BCA SEM-3

Programming using Java
(05BC3306)

Unit – 1

Introduction to Java and OOP :

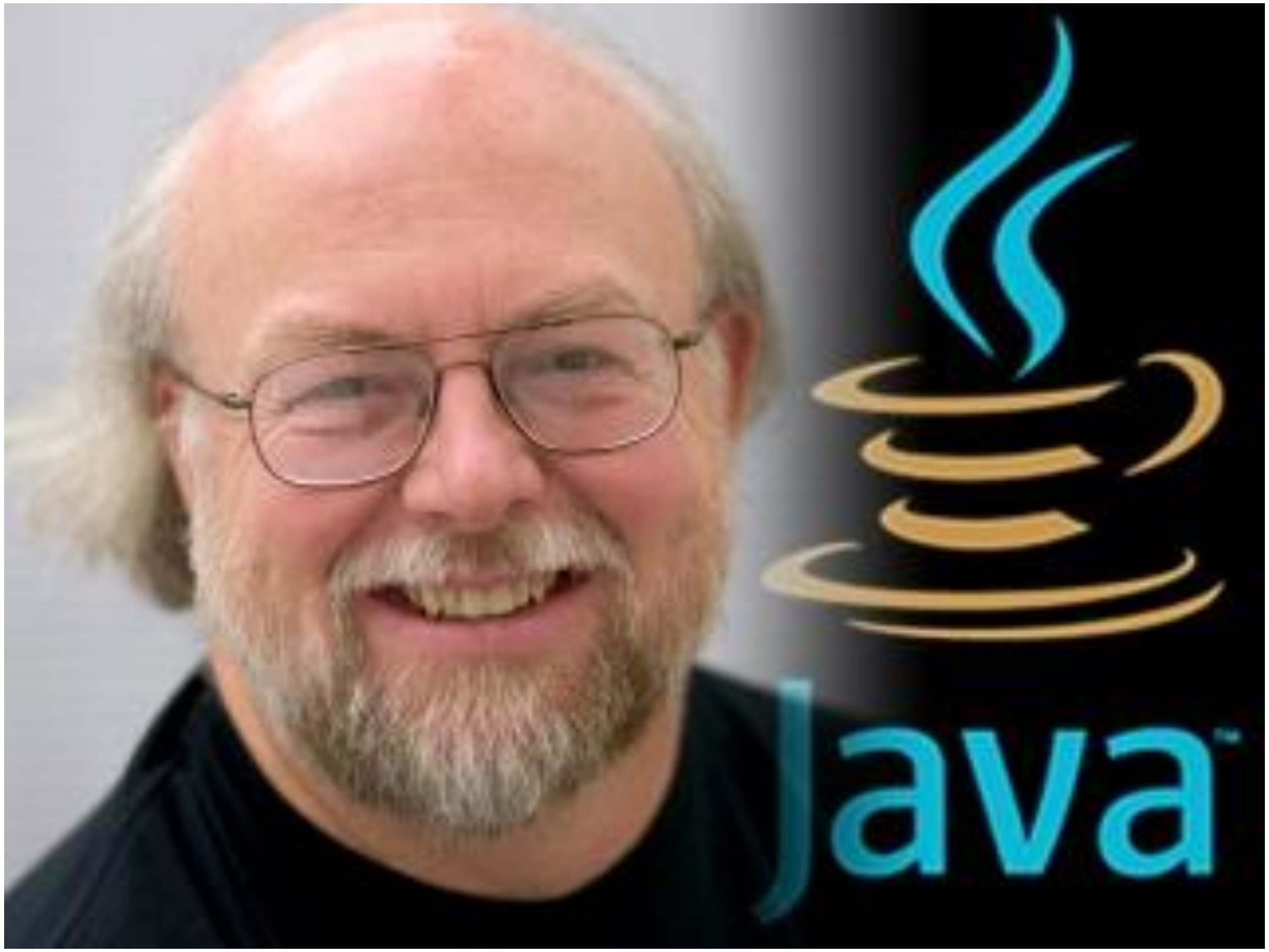
Introduction to Java

Java is a general purpose object oriented Programming language. It is developed by Sun Microsystems of USA.

Its original name was **Oak** given by **James Gosling,** who is one of the inventors of the language.

Java was initially Developed for Softwares for Consumer Electronics Devices like TVs, VCRs, and others.

The father of Java – James Gosling



Introduction to Java

Java is an advance level programming language specially to design client side as well as server side applications. Java is certainly a very good programming language which provides great functionalities. Java is not only a single programming language but it is a whole package which contains huge library, lots of reusable codes and execution environment that provides many services such as security, portability across different operating system and garbage collection.

Introduction to Java

Java supports object oriented concepts so it is a one type of Object oriented programming language. One of the main feature of java language is "**Write Once, Run Anywhere**" which means, you just write code once and it can be reused at anytime, anywhere. In the application developing operation this feature will be very useful as the code which has written once can be reused whenever a developer want to build same type of another module.

History of Java

Year	Development
1990	Sun Microsystems decided to develop a special software for consumer electronics devices. A Team has been formed to undertake this task. James Gosling was the head of that team.
1991	The team announce a new language called “Oak”
1992	The team known as “Green Project” team, have demonstrated the use of language on a list of home appliances.
1993	World Wide Web (WWW) has given support to Green Project Team and they have started thinking for development of Web Applets
1994	A new Web browser called HotJava has been developed by the Team to run applets.
1995	Oak was rename to Java due to some legal problems.
1996	Sun release Java Development Kit 1.0 (JDK 1.0)

History of Java

Year	Development
1997	Sun release JDK 1.1
1998	Sun release Java 2 with JDK 1.2
1999	Sun release J2SE and J2EE
2000	JDK 1.3
2002	JDK 1.4
2004	JDK 1.5
2006	JDK 1.6

Features to Java

Java has various features which makes it simple, secure and compact. They are as follows:

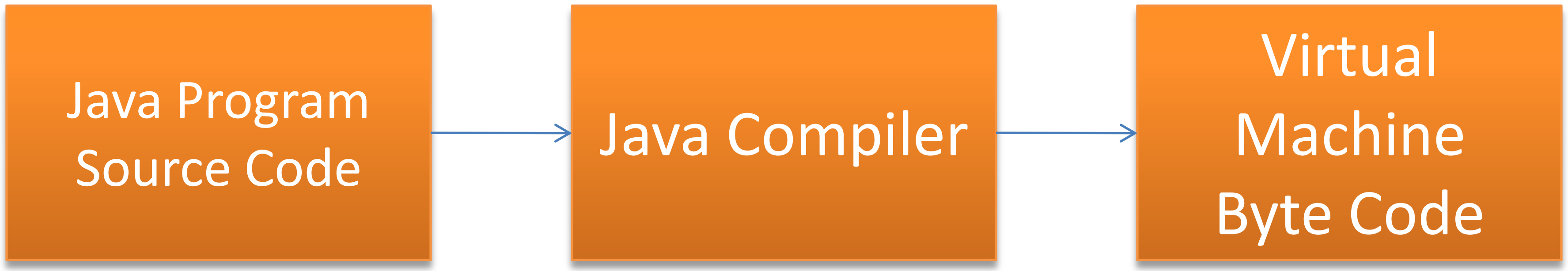
(1) Compiled and Interpreted
(2) Platform Independent
(3) Object Oriented
(4) Secure
(5) Multithreaded
(6) Distributed System
(7) Simple and easy
(8) Robust
(9) Portable
(10) High Performance
(11) Dynamic and extensible

Features to Java

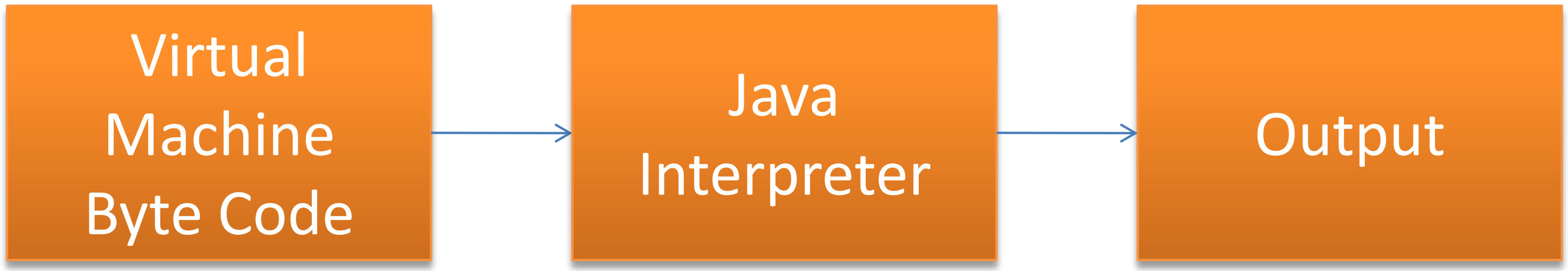
(1) Compiled and Interpreted :

Usually a computer language is either compiled or interpreted. But java combines both the approaches. First java compiler translates the source code into byte code instructions. ***Bytecode is a highly optimized set of instructions designed to be executed by* the Java run-time system, which is called the *Java Virtual Machine (JVM).*** Bytecodes are not machine instructions so in second stage, Java Interpreter generates machine code and execute the code. So we can say that java has Compiler and Interpreter.

Phase – I (Compilation)



Phase – II (Interpretation)



Features to Java

(2) Platform Independent :

Java programs can be moved easily from one system to another, anywhere and anytime. Due to this reason only the java is the most popular language on the internet. Java is Platform independent because it does not generates the machine code but it generates a code for JVM (Java Virtual Machine) The **Java** Virtual machine (**JVM**) is the virtual machine that run the **Java** bytecodes. so the program can be run on any machine without any problem.

Features to Java

(3) Object Oriented :

Java is pure Object Oriented Programming Language. Almost everything in java is an object. All programs, codes and data always resides inside the objects and classes.

