be used to invoke current class constructor.

Syntax:

```
Box(double w,
double h, double d)
 {
 this.width = w;
this.height = h;
this.depth = d;
}
```

Example:

```
class Student
{
int id;
  String name;
  student(int id, String name)
 {
   this.id = id;
   this.name = name;

}
void display()
{
      System.out.println(id+" "+name);
 }
 public static void main(String args[])
{
   Student stud1 = new Student(01,"Tarun");
   Student stud2 = new Student(02,"Barun");
   stud1.display();
stud2.display();
  }
}
```

Output:

```
01 Tarun
02 Barun
```

STATIC MEMBERS

Static is a non-access modifier in Java

```
// static variable
static int a = 10;
static int b;
```

Static variables

- ✓ When a variable is declared as static, then a single copy of variable is created and shared among all objects at class level. Static variables are, essentially, global variables. All instances of the class share the same static variable.

Important points for static variables :-

We can create static variables at class-level only.

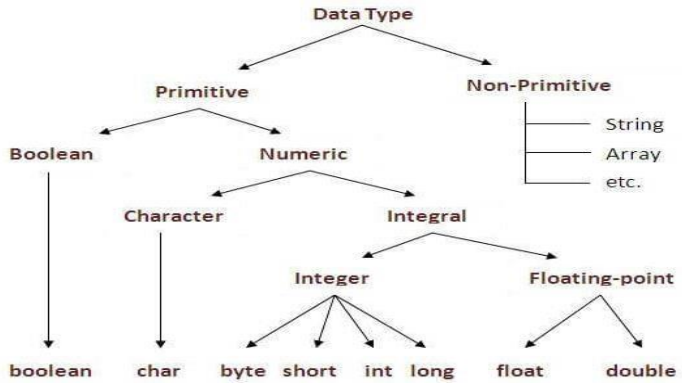
static block and static variables are executed in order they are present in a program.

DATATYPES IN JAVA

Data types specify the different sizes and values that can be stored in the variable.

There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include Integer, Character, Boolean, and Floating Point.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.



Java defines eight primitive types of data:

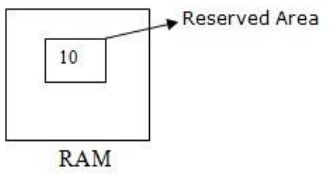
byte, short, int, long, char, float, double, and boolean.

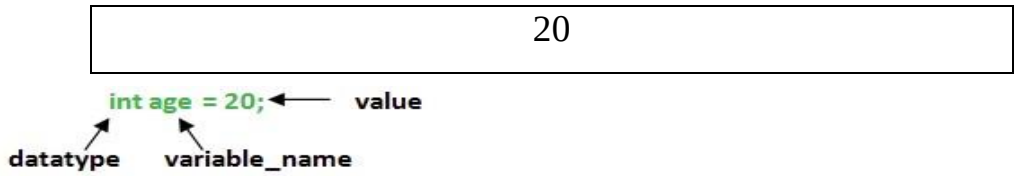
These can be put in four groups:

- Integers This group includes byte, short, int, and long, which are for whole-valued signed numbers.
- Floating-point numbers This group includes float and double, which represent numbers with fractional precision.
- Characters This group includes char, which represents symbols in a character set, like letters and numbers.
- Boolean This group includes Boolean, which is a special type for representing true/false values.

VARIABLES

A variable is a container which holds the value and that can be changed durig the execution of the program. A variable is assigned with a datatype. Variable is a name of memory location. All the variables must be declared before they can be used. There are three types of variables in java: local variable, instance variable and static variable.





1) Local Variable

A variable defined within a block or method or constructor is called local variable.

- These variable are created when the block in entered or the function is called and destroyed after exiting from the block or when the call returns from the function.
- The scope of these variables exists only within the block in which the variable is declared. i.e. we can access these variable only within that block.

2) Instance Variable

Instance variables are non-static variables and are declared in a class outside any method, constructor or block.

- As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed.
- Unlike local variables, we may use access specifiers for instance variables. If we do not specify any access specifier then the default access specifier will be used.