Features to Java

(4) Secure :

Java is a secure language. It has compile time and run time checking for data types. On the other hand java provides the assurance that no viruses will be communicated with applets. One more thing is java does not support pointers so no question of memory address to user.

Features to Java

(5) Multithreaded:

Multithreaded means handling a multiple tasks simultaneously and it is a one of the best feature in Java. Java is able to perform multiple tasks simultaneously and that’s why it is very much beneficial for service side application developing. (At here, Thread is a tiny unit of one process and process is a collection of different threads)

Features to Java

(6) Distributed System :

Java is a language developed for distributed language for creating application on networks. It has ability to share both – data and programs. This will allow the programmers at various remote locations to work together on a single project.

Features to Java

(7) Simple and easy :

Java has many inbuilt and readymade features for developing an application means it is very easy to write therefore it is very easy to use and run.

Features to Java

(8) Robust :

Java is a robust language which provides a facility that at the time of compile, it automatically detects whether it has any errors or not in the application. (At here the word “Robust” means Reliable)

Features to Java

(9) Portable:

Java ensures portability because it is platform independent. For Example an int in Java is always a 32-bit integer. In C/C++, int can be a 16-bit integer, 32-bit or any other size integer whatever the compiler vendor likes. The only restriction is that the int type must have at least as many bytes as a short int and can not have more bytes than a long int. Having a fixed size for number types eliminates a major porting headache. Binary data is stored and transmitted in a fixed format, eliminating confusion about byte ordering. Strings are saved in standard Unicode format.

Features to Java

(10) High Performance:

Performance of java is such a great and impressive because it has to interpret the compiled bytecode. The feature of multithreading also responsible for making the performance faster. Java approach speeds up commonly used code tremendously because one has to do interpretation only once.

Features to Java

(11) Dynamic and extensible :

This is an important feature of java that supports the developer to develop dynamic content in web application with the help of JSP. In Dynamic concept the code is to added at runtime. Java is a dynamic language because it is capable for generating dynamic content.

Object Oriented Programming

Object Oriented Programming is a paradigm that provides many concepts such as inheritance, data binding, polymorphism etc.

Simula is considered as the first object-oriented programming language. The programming paradigm where everything is represented as an object, is known as truly object-oriented programming language.

Object Oriented Programming

Smalltalk is considered as the first truly object-oriented programming language. OOPs (Object Oriented Programming System)

Object Oriented Programming

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

Object Oriented Programming

Object

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical. **Object is Instance of Class**

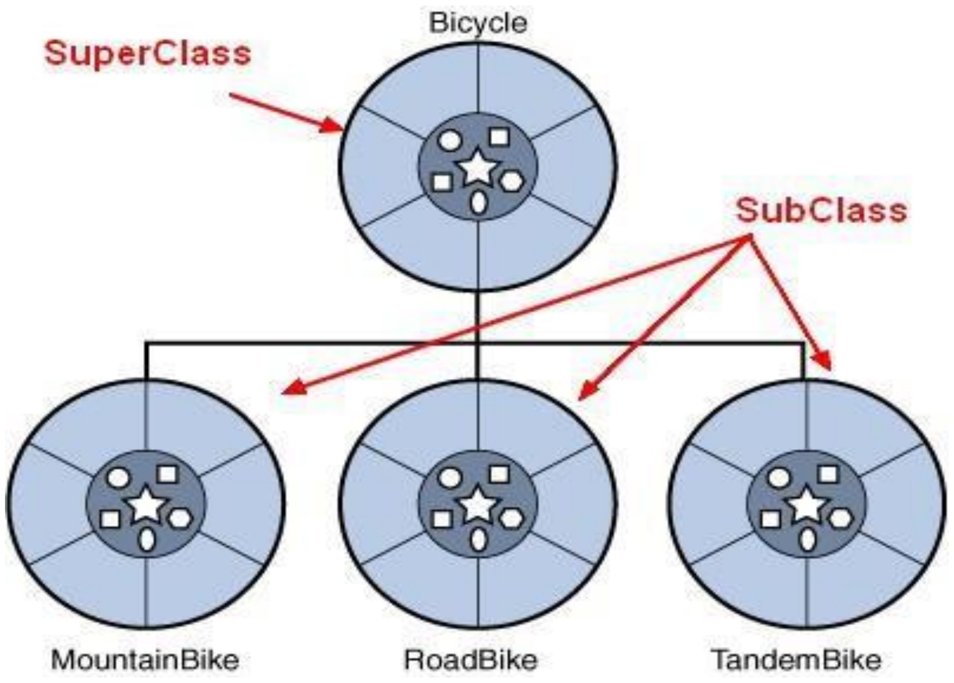
Class

Collection of objects is called class. It is a logical entity. It is also called a blue print of your whole programming structure which contains all the members of your class.

Object Oriented Programming

Inheritance

When one object acquires all the properties and behaviours of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.



Object Oriented Programming

* Different Terms used in Inheritance

Parent Class	Child Class
Super Class	Sub Class
Class	Derived Class

Object Oriented Programming

Polymorphism

“Poly” means many and “Morph” means Forms. One Name Many Forms.

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

In java, we use method overloading and method overriding to achieve polymorphism.
Another example can be to speak something e.g. cat speaks meaw, dog barks woof etc.

Object Oriented Programming

"Now Speak!"
"Woof"
"Quack"
"Meow"

Object Oriented Programming

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing. In java, we use abstract class and interface to achieve abstraction.

Object Oriented Programming

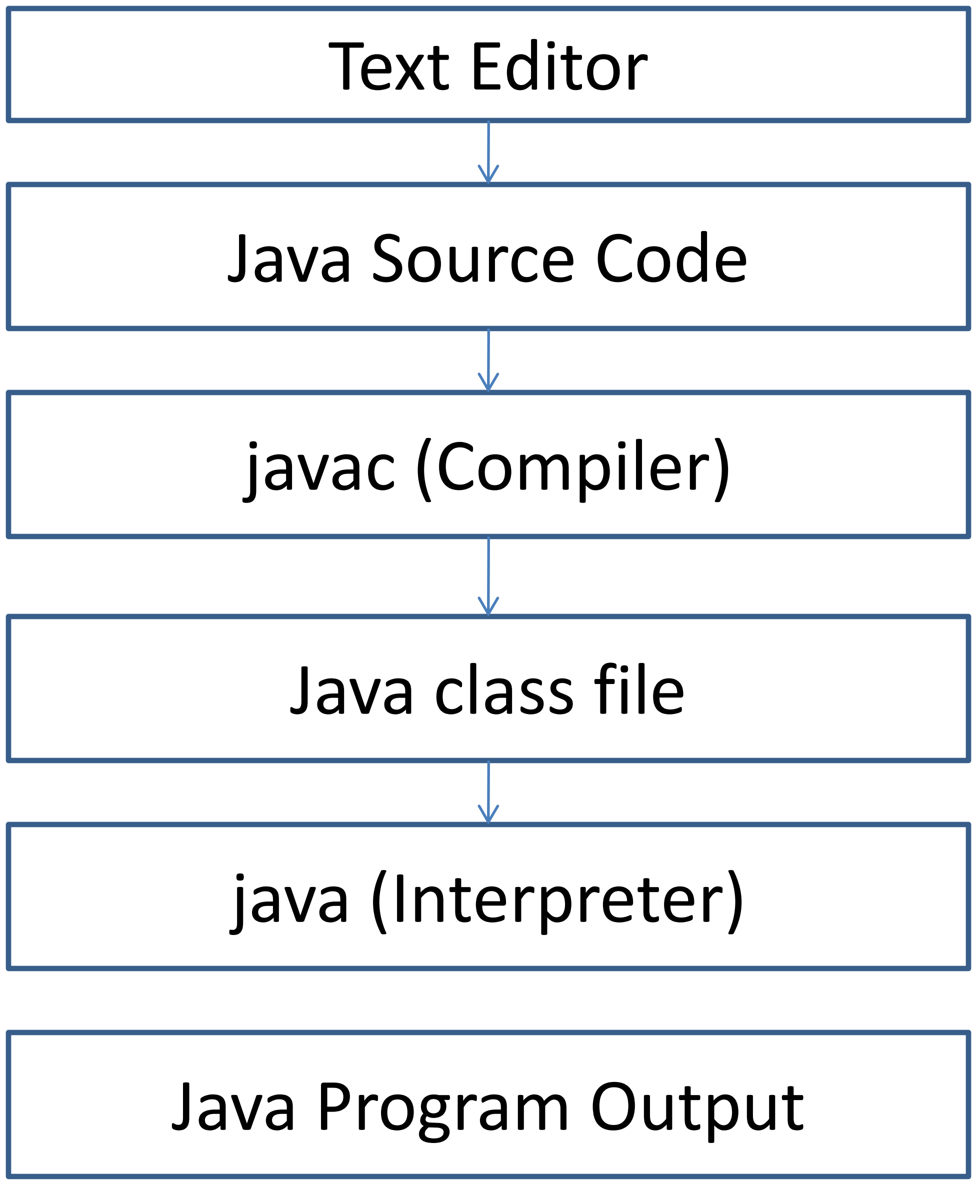
Encapsulation

Binding (or wrapping) code and data together into a single unit is known as encapsulation.

For example: capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Process of Building and Running Java Application



Difference between object and class

No.	Object	Class
1)	Object is an **instance** of a class.	Class is a **blueprint or template** from which objects are created.
2)	Object is a **real world entity** such as pen, laptop, mobile, bed, keyboard, mouse, chair etc.	Class is a **group of similar objects**.
3)	Object is a **physical** entity.	Class is a **logical** entity.
4)	Object is created through **new keyword** mainly e.g. Student s1=new Student();	Class is declared using **class keyword** e.g. class Student{}
5)	Object is created **many times** as per requirement.	Class is declared **once**.
6)	Object **allocates memory when it is created**.	Class **doesn't allocated memory when it is created**.