Difference between Instance variable and Static variable

INSTANCE VARIABLE	STATIC VARIABLE
Each object will have its **own copy** of instance variable	We can only have **one copy** of a static variable per class irrespective of how many objects we create.

Changes made in an instance variable using one object will **not be reflected** in other objects as each object has its own copy of instance variable	In case of static changes **will be reflected** in other objects as static variables are common to all object of a class.
We can access instance variables **through object references**	Static Variables can be accessed **directly using class name.**

Class Sample
{
int a;
}
	Class Sample
	{
	static int a;
	}

OPERATORS IN JAVA

Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups –

1. Arithmetic Operators
2. Increment and Decrement
3. Bitwise Operators
4. Relational Operators
5. Boolean Operators
6. Assignment Operator
7. Ternary Operator

Arithmetic Operators

Arithmetic operators are used to manipulate mathematical expressions

Operator Result

Operator	Result
+	Addition (also unary plus)
–	Subtraction (also unary minus)
*	Multiplication
/	Division
%	Modulus
++	Increment
+=	Addition assignment
–=	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
––	Decrement

Example:

// Demonstrate the basic arithmetic operators.

```
 class BasicMath
{
public static void main(String args[])
{
// arithmetic using integers
System.out.println("Integer Arithmetic");
int a = 1 + 1;
int b = a * 3;
int c = b / 4;
int d = c - a;
 int e = -d;
System.out.println("a = " + a);
System.out.println("b = " + b);
```

```
System.out.println("c = " + c);
System.out.println("d = " + d);
System.out.println("e = " + e);
}
}
```

Output:

```
Integer Arithmetic
a = 2
b = 6
c = 1
d = -1
e = 1
```

Modulus Operator

- ✓ The modulus operator, **%**, returns the remainder of a division operation. It can be applied to floating-point types as well as integer types.

Example:

```
// Demonstrate the % operator.
class Modulus
public static void main(String args[])
{
int x = 42;
double y = 42.25;
System.out.println("x mod 10 = " + x % 10);
System.out.println("y mod 10 = " + y % 10);
}
 }
```

Output:

x mod 10 = 2

y mod 10 = 2.25

Arithmetic Compound Assignment Operators

- ✓ Java provides special operators that can be used to combine an arithmetic operation with an assignment.

a = a + 4;

In Java, you can rewrite this statement as shown here: a += 4;

Syntax:

var op= expression;

Example:

// Demonstrate several assignment operators.

```
 class OpEquals
{
public static void main(String args[])
{
 int a = 1;
 int b = 2;
 int c = 3;
 a += 5;
 b *= 4;
 c += a * b;
 c %= 6;
System.out.println("a = " + a);
```

```
System.out.println("b = " + b);
System.out.println("c = " + c);
}
}
```

Output:

```
a = 6
 b = 8
 c = 3
```

Increment and Decrement Operators

The ++ and the – – are Java's increment and decrement operators. The increment operator increases its operand by one. The decrement operator decreases its operand by one.

Example:

```
// Demonstrate ++.
class IncDec
{
 public static void main(String args[])
 {
int a =1;
int b = 2;
int c;
int d;
c = ++b;
d = a++;
c++;
```

```
System.out.println ("a = " + a);
System.out.println ("b = " + b);
System.out.println ("c = " + c);
System.out.println ("d = " + d);
}
}
```

Output:

a = 2

b = 3

c = 4

d = 1

Bitwise Operators

- ✓ Java defines several *bitwise operators* that can be applied to the integer types: **long**, **int**, **short**, **char**, and **byte**. These operators act upon the individual bits of their operands.

Operator	Result
~	Bitwise unary NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND assignment
|=	Bitwise OR assignment
^=	Bitwise exclusive OR assignment
>>=	Shift right assignment
>>>=	Shift right zero fill assignment
<<=	Shift left assignment

Bitwise Logical Operators

- ✓ The bitwise logical operators are **&**, |, ^, and ~.
- ✓ The bitwise operators are applied to each individual bit within each operand.