Constructor in Java

Constructor in java is a special type of method that is used to initialize the object.
Java constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object that is why it is known as constructor.

* Rules for creating java constructor
There are basically two rules defined for the constructor.
1. Constructor name must be same as its class name
2. Constructor must have no explicit return type

Types of java constructors
There are two types of constructors:
1. Default constructor (no-arg constructor)
2. Parameterized constructor

Java Default Constructor

A constructor that has no parameter is known as default constructor.

Syntax of default constructor:

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

Output:

```
Bike is created
```

Rule: If there is no constructor in a class, compiler automatically creates a default constructor.

What is the purpose of default constructor?

Default constructor provides the default values to the object like 0, null etc. depending on the type.

Example of default constructor that displays the default values

```
class Student3
{
        int id;
        String name;
        void display(){System.out.println(id+" "+name);}
        public static void main(String args[])
        {
                Student3 s1=new Student3();
                Student3 s2=new Student3();
                s1.display();
                s2.display();
        }
}
```

Output:

```
0 null
0 null
```

Explanation: In the above class, you are not creating any constructor so compiler provides you a default constructor. Here 0 and null values are provided by default constructor.

Java parameterized constructor

Constructors that have parameters is known as parameterized constructor.

Why use parameterized constructor?
Parameterized constructor is used to provide different values to the distinct objects.

Example of parameterized constructor
In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

Java parameterized constructor

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

Output:

```
111 Karan
222 Aryan
```

Instance variables:

Instance variables are declared in a class, but outside a method, constructor or any block.

When a space is allocated for an object in the heap, a slot for each instance variable value is created.

Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.

Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.

Instance variables can be declared in class level before or after use.

Instance variables:

Access modifiers can be given for instance variables.

The instance variables are visible for all methods, constructors and block in the class. Normally, it is recommended to make these variables private (access level). However visibility for subclasses can be given for these variables with the use of access modifiers.

Instance variables have default values. For numbers the default value is 0, for Booleans it is false and for object references it is null. Values can be assigned during the declaration or within the constructor.

Instance variables can be accessed directly by calling the variable name inside the class. However within static methods and different class (when instance variables are given accessibility) should be called using the fully qualified name .ObjectReference.VariableName.

Instance variables:

Example:

```
import java.io.*;
public class Employee{
// this instance variable is visible for any child class.
public String name;
// salary variable is visible in Employee class only.
private double salary;

public Employee (String empName)
{ name = empName; }

// The salary variable is assigned a value.
public void setSalary(double empSal){
salary = empSal;
}
```

Instance variables:

```
// This method prints the employee details.
public void printEmp(){
System.out.println("name : " + name ); System.out.println("salary :" +
salary);
}

public static void main(String args[]){
Employee empOne = new Employee("Ransika");
empOne.setSalary(1000);
empOne.printEmp();
}
}
```

This would produce the following result: name : Ransika
salary :1000.0

Java Enum

Enum in java is a data type that contains fixed set of constants. It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY and SATURDAY) , directions (NORTH, SOUTH, EAST and WEST) etc.

The java enum constants are static and final implicitly. It is available from JDK 1.5.
Java Enums can be thought of as classes that have fixed set of constants.

Points to remember for Java Enum

- enum improves type safety
- enum can be easily used in switch
- enum can have fields, constructors and methods
- enum may implement many interfaces but cannot extend any class because it internally extends Enum class

What is the purpose of values() method in enum?

The java compiler internally adds the values() method when it creates an enum. The values () method returns an array containing all the values of the enum.

```
* Simple example of java enum
class EnumExample1
{
public enum Season { WINTER, SPRING, SUMMER, FALL }
public static void main(String[] args) { for (Season s : Season.values())
System.out.println(s);
}
}
```

```
Output
WINTER
SPRING
SUMMER
FALL
```

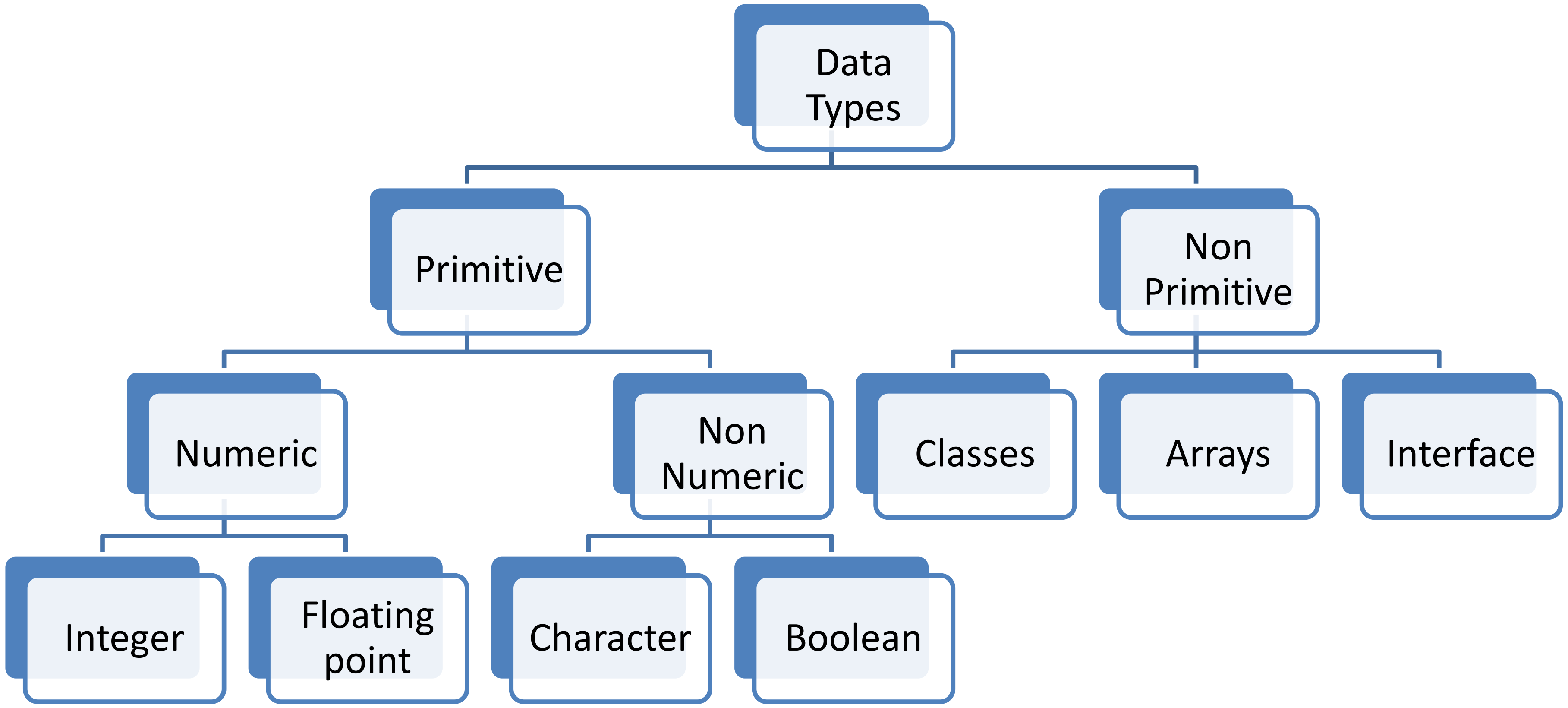
Keywords

Keywords have special meaning to the Java compiler. They help in identifying a data type name or program construct name.

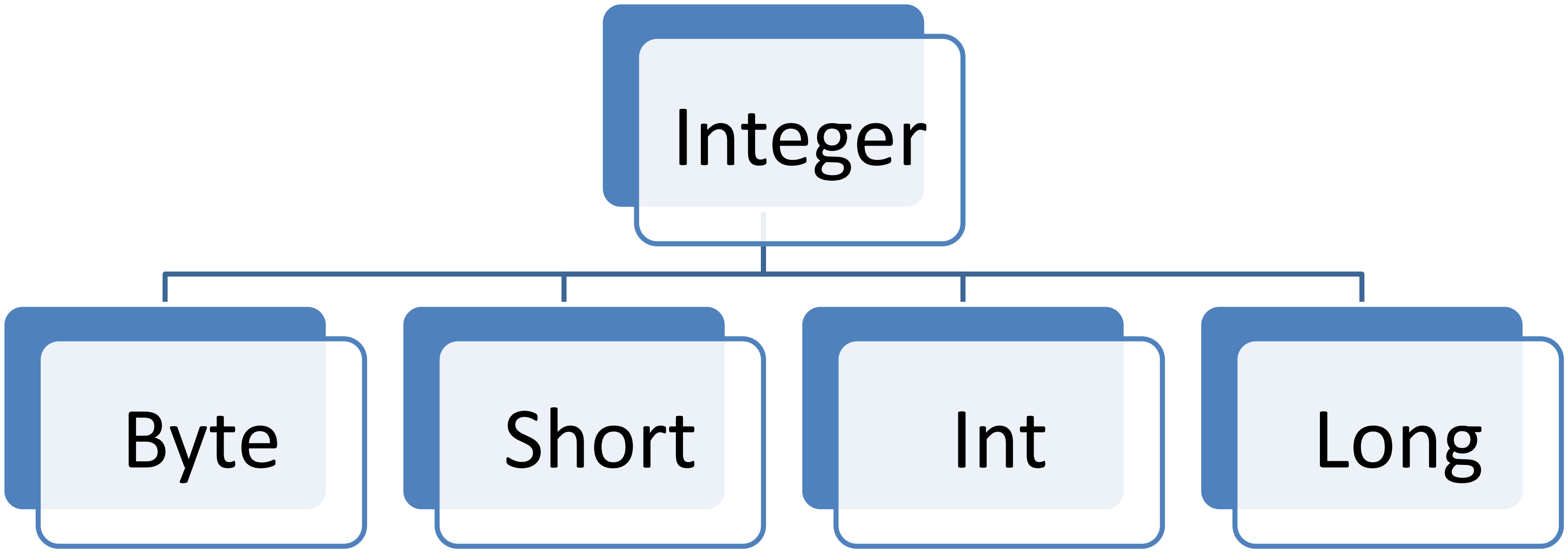
Java Programming Language Keywords: Keywords also termed as a reserve keyword and it cannot be used as a name of variable, class, method etc.