A	B	A | B	A&B	A^B	~A
0	0	0	0	0	1
1	0	1	0	1	0
0	1	1	0	1	1
1	1	1	1	0	0

Example:

```
// Demonstrate the bitwise logical operators.
class BitLogic
{
public static void main(String args[])
{
String binary[] = {"0000", "0001", "0010", "0011"};
int a = 3;
int b = 6;
int c = a | b;
int d = a & b;
 int e = a ^ b;
System.out.println(" a = " + binary[a]);
System.out.println(" b = " + binary[b]);
System.out.println(" a|b = " + binary[c]);
System.out.println(" a&b = " + binary[d]);
System.out.println(" a^b = " + binary[e]);
```

```
}
 }
```

Output:

```
a = 0011
b = 0110
a|b = 0111
a&b = 0010
a^b = 0101
~a&b|a&~b = 0101
~a = 1100
```

Left Shift Operator

The Java left shift operator << is used to shift all of the bits in a value to the left side of a specified number of times.

Example:

```
class OperatorExample
{
  public static void main(String args[])
{
System.out.println(10<<2);//10*2^2=10*4=40
System.out.println(10<<3);//10*2^3=10*8=80
System.out.println(20<<2);//20*2^2=20*4=80
System.out.println(15<<4);//15*2^4=15*16=240
}
 }
```

Output:

40

80

80

240

Right Shift Operator

The Java right shift operator >> is used to move left operands value to right by the number of bits specified by the right operand.

Example:

```
class OperatorExample
{
  public static void main(String args[])
{
System.out.println(10>>2);//10/2^2=10/4=2
System.out.println(20>>2);//20/2^2=20/4=5
System.out.println(20>>3);//20/2^3=20/8=2
}
}
```

Output:

2

5

2

Relational Operators

The *relational operators* determine the relationship that one operand has to the other.Specifically, they determine equality and ordering.

Operator	Result
&	Logical AND
|	Logical OR
^	Logical XOR (exclusive OR)
||	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT
&=	AND assignment
|=	OR assignment
^=	XOR assignment
==	Equal to
!=	Not equal to
?:	Ternary if-then-else

Example:

```
// Demonstrate the boolean
logical operators. class
BoolLogic
{
public static void main(String args[])
{
boolean a = true; boolean b = false; boolean c = a | b; boolean d= a & b; boolean e
= a ^ b;
System.out.println(" a = " + a);
System.out.println(" b = " + b);
System.out.println(" a|b = " + c);
System.out.println(" a&b = " + d);
System.out.println(" a^b = " + e);
} }
```

Output:

```
a = true
b = false a|b = true a&b = false a^b = true
```

Assignment Operator

The *assignment operator* is the single equal sign, =.

Syntax:

***var = expression*;**

Here, the type of *var* must be compatible with the

type of *expression*.

int x, y, z; x = y = z = 100; // set x, y, and z to 100

This fragment sets the variables **x**, **y**, and **z** to 100 using a single statement.

Ternary Operator

- ✓ Ternary operator in java is used as one liner replacement for if-then-else statement and used a lot in java programming. it is the only conditional operator which takes three operands.

Syntax:

expression1 ? expression2 : expression3

Example:

```
class OperatorExample
{
public static void main(String args[])
{
 int a=2;   int b=5;
int min=(a<b)?a:b;
System.out.println(min);
```

```
}
}
```

Output:

2

CONTROL STATEMENTS

Selection Statements in Java

A programming language uses control statements to control the flow of execution of program based on certain conditions.

Java's Selection statements:

- if
- if-else
- nested-if
- if-else-if
- switch-case
- jump – break, continue, return

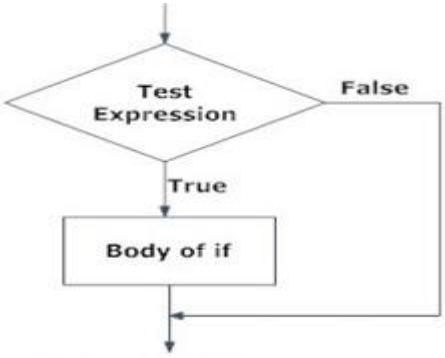
if Statement

- ✓ if statement is the most simple decision making statement. It is used to decide whether a certain statement or block of statements will be executed or not that is if a certain condition is true then a block of statement is executed otherwise not.

Syntax:

```
if(condition)
{
      //statements to execute if
       //condition is true
}
```

- ✓ Condition after evaluation will be either true or false.
- ✓ If the value is true then it will execute the block of statements under it. If there are no curly braces '{' and '}' after **if(condition)** then by default if statement will consider the immediate one statement to be inside its block.



Example:

```
class IfSample
{
public static void main(String args[])
{
 int x, y; x = 10; y = 20;
if(x < y)
System.out.println("x is less than y");
x = x * 2;
 if(x == y)
System.out.println("x now equal to y");
x = x * 2;
 if(x > y)
System.out.println("x now greater than y");
// this won't display anything
if(x == y)
System.out.println("you won't see this");
```

}

}

Output:

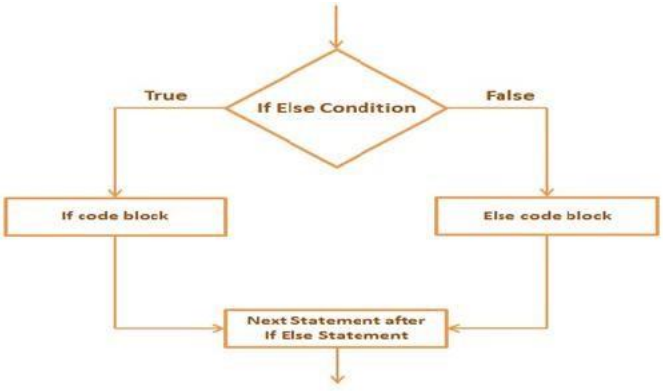
x is less than y x now equal to y x now greater than y

if-else Statement

- ✓ The Java if-else statement also tests the condition. It executes the *if* block if condition is true else if it is false the else block is executed.

Syntax:.

```
If(condition)
{
      //Executes this block if
       //condition is true
}
else
{
//Executes this block if
        //condition is false
}
```



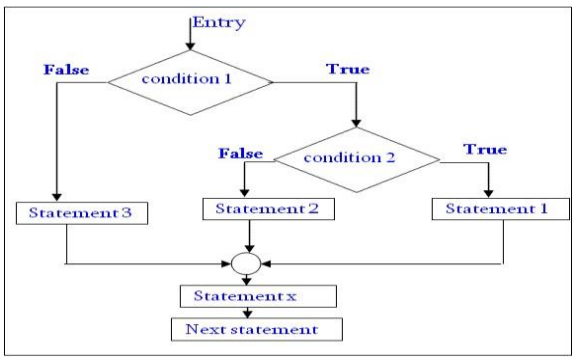
Example:

```
public class IfElseExample
{
public static void main(String[] args)
  {
    int number=13;
if(number%2==0)
{
        System.out.println("even number");
  }
else
  {
        System.out.println("odd number");
  }
 }
 }
```

Output: odd number

Nested if Statement

Nested if-else statements, is that using one if or else if statement inside another if or else if statement(s).



Example:

```
// Java program to illustrate nested-if statement
class NestedIfDemo
{
   public static void main(String args[])
   {
      int i = 10;
        if (i == 10)
      {
         if (i < 15)
            System.out.println("i is smaller than 15");
         if (i < 12)
            System.out.println("i is smaller than 12 too");
         else
            System.out.println("i is greater than 15");
      }
   }
}
```

Output:

i is smaller than 15 i is smaller than 12 too

if-else-if ladder statement

The *if* statements are executed from the top down. The conditions controlling the *if* is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.