abstract	continue	For	new	switch
assert	default	Goto	package	synchronized
boolean	do	If	private	this
break	double	implements	protected	throw
byte	else	Import	public	throws
case	enum	instanceof	return	transient
catch	extends	Int	short	try
char	final	interface	static	void
class	finally	Long	strictfp	volatile
const	float	Native	super	while

Data Types

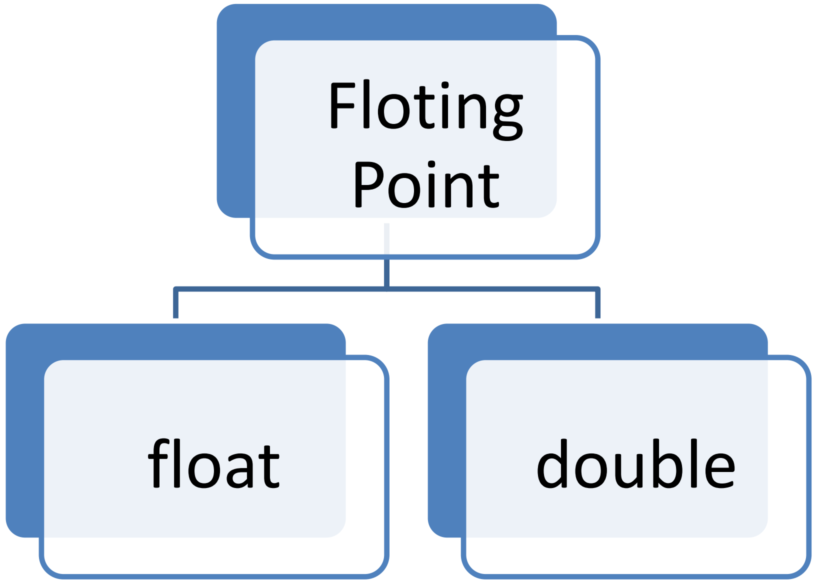

Data Types
Primitive
Non Primitive
Numeric
Non Numeric
Classes
Arrays
Interface
Integer
Floating point
Character
Boolean

Data Types



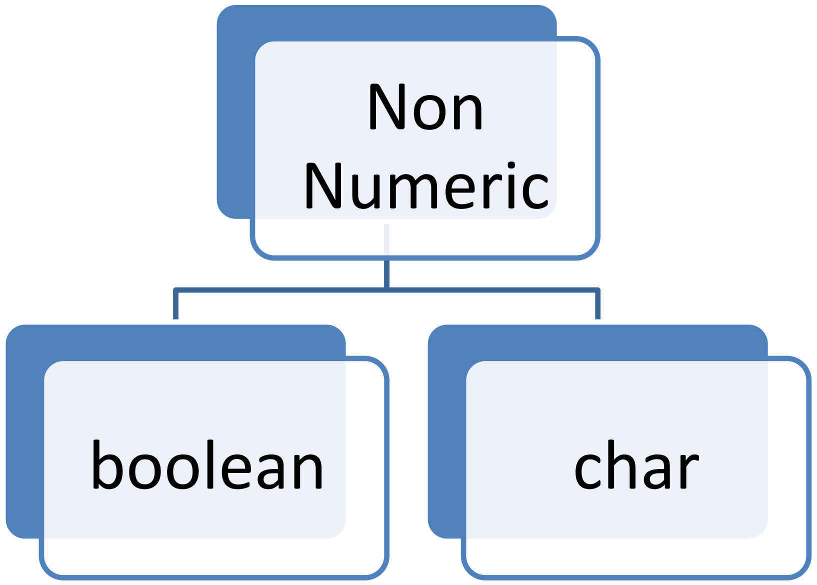
Type	Size	Min. Value	Max. Value
byte	One byte	-128	127
short	Two bytes	-32768	32767
int	Four bytes	-2147483648	2147483647
long	Eight bytes	-9223372036854775808	9223372036854775807

Data Types



Type	Size	Min. Value	Max. Value
float	Four bytes	1.40129846432481707e-45	3.40282346638528860e+38
double	Eight bytes	4.94065645841246544e-324	1.79769313486231570e+308

Data Types



Type	Size	Min. Value	Max. Value
boolean	One Byte	Only True/False Value	
char	Two Bytes	0	65,535

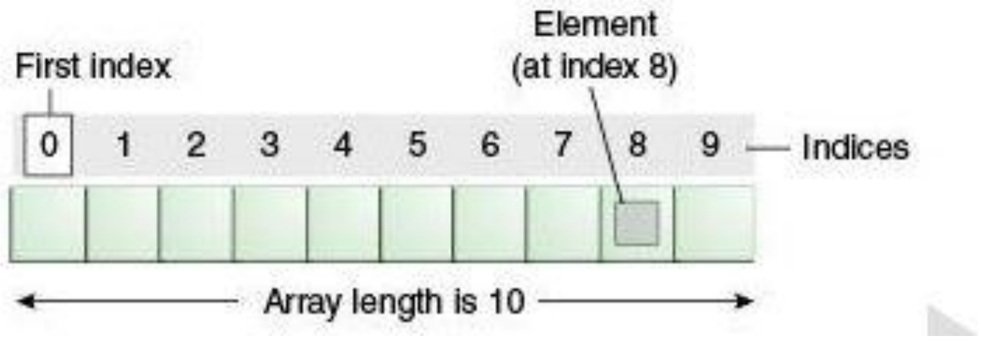
2) Non-Primitive (Reference) Data type

A reference data type is used to refer to an object. A reference variable is declared to be of specific and that type can never be change. Classes, Arrays, Interface are an example of Non Primitive types.

Java Array

Normally, array is a collection of similar type of elements that have contiguous memory location. Java array is an object the contains elements of similar data type. It is a data structure where we store similar elements. We can store only fixed set of elements in a java array.

Array in java is index based, first element of the array is stored at 0 index.



Java Array

Advantage of Java Array

- Code Optimization: It makes the code optimized, we can retrieve or sort the data easily.
- Random access: We can get any data located at any index position.

Disadvantage of Java Array

- 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

There are two types of array.

1. Single Dimensional Array
2. Multidimensional Array

Single Dimensional Array in java

It is a simple array which can store the multiple elements inside which must belong to the same data type.
Syntax to Declare an Array in java

```
dataType[] arr; (or) dataType arr[];
```

Single Dimensional Array in java

```
class ArrayEx
{

        public static void main(String args[])
        {
                int a[]=new int[3];
                a[0]=10;
                a[1]=20;
                a[2]=30;
                for(int i=0;i<3;i++)
                {
                        System.out.println(a[i]);
                }
        }
}
```

Multidimensional array in java

In such case, data is stored in row and column based index (also known as matrix form).

Syntax to Declare Multidimensional Array in java

dataType[][] arrayRefVar; (or) dataType [][]arrayRefVar; (or) dataType arrayRefVar[][]; (or) dataType []arrayRefVar[];

* Example to instantiate Multidimensional Array in java
int[][] arr=new int[3][3];//3 row and 3 column

Multidimensional array in java

```
class MethodOverload
{

        public static void main(String args[])
        {
                int arr[][]=new int[3][3];
                arr[0][0]=1;
                arr[0][1]=2;
                arr[0][2]=3;
                arr[1][0]=4;
                arr[1][1]=5;
                arr[1][2]=6;
                arr[2][0]=7;
                arr[2][1]=8;
                arr[2][2]=9;
```

Multidimensional array in java

```
                for(int i=0;i<3;i++)
                {
                        for(int j=0;j<3;j++)
                        {
                                System.out.print(arr[i][j]+" ");
                        }
                        System.out.print("\n");
                }
        }
}
```

Scanner Class

The Scanner class is used to get user input, and it is found in the java.util package.

To use the Scanner class, create an object of the class and use any of the available methods found in the Scanner class documentation. In our example, we will use the nextLine() method, which is used to read Strings:

```
import java.util.Scanner;  // Import the Scanner class

class Main {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);  // Create a Scanner object
    System.out.println("Enter username");

    String userName = m System.out.println("Username is: " + userName);  //
Output user inputyObj.nextLine();  // Read user input
}
}
```

Scanner Class

Method	**Description**
nextBoolean()	Reads a boolean value from the user
nextByte()	Reads a byte value from the user
nextDouble()	Reads a double value from the user
nextFloat()	Reads a float value from the user
nextInt()	Reads a int value from the user
nextLine()	Reads a String value from the user
nextLong()	Reads a long value from the user
nextShort()	Reads a short value from the user

Super Types and Sub Types

Java has defined a relationship between data types. We have a relationship of super-type and sub-type between the data types. A data type is a sub-type of given data type if it is a special kind of the given data type. E.g. in case of primitive data types int is a sub-type of long, as long can always be used wherever int is used. In other words we can say that all ints are long but not all longs may be ints. i.e. we can use super- type for sub-type but we cannot use sub-type for super type.

Super Types and Sub Types

Java has defined a relationship between data types. We have a relationship of super-type and sub-type between the data types. A data type is a sub-type of given data type if it is a special kind of the given data type. E.g. in case of primitive data types int is a sub-type of long, as long can always be used wherever int is used. In other words we can say that all ints are long but not all longs may be ints. i.e. we can use super-type for sub-type but we cannot use sub-type for super type.