Syntax:

```
if(condition) statement;
else if(condition) statement;
else if(condition) statement;
.
.
else statement;
```

Example:

```
public class IfElseIfExample
{
public static void main(String[] args)
 {
int marks=65;
if(marks<50)
{
    System.out.println("fail");
  }
  else if(marks>=50 && marks<60)
{
    System.out.println("D grade");
  }
  else if(marks>=60 && marks<70)
{
    System.out.println("C grade");
  }
  else if(marks>=70 && marks<80)
```

```
{
        System.out.println("B grade");
    }
    else if(marks>=80 && marks<90)
{
System.out.println("A grade");
    }
else if(marks>=90 && marks<100)
{
        System.out.println("A+ grade");
    }
else
{
        System.out.println("Invalid!");
    }
 }
}
```

Output:

C grade

Switch Statements

The **switch** statement is Java's multiway branch statement. It provides an easy way to dispatch execution to different parts of your code based on the value of an expression.

Syntax:

switch (expression) { case value1:

// statement sequence break; case value2:

```
// statement sequence break;
.
. case valueN : // statement sequence break; default:
// default statement sequence
}
```

Example

```
 class SwitchCaseDemo
{
    public static void main(String args[])
    {
        int i = 9;
        switch (i)
        {
        case 0:
            System.out.println("i is zero.");
            break;
        case 1:
            System.out.println("i is one.");
            break;
        case 2:
            System.out.println("i is two.");
```

```
            break;

        default:
            System.out.println("i is greater than 2.");

        }

    }
}
```

Output:

i is greater than 2.

ITERATIVE STATEMENTS

- ✓ In programming languages, loops are used to execute a set of instructions/functions repeatedly when some conditions become true. There are three types of loops in java.

1. while loop
2. do-while loop
3. For loop

while loop

A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.

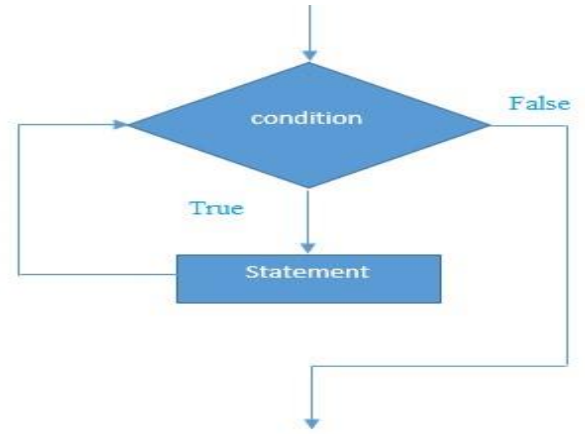
Syntax:

while(*condition*)

{

// body of loop

}



// Demonstrate the while loop.

class While

{

public static void main(String args[])

{

int n = 5;

while(n > 0)

{

System.out.println("tick " + n); n--; }

}

}

Output:

tick 5

tick 4

tick 3

tick 2

tick 1

do-while loop:

do while loop checks for condition after executing the statements, and therefore it is called as Exit Controlled Loop**.**

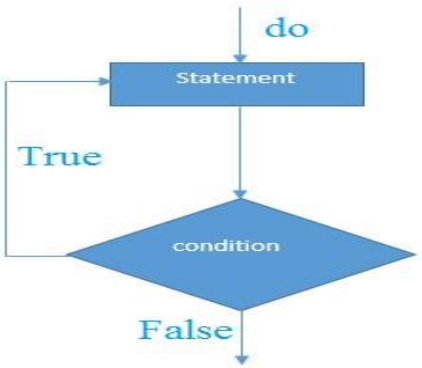
Syntax:

do

{

// body of loop

} while (condition);



public class DoWhileExample

{

public static void main(String[] args)

{

int i=1;

```
  do{
       System.out.println(i);
    i++;
    }while(i<=5);
}
}
```

Output:

1

2

3

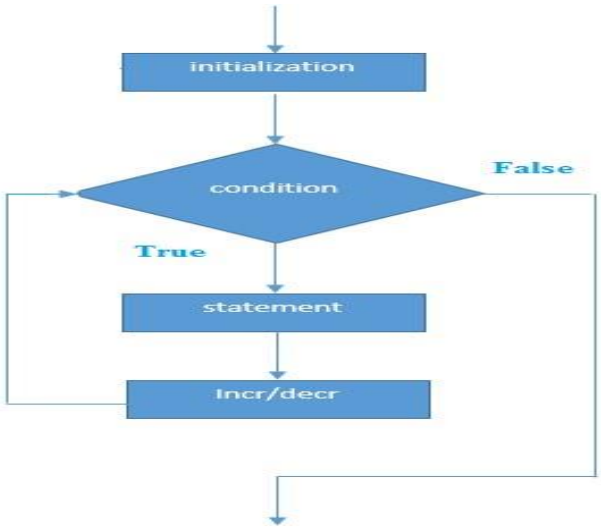
4

5

for loop

- ✓ for loop provides a concise way of writing the loop structure. A for statement consumes the initialization, condition and increment/decrement in one line.
- ✓ **Syntax**

```
for(initialization; condition; iteration)
 {
// body
}
```



Example

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

Output:

1

2

3

4

5

for-each Loop

- ✓ The for-each loop is used to traverse array or collection in java. It is easier to use than simple for loop because we don't need to increment value and use subscript notation. It works on elements basis not index.
- ✓ It returns element one by one in the defined variable.

Syntax:

for(*type itr-var* : *collection*) *statement-block*

Example:

```
// Use a for-each style for loop.
class ForEach
{
public static void main(String args[])
{
int nums[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
 int sum = 0;
// use for-each style for to display and sum the values
for(int x : nums)
{
System.out.println("Value is: " + x);
sum += x;
}
System.out.println("Summation: " + sum);
}
}
```

Output:

Value is: 1

Value is: 2

Value is: 3

Value is: 4

Value is: 5

Value is: 6

Value is: 7

Value is: 8

Value is: 9

Value is: 10

Summation: 55

Packages

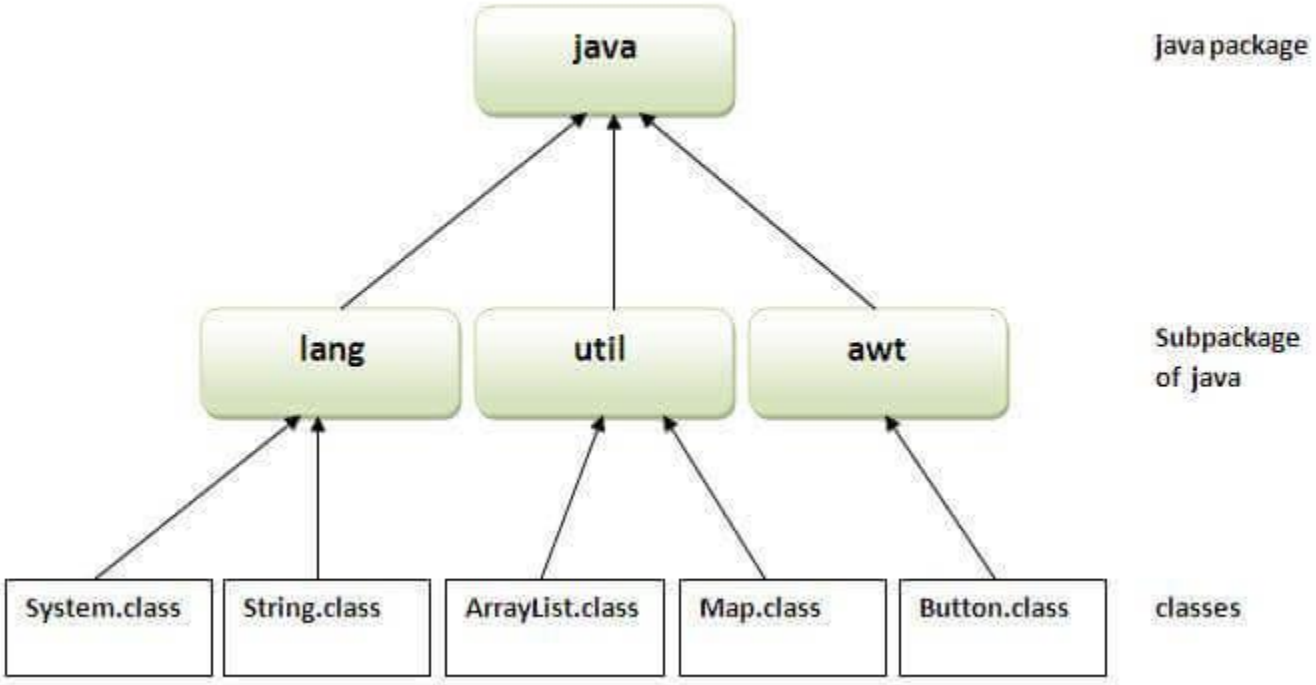
- ✓ A **java package** is a group of similar types of classes, interfaces and sub-packages.
- ✓ Package in java can be categorized in two form, built-in package and user-defined package.
- ✓ There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Advantage of Java Package