Super Types and Sub Types

$$\begin{tikzcd}
 & \boxed{\text{double}} \arrow[d] & \\
 & \boxed{\text{float}} \arrow[d] & \\
 & \boxed{\text{long}} \arrow[d] & \\
 & \boxed{\text{int}} \arrow[dl] \arrow[dr] & \\
\boxed{\text{short}} \arrow[d] & & \boxed{\text{char}} \\
\boxed{\text{byte}} & &
\end{tikzcd}$$

Unicode System

Unicode and Ascii code are both ways that computer languages store characters as numbers. Ascii stands for "American Standard Code for Information Interchange" and it allows encoding for 128 characters. Fine for the English language, but not enough for others.

Unicode can handle 100,000 characters, so by using this encoding scheme, Java allows programmers to work with printed languages from around the world!
Unicode is a universal international standard character encoding that is capable of representing most of the world's written languages.

Why java uses Unicode System?
Before Unicode, there were many language standards:

- ASCII (American Standard Code for Information Interchange) for the United States.
- ISO 8859-1 for Western European Language. (International Organization for Standardization)
- KOI-8 for Russian.(KSC Operation Instruction)
- GB18030 and BIG-5 for chinese, and so on.

Unicode System

This caused two problems:

- A particular code value corresponds to different letters in the various language standards.
- The encodings for languages with large character sets have variable length. Some common characters are encoded as single bytes, other require two or more byte.

To solve these problems, a new language standard was developed i.e. Unicode System. In unicode, character holds 2 byte, so java also uses 2 byte for characters.
lowest value:\u0000 highest value:\uFFFF

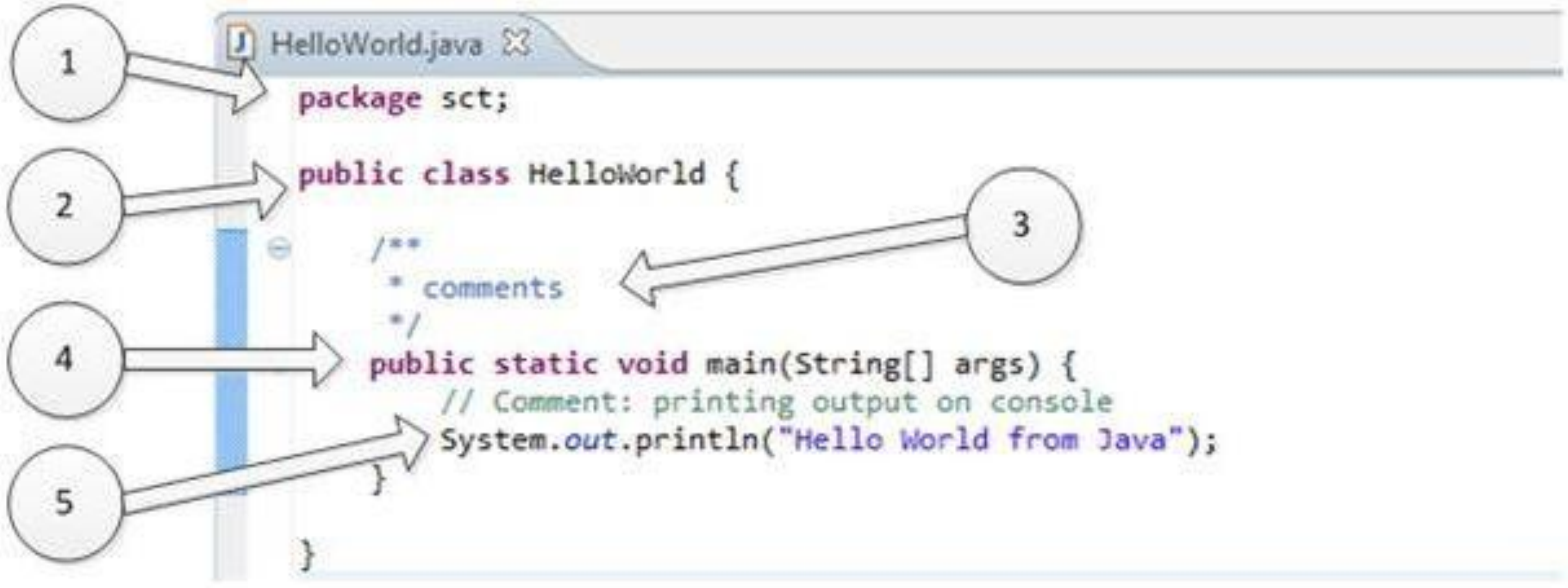
Identifiers in Java

All Java components require names. Name used for classes, methods, interfaces and variables are called Identifier. Identifier must follow some rules. Here are the rules:

- All identifiers must start with either a letter(a to z or A to Z) or currency character($) or an underscore.
- After the first character, an identifier can have any combination of characters.
- A Java keyword cannot be used as an identifier.
- Identifiers in Java are case sensitive, foo and Foo are two different identifiers.

Java Program Structure

Let's use example of HelloWorld Java program to understand structure and features of class. This program is written on few lines, and its only task is to print "Hello World from Java" on the screen. Refer the following picture.



```
package sct;

public class HelloWorld {

    /**
     * comments
     */
    public static void main(String[] args) {
        // Comment: printing output on console
        System.out.println("Hello World from Java");
    }

}
```

Java Program Structure

1. “package sct”:

It is package declaration statement. The package statement defines a name space in which classes are stored. Package is used to organize the classes based on functionality. If you omit the package statement, the class names are put into the default package, which has no name. Package statement cannot appear anywhere in program. It must be first line of your program or you can omit it.

2. “public class HelloWorld”:

This line has various aspects of java programming.

a.public: This is access modifier keyword which tells compiler access to class. Various values of access modifiers can be public, protected, private or default (no value).

b.class: This keyword used to declare class. Name of class (HelloWorld) followed by this keyword.

Java Program Structure

3.Comments section:

We can write comments in java in two ways.

a.Line comments: It start with two forward slashes (//) and continue to the end of the current line. Line comments do not require an ending symbol.

b.Block comments start with a forward slash and an asterisk (/*) and end with an asterisk and a forward slash (*/).Block comments can also extend across as many lines as needed.

Java Program Structure

4.“public static void main (String [ ] args)”:

Its method (Function) named main with string array as argument.

a.public : Access Modifier

b.static: static is reserved keyword which means that a method is accessible and usable even though no objects of the class exist.

c.void: This keyword declares nothing would be returned from method. Method can return any primitive or object.

d.Method content inside curly braces. { }

Java Program Structure

5.System.out.println("Hello World from Java") :

a. System: It is name of Java utility class.

b. out:It is an object which belongs to System class but actually it is object of PrintStream class of io package

c. println: It is utility method of PrintStream class which is used to send any String to console.

d. “Hello World from Java”:It is String literal set as argument to println method.

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) Relational Operators
(3) Bitwise Operators
(4) Logical Operators
(5) Assignment Operators
(6) Miscellaneous / Other Operator

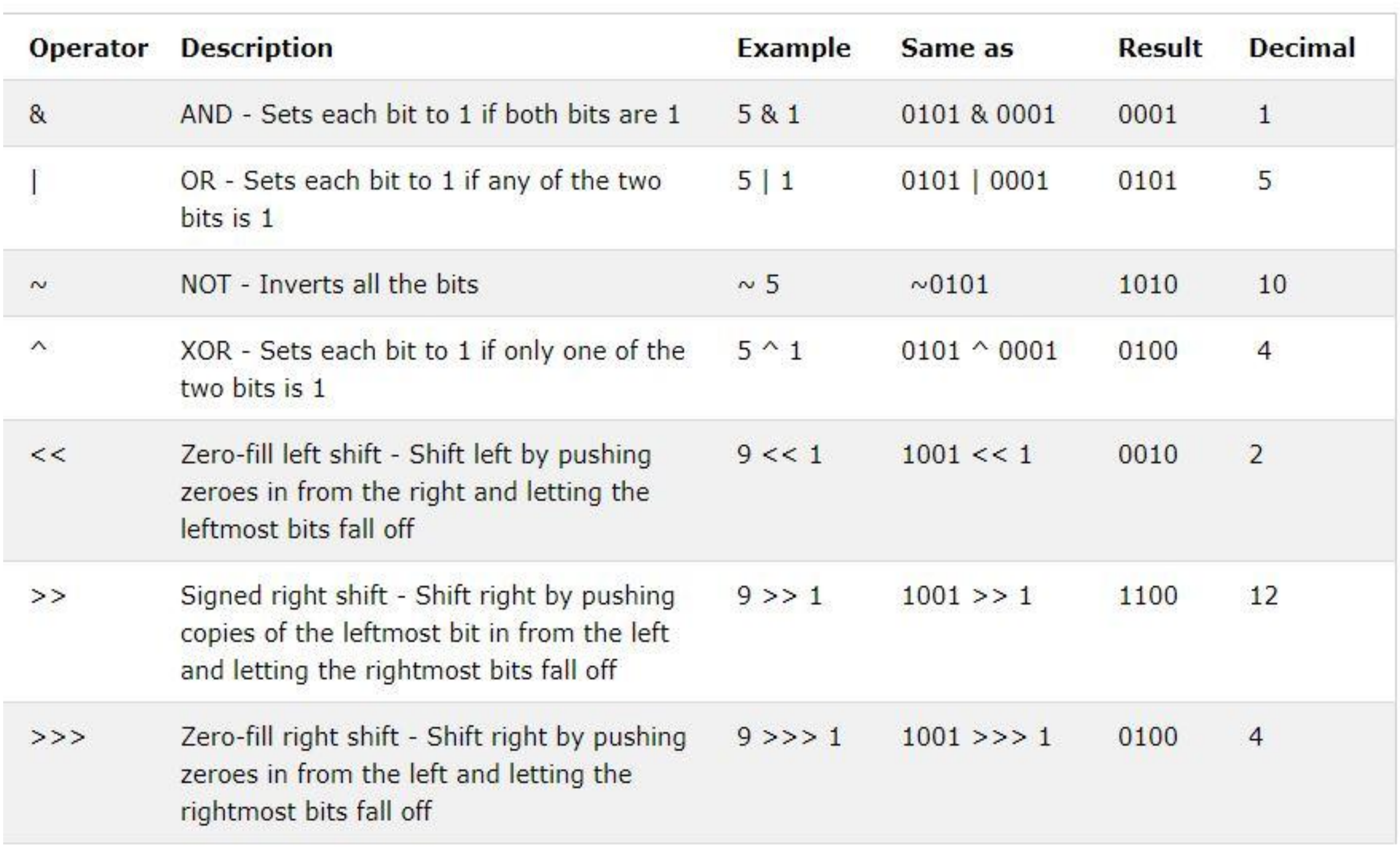
(1) Arithmetic Operators:

Operator	Description	Example
+	Addition - Adds values on either side of the operator	A + B will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	A - B will give -10
*	Multiplication - Multiplies values on either side of the operator	A * B will give 200
/	Division - Divides left hand operand by right hand operand	B / A will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	B % A will give 0
++	Increment - Increases the value of operand by 1	B++ gives 21
--	Decrement - Decreases the value of operand by 1	B-- gives 19

(2) Relational Operators:

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

(3) Bitwise Operators:



Operator	Description	Example	Same as	Result	Decimal
&	AND - Sets each bit to 1 if both bits are 1	5 & 1	0101 & 0001	0001	1
|	OR - Sets each bit to 1 if any of the two bits is 1	5 | 1	0101 | 0001	0101	5
~	NOT - Inverts all the bits	~ 5	~0101	1010	10
^	XOR - Sets each bit to 1 if only one of the two bits is 1	5 ^ 1	0101 ^ 0001	0100	4
<<	Zero-fill left shift - Shift left by pushing zeroes in from the right and letting the leftmost bits fall off	9 << 1	1001 << 1	0010	2
>>	Signed right shift - Shift right by pushing copies of the leftmost bit in from the left and letting the rightmost bits fall off	9 >> 1	1001 >> 1	1100	12
>>>	Zero-fill right shift - Shift right by pushing zeroes in from the left and letting the rightmost bits fall off	9 >>> 1	1001 >>> 1	0100	4

(4) Logical Operators:

The following table lists the logical operators:
Assume Boolean variables A holds true and variable B holds false, then:

Show Examples

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non- zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is

(5) Assignment Operator:

There are following assignment operators supported by Java language:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A

(5) Assignment Operator:

%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

(6) Misc./ Other Operators

(i) Conditional Operator (? :):

Conditional operator is also known as the ternary operator. This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide which value should be assigned to the variable. The operator is written as:

variable x = (expression) ? value if true : value if false

Following is the example:

```
public class Test {
public static void main(String args[]){ int a , b;
a = 10;
b = (a == 1) ? 20: 30;
System.out.println( "Value of b is : " + b );

b = (a == 10) ? 20: 30;
System.out.println( "Value of b is : " + b );
}
}
```

This would produce the following result: Value of b is : 30

Value of b is : 20

(6) Misc./ Other Operators

(ii) Instance of Operator:

This operator is used only for object reference variables. The operator checks whether the object is of a particular type (class type or interface type). Instance of operator is written as:

(Object reference variable) instanceof (class/interface type)

If the object referred by the variable on the left side of the operator passes the IS-A check for the class/interface type on the right side, then the result will be true. Following is the example:

```
public class Test {


public static void main(String args[]){ String name = "James";
// following will return true since name is type of String boolean result = name instanceof
String; System.out.println( result );
}
}
```

This would produce the following result:

```
true
```

String Concatenation

+ operator will be used for doing string concatenation.

Remember that String concatenation will be done whenever any one of the two operands is of String type.

e.g.

(1) 5 + 5 + “ is ten” will gives output “10 is ten”

(2) “Fifty five is “ + 5 + 5 will gives output “Fifty five is 55”

Type Conversion (Typecasting)

Java supports two types of castings – primitive data type casting and reference type casting. Reference type casting is nothing but assigning one Java object to another object. It comes with very strict rules and is explained clearly in Object Casting. Now let us go for data type casting.

Java data type casting comes with 2 types.

1. Implicit casting
2. Explicit casting

Type Conversion (Typecasting)

1.　　Implicit casting (widening conversion)

A data type of lower size (occupying less memory) is assigned to a data type of higher size. This is done implicitly by the JVM. The lower size is widened to higher size. This is also named as automatic type conversion.

Examples:

```
double y;
int x = 10;             // occupies 4 bytes double
y = x;                  // occupies 8 bytes
System.out.println(y);       // prints 10.0
```

In the above code 4 bytes integer value is assigned to 8 bytes double value.

Type Conversion (Typecasting)

2. Explicit casting (narrowing conversion)

A data type of higher size (occupying more memory) cannot be assigned to a data type of lower size. This is not done implicitly by the JVM and requires explicit casting; a casting operation to be performed by the programmer. The higher size is narrowed to lower size.

```
double x = 10.5; // 8 bytes
int y = x; // 4 bytes ;          raises compilation error
```

In the above code, 8 bytes double value is narrowed to 4 bytes int value. It raises error.

Let us explicitly type cast it.

```
double x = 10.5;
int y = (int) x;
```

The double x is explicitly converted to int y. The thumb rule is, on both sides, the same data type should exist.

Widening Conversion

A widening conversion is the conversion of a sub-type to one of its super types. For numeric types, the following are the widening conversions.

(1) byte to short,int,long,float and double
(2) short to int,long,float and double
(3) int to long,float and double
(4) long to float and double
(5) float to double

Narrowing Conversion

A Narrowing conversion is the conversion of a super-type to one of its sub-type. For numeric types, following are the narrowing conversions.

(1) double to byte,short,int,long and float
(2) float to byte,short,int and long
(3) long to byte,short and int
(4) int to byte and short
(5) short to byte

Decision Making Statement

Decision making statement

Here are two types of decision making statements in Java.
They are:

1. IF statements
2. switch statements

IF Statements

(a) IF Statement:

IF statement consists of a Boolean expression followed by one or more statements.

Syntax:

The syntax of a if statement is:

```
if(Boolean_expression)
{
        //Statements will execute if the Boolean expression is true
}
```

If the Boolean expression evaluates to true then the block of code inside the if statement will be executed. If not the first set of code after the end of the if statement (after the closing curly brace) will be executed.

IF Statements

Example:

```
public class Test
{

public static void main(String args[])
{
            int x = 10;

if( x < 20 ){
System.out.print("This is if statement");
}
}
}
```

This would produce the following result: This is if statement

IF Statements

(b) if...else Statement:

A if statement can be followed by an optional else statement, which executes when the Boolean expression is false.

Syntax:

The syntax of a if...else is: if(Boolean_expression) {

```
//Executes when the Boolean expression is true
} else {
//Executes when the Boolean expression is false
}
```

IF Statements

Example:

```
public class Test {

public static void main(String args[]){ int x = 30;

if( x < 20 ){
System.out.print("This is if statement");
}else{
System.out.print("This is else statement");
}
}
}
```

This would produce the following result: This is else statement

IF Statements

(c) The if...else if...else Statement: (Multiple IF)

if statement can be followed by an optional else if...else statement, which is very useful to test various conditions using single if...else if statement.

When using if , else if , else statements there are few points to keep in mind.

- if can have zero or one else's and it must come after any else if's.
- if can have zero to many else if's and they must come before the else.
- Once an else if succeeds, none of the remaining else if's or else's will be tested.

IF Statements

Syntax:

```
The syntax of an if...else is: if(Boolean_expression 1) {
//Executes when the Boolean expression 1 is true
} else if(Boolean_expression 2) {
//Executes when the Boolean expression 2 is true
} else if(Boolean_expression 3) {
//Executes when the Boolean expression 3 is true
} else {
//Executes when the none of the above condition is true.
}
```

IF Statements

(d) Nested if...else Statement:

It is always legal to nest if-else statements which means you can use one if or else if statement inside another if or else if statement.

Syntax:

```
The syntax for a nested if...else is as follows: if(Boolean_expression 1) {
//Executes when the Boolean expression 1 is true if(Boolean_expression 2) {
//Executes when the Boolean expression 2 is true
}
}
```

You can nest else if...else in the similar way as we have nested if statement.

IF Statements

Example:

```
public class Test {

public static void main(String args[]){ int x = 30;
int y = 10;

if( x == 30 ){ if( y == 10 ){
System.out.print("X = 30 and Y = 10");

}
}
}
}
```

This would produce the following result:

```
X = 30 and Y = 10
```

(2) The switch Statement:

A switch statement allows a variable to be tested for equality against a list of values.
Each value is called a case, and the variable being switched on is checked for each case.

Syntax:

```
The syntax of switch is: switch(expression) {
case value :
//Statements break; //optional
case value :
//Statements break; //optional
//You can have any number of case statements. default : //Optional
//Statements
}
```

(2) The switch Statement:

The following rules apply to a switch statement:

- The variable used in a switch statement can only be a byte, short, int, or char.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The value for a case must be the same data type as the variable in the switch and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a break statement is reached.
- When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a break. If no break appears, the flow of control will fall through to subsequent cases until a break is reached.
- A switch statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.

Loops in Java

There may be a situation when we need to execute a block of code several number of times, and is often referred to as a loop.

Java has very flexible three looping mechanisms. You can use one of the following three loops:

(1) while Loop

(2) do...while Loop

(3) for Loop

As of Java 5, the enhanced for loop was introduced. This is mainly used for Arrays.

(1) while Loop:

A while loop is a control structure that allows you to repeat a task a certain number of times.

Syntax:

The syntax of a while loop is:

```
while(Boolean expression)
{
//Statements
}
```

When executing, if the boolean_expression result is true, then the actions inside the loop will be executed. This will continue as long as the expression result is true. Here, key point of the while loop is that the loop might not ever run. When the expression is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

(1) while Loop:

Example:

```
public class Test {

public static void main(String args[]) { int x = 10;
while( x < 20 ) { System.out.print("value of x : " + x ); x++;
System.out.print("\n");
}}
}
```

This would produce the following result: value of x : 10

```
value of x : 11
value of x : 12
value of x : 13
value of x : 14
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19
```

(2) do...while Loop:

A do...while loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

Syntax:

The syntax of a do...while loop is: do

```
{
//Statements
} while(Boolean expression);
```

Notice that the Boolean expression appears at the end of the loop, so the statements in the loop execute once before the Boolean is tested.

If the Boolean expression is true, the flow of control jumps back up to do, and the statements in the loop execute again. This process repeats until the Boolean expression is false.

(2) do...while Loop:

Example:

```
public class Test {

public static void main(String args[]){ int x = 10;

do{
System.out.print("value of x : " + x ); x++;
System.out.print("\n");
}while( x < 20 );
}
}
```

This would produce the following result:

```
value of x : 10
value of x : 11 value of x : 12 value of x : 13 value of x : 14 value of x : 15 value of x :
16 value of x : 17 value of x : 18 value of x : 19
```

(3) for Loop:

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times. A for loop is useful when you know how many times a task is to be repeated.

Syntax:
The syntax of a for loop is:

```
for(initialization; Boolean_expression; update)
{
//Statements
}
```

(3) for Loop:

Here is the flow of control in a for loop:

- The initialization step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.

- Next, the Boolean expression is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement past the for loop.

- After the body of the for loop executes, the flow of control jumps back up to the update statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the Boolean expression.

- The Boolean expression is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then update step, then Boolean expression). After the Boolean expression is false, the for loop terminates.

(3) for Loop:

Example:

```
public class Test {
public static void main(String args[]) { for(int x = 10; x < 20; x = x+1) {
System.out.print("value of x : " + x );
System.out.print("\n");
}
}
}
```

(4) Enhanced for loop in Java:

As of Java 5, the enhanced for loop was introduced. This is mainly used for Arrays.

Syntax:

The syntax of enhanced for loop is: for(declaration : expression)

```
{
//Statements
}
```

- Declaration: The newly declared block variable, which is of a type compatible with the elements of the array you are accessing. The variable will be available within the for block and its value would be the same as the current array element.

- Expression: This evaluates to the array you need to loop through. The expression can be an array variable or method call that returns an array.

The break Keyword

The break keyword is used to stop the entire loop. The break keyword must be used inside any loop or a switch statement. The break keyword will stop the execution of the innermost loop and start executing the next line of code after the block.

Syntax:

The syntax of a break is a single statement inside any loop: break;

Example:

```
public class Test {
public static void main(String args[]) { int [] numbers = {10, 20, 30, 40, 50};

for(int x : numbers ) { if( x == 30 ) {
break;
}
System.out.print( x ); System.out.print("\n");
}}
}
```

This would produce the following result: 10

20

The continue Keyword:

The continue keyword can be used in any of the loop control structures. It causes the loop to immediately jump to the next iteration of the loop. In a for loop, the continue keyword causes flow of control to immediately jump to the update statement.
In a while loop or do/while loop, flow of control immediately jumps to the Boolean expression.

Syntax:
The syntax of a continue is a single statement inside any loop: continue;
Example:

```
public class Test {
public static void main(String args[]) { int [] numbers = {10, 20, 30, 40, 50};
for(int x : numbers ) { if( x == 30 ) {
continue;
}
System.out.print( x ); System.out.print("\n");
}
}}
```

This would produce the following result:

```
10
20
40
50
```

Java Garbage Collection

In java, garbage means unreferenced objects. Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects. To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

Advantage of Garbage Collection

o It makes java memory efficient because garbage collector removes the unreferenced objects from heap memory.

o It is automatically done by the garbage collector(a part of JVM) so we don't
need to make extra efforts.

How can an object be unreferenced?

There are many ways:

o By nulling the reference

o By assigning a reference to another

o By annonymous object etc.

Java Garbage Collection

1) By nulling a reference:

Employee e=new Employee(); e=null;

2) By assigning a reference to another:
Employee e1=new Employee(); Employee e2=new Employee();
e1=e2; //now the first object referred by e1 is available for garbage collection

3) By anonymous object:

new Employee();

Java Garbage Collection

finalize() method
The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:

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

gc() method
The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

```
public static void gc(){}
```

Note: Garbage collection is performed by a daemon thread called Garbage Collector(GC). This thread calls the finalize() method before object is garbage collected.

Java Garbage Collection

Simple Example of garbage collection in java

```
public class TestGarbage1{
public void finalize(){System.out.println("object is garbage collected");}
public static void main(String args[]){
TestGarbage1 s1=new TestGarbage1();
TestGarbage1 s2=new TestGarbage1();
s1=null;
s2=null;
System.gc();
}
}
```

output:

```
object is garbage collected
object is garbage collected
```

Note: Neither finalization nor garbage collection is guaranteed.

Instance initializer block:

Instance Initializer block is used to initialize the instance data member. It run each time when object of the class is created.
The initialization of the instance variable can be directly but there can be performed extra operations while initializing the instance variable in the instance initializer block.

Q) What is the use of instance initializer block while we can directly assign a value in instance data member? For example:

```
class Bike{
int speed=100;
}
```

Why use instance initializer block?
Suppose I have to perform some operations while assigning value to instance data member e.g. a for loop to fill a complex array or error handling etc.

Instance initializer block:

Example of instance initializer block

Let's see the simple example of instance initializer block the performs initialization.

```
class Bike7{
int speed;
Bike7(){System.out.println("speed is "+speed);}

{speed=100;}

public static void main(String args[]){ Bike7 b1=new Bike7();
Bike7 b2=new Bike7();
}
}
```

```
Output:speed is 100
speed is 100
```

Instance initializer block:

- There are three places in java where you can perform operations:
    - method
    - constructor
    - block

What is invoked first, instance initializer block or constructor?

```
class Bike8{
	int speed;
	Bike8(){System.out.println("constructor is invoked");}
	{System.out.println("instance initializer block invoked");}
	public static void main(String args[]){
	Bike8 b1=new Bike8();
	Bike8 b2=new Bike8();
	}
}
```

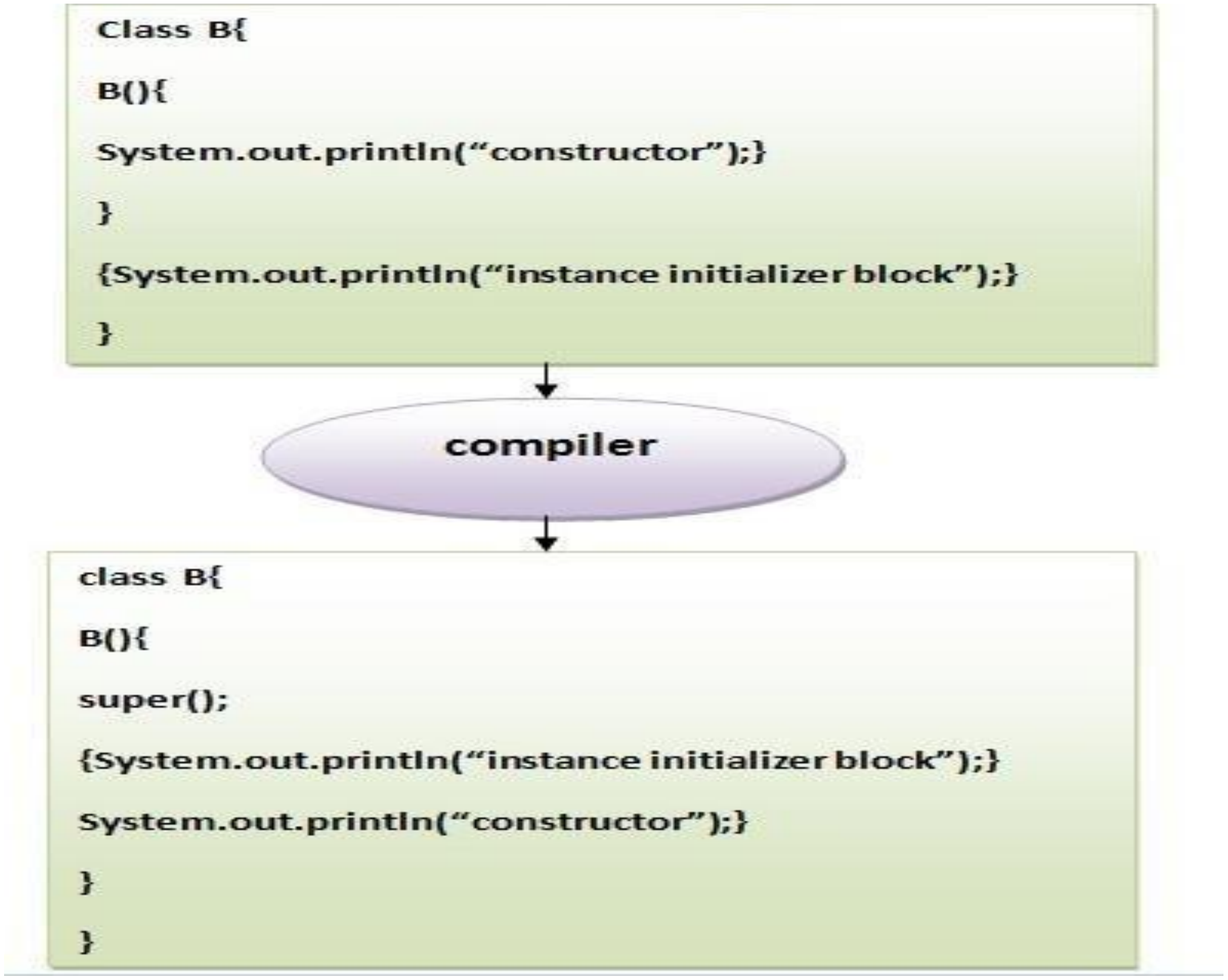
Output:instance initializer block invoked constructor is invoked instance initializer block invoked constructor is invoked

What is invoked first, instance initializer block or constructor?