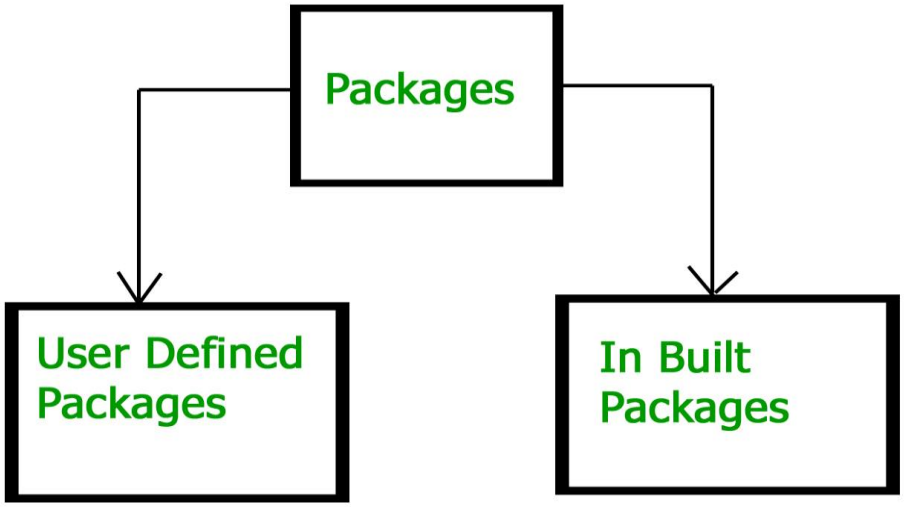
1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.



Types of packages:



Built-in Packages

These packages consist of a large number of classes which are a part of Java **API**.Some of the commonly used built-in packages are:

1) **java.lang:** Contains language support classes(e.g classed which defines primitive data types, math operations). This package is automatically imported.

2) **java.io:** Contains classed for supporting input / output operations.

3) **java.util:** Contains utility classes which implement data structures like Linked List, Dictionary and support for Date / Time operations.

4) **java.applet:** Contains classes for creating Applets.

5) **java.awt:** Contain classes for implementing the components for

graphical user interfaces (like button , menus etc).

6) **java.net:** Contain classes for supporting networking operations.

User-defined packages

These are the packages that are defined by the user.create a directory **myPackage** (name should be same as the name of the package). Then create the **MyClass** inside the directory with the first statement being the **package names**.

Simple example of java package

The **package keyword** is used to create a package in java.

```
1. //save as Simple.java
2. package mypack;
3. public class Simple
4. {
5.  public static void main(String args[])
6. {
7.     System.out.println("Welcome to package");
8.    }
9. }
```

How to compile java package

If you are not using any IDE, you need to follow the **syntax** given below:

1. **javac -d directory javafilename**

   For **example**

1. javac -d . Simple.java

   The -d switch specifies the destination where to put the generated class file.

How to access package from another package?

There are three ways to access the package from outside the package.

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

*1) Using packagename.**

- ✓ If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.
- ✓ The import keyword is used to make the classes and interface of another package accessible to the current package.

Example of package that import the packagename.*

```
1. //save by A.java
2. package pack;
3. public class A
4. {
5.   public void msg(){System.out.println("Hello");
6. }
7. }
1. //save by B.java
2. package mypack;
3. import pack.*;
4.   class B
5. {
6.   public static void main(String args[])
7. {
8.    A obj = new A();
9.    obj.msg();
10. }
11.}
```

```
Output:Hello
```

---

2) Using packagename.classname

- ✓ import package.classname then only declared class of this package will be accessible.