In the above example, it seems that instance initializer block is firstly invoked but NO. Instance intializer block is invoked at the time of object creation. The java compiler copies the instance initializer block in the constructor after the first statement super(). So firstly, constructor is invoked. Let's understand it by the figure given below:

Note: The java compiler copies the code of instance initializer block in every constructor.

What is invoked first, instance initializer block or constructor?



```
Class B{
B(){
System.out.println("constructor");}
}
{System.out.println("instance initializer block");}
}
```

```
class B{
B(){
super();
{System.out.println("instance initializer block");}
System.out.println("constructor");}
}
}
```

What is invoked first, instance initializer block or constructor?

Rules for instance initializer block:

There are mainly three rules for the instance initializer block. They are as follows:

- The instance initializer block is created when instance of the class is created.
- The instance initializer block is invoked after the parent class constructor is invoked (i.e. after super() constructor call).
- The instance initializer block comes in the order in which they appear.

What is invoked first, instance initializer block or constructor?

Rules for instance initializer block:

There are mainly three rules for the instance initializer block. They are as follows:

- The instance initializer block is created when instance of the class is created.
- The instance initializer block is invoked after the parent class constructor is invoked (i.e. after super() constructor call).
- The instance initializer block comes in the order in which they appear.

Static keyword

The static keyword in Java is used for memory management mainly. We can apply static keyword with variables, methods, blocks and nested classes. The static keyword belongs to the class than an instance of the class.

The static can be:

Variable (also known as a class variable)
Method (also known as a class method)
Block

Java static variable

If you declare any variable as static, it is known as a static variable.

The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.

Advantages of static variable
It makes your program **memory efficient** (i.e., it saves memory).

Java static variable

```
class Student{
    int rollno;
    String name;
    String college=“Marwadi University";
}
```

Java static variable

```
class Student{
  int rollno;//instance variable
  String name;
  static String college ="Marwadi University";//static variable
  //constructor
  Student(int r, String n){
  rollno = r;
  name = n;
  }
  //method to display the values
  void display (){System.out.println(rollno+" "+name+" "+college);}
}
```

Java static variable

```
//Test class to show the values of objects
public class TestStaticVariable1{
 public static void main(String args[]){
 Student s1 = new Student(111,"Karan");
 Student s2 = new Student(222,"Aryan");
 //we can change the college of all objects by the single line of code
 //Student.college="BBDIT";
 s1.display();
 s2.display();
 }
}
```

Java static method

If you apply static keyword with any method, it is known as static method.
A static method belongs to the class rather than the object of a class.
A static method can be invoked without the need for creating an instance of a class.
A static method can access static data member and can change the value of it.

Java static method

```
class Student{
    int rollno;
    String name;
    static String college = "ITS";
    //static method to change the value of static variable
    static void change(){
    college = "BBDIT";
    }
    //constructor to initialize the variable
    Student(int r, String n){
    rollno = r;
    name = n;
    }
    //method to display values
    void display(){System.out.println(rollno+" "+name+" "+college);}
}
```

Java static method

```
//Test class to create and display the values of object
public class TestStaticMethod{
    public static void main(String args[]){
    Student.change();//calling change method
    //creating objects
    Student s1 = new Student(111,"Karan");
    Student s2 = new Student(222,"Aryan");
    Student s3 = new Student(333,"Sonoo");
    //calling display method
    s1.display();
    s2.display();
    s3.display();
    }
}
```

Java static block/Class Initializer Block

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

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

Java static block/Class Initializer Block

Just like we have constructor for initializing the instance variables, we have the class initializer block to initialize the class variables. A class initializer block is created just like the initializer block, but it is declared to be static. We also call this block as the static block.

```
class ClassInitializerEx
{
   static int[] arrVal = new int[10];
   // Class Initialization block
   static
   {
      System.out.println("In initialization block\n");
      for(int i=0; i<arrVal.length; i++)
      {
         arrVal[i] = (int)(100.0*Math.random());
      }
   }
```

Java static block/Class Initializer Block

```
void Display()
   {
      System.out.println("Start display\n");
      for(int i=0; i<arrVal.length; i++)
      {
         System.out.println("  " + arrVal[i]);
      }
      System.out.println("End display\n");
   }

   public static void main(String[] args)
   {
      ClassInitializerEx ib1 = new ClassInitializerEx();
      System.out.println("First object is created");
      ib1.Display();

      ClassInitializerEx ib2 = new ClassInitializerEx();
      System.out.println("Second object is created");
      ib2.Display();
   }
}
```

JVM

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. JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, Java is platform independent.

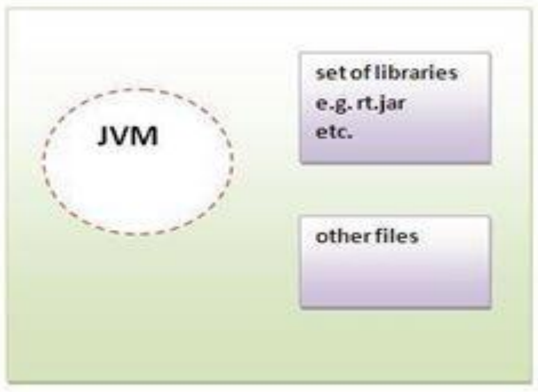
The JVM performs following main tasks:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JRE

JRE is an acronym for Java Runtime Environment.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.It physically exists.It contains JRE + development tools.

What does *Java Development Kit (JDK)* mean?
The Java Development Kit (JDK) is a software development environment used for developing Java applications and applets. It includes the Java Runtime Environment (JRE), an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (javadoc) and other tools needed in Java development.

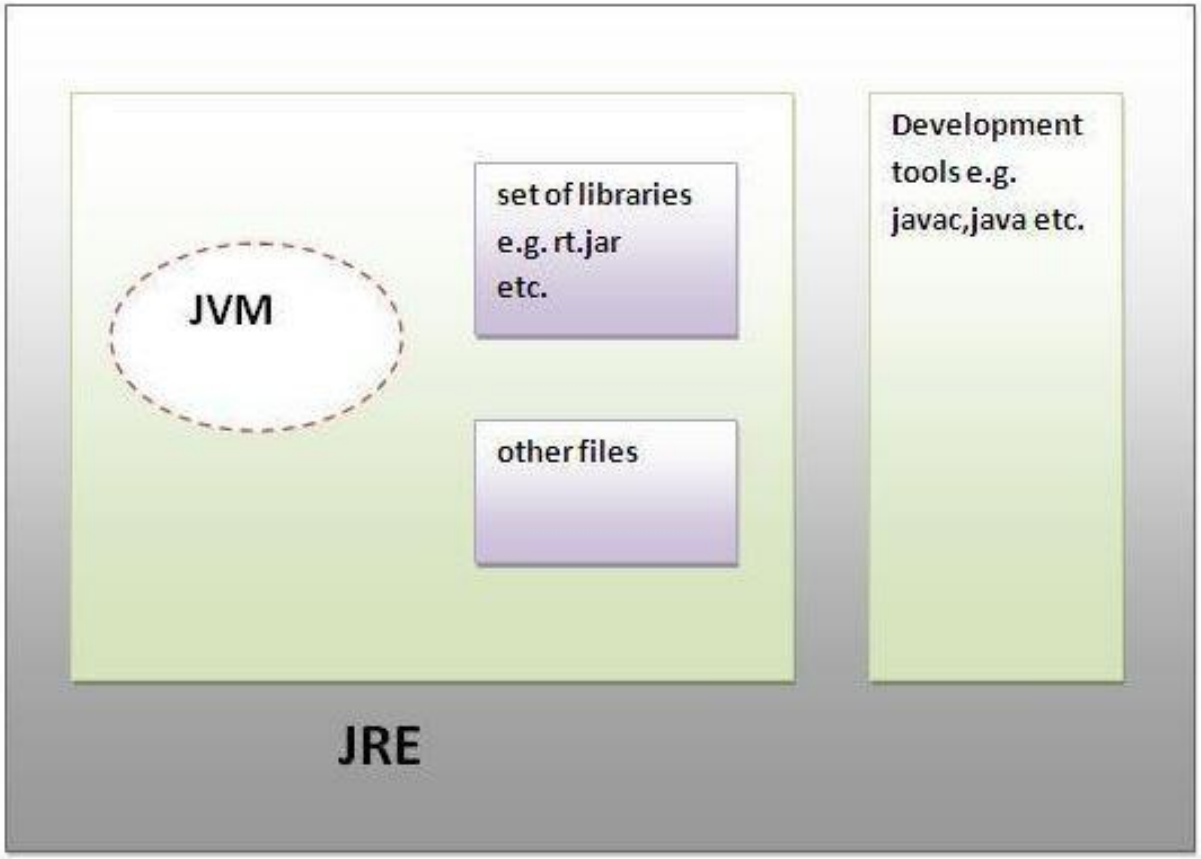
JDK

Java developers are initially presented with two JDK tools, java and javac. Both are run from the command prompt. Java source files are simple text files saved with an extension of .java. After writing and saving Java source code, the javac compiler is invoked to create .class files. Once the .class files are created, the 'java' command can be used to run the java program.

For developers who wish to work in an integrated development environment (IDE), a JDK bundled with Netbeans can be downloaded from the Oracle website. Such IDEs speed up the development process by introducing point-and-click and drag-and-drop features for creating an application.

There are different JDKs for various platforms. The supported platforms include Windows, Linux and Solaris. Mac users need a different software development kit, which includes adaptations of some tools found in the JDK.

JDK


JVM
set of libraries e.g. rt.jar etc.
other files
Development tools e.g. javac,java etc.
JRE

JDK

Thank You