Example of package by import package.classname

```
1. //save by A.java
2.   package pack;
3. public class A
4. {
5.   public void msg()
6. {
7. System.out.println("Hello");
8. }
9. }
1. //save by B.java
2. package mypack;
3. import pack.A;
4.
5. class B
6. {
7.   public static void main(String args[])
8. {
9.    A obj = new A();
10.  obj.msg();
11. }
12.}
```

```
Output:Hello
```

3) Using fully qualified name

If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

Example of package by import fully qualified name

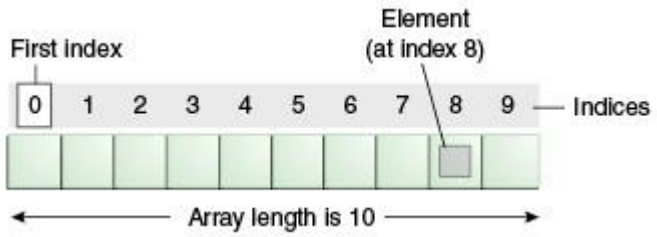
```
1. //save by A.java
2. package pack;
3. public class A
4. {
5.   public void msg()
6. {
7. System.out.println("Hello");}
8. }
1. //save by B.java
2. package mypack;
3. class B
4. {
5.   public static void main(String args[])
6. {
7.    pack.A obj = new pack.A();//using fully qualified name
8.    obj.msg();
9.   }
```

10.}

```
Output:
Hello
```

ARRAYS

- ✓ Array is a collection of similar type of elements that have contiguous memory location.
- ✓ In Java all arrays are dynamically allocated. Since arrays are objects in Java, we can find their length using member length.
- ✓ A Java array variable can also be declared like other variables with [] after the data type.
- ✓ The variables in the array are ordered and each have an index beginning from 0.
- ✓ Java array can be also be used as a static field, a local variable or a method parameter. The **size** of an array must be specified by an int value and not long or short.
- ✓ The direct superclass of an array type is Object.
- ✓ Every array type implements the interfaces Cloneable and java.io.Serializable.



Advantage of Java Array

1. **Code Optimization:** It makes the code optimized, we can retrieve or sort the data easily.

2. **Random access:** We can get any data located at any index position.

Disadvantage of Java Array

1. **Size Limit:** We can store only fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in java.

Types of Array in java

1. One- Dimensional Array
2. Multidimensional Array

One-Dimensional Arrays

- ✓ An array is a group of like-typed variables that are referred to by a common name. An array declaration has two components:
- ✓ the type and the name.

type declares the element type of the array.

- ✓ The element type determines the data type of each element that comprises the array.

Syntax:

type var-name[ ];

Instantiation of an Array in java

array-var = new type [size];

Example:

```
class Testarray
{
public static void main(String args[])
{
 int a[]=new int[5];  //declaration and instantiation
a[0]=10;   //initialization
 a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
//printing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);
}
}
```

Output:

```
10
20
70
40
50
```

Declaration, Instantiation and Initialization of Java Array Example:

```
class Testarray1
{
public static void main(String args[])
{
int a[]={33,3,4,5};//declaration, instantiation and initialization
//printing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);
}
}
```

Output:

33

3

4

5

Multidimensional Arrays

- ✓ Multidimensional arrays are arrays of arrays with each element of the array holding the reference of other array. These are also known as Jagged Arrays. A multidimensional array is created by appending one set of square brackets ([]) per dimension.

Syntax:

type var-name[ ][ ]=new type[row-size][col-size];

Example:

```
// Demonstrate a two-dimensional array.
 class TwoDArray
{
public static void main(String args[])
{
int twoD[][]= new int[4][5];
 int i, j, k = 0;
for(i=0; i<4; i++)
for(j=0; j<5; j++)
{
twoD[i][j] = k;
k++;
}
for(i=0; i<4; i++)
{
for(j=0; j<5; j++)
System.out.print(twoD[i][j] + " ");
System.out.println();
 }
 }
 }
```

Output:

```
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
```