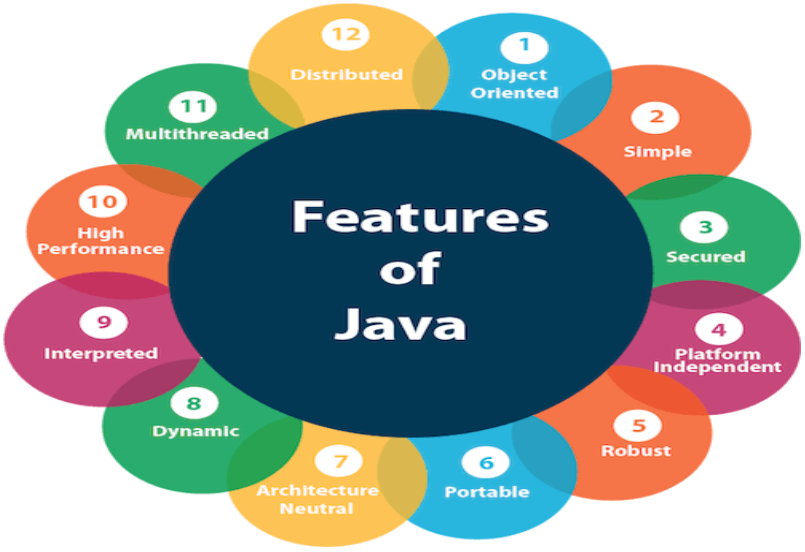
CHAPTER-1
INTRODUCTION TO JAVA LANUGAGE

Topic	Details
History, Introduction and Language Basics	• History and features of java • Java editions • JDK,JVM,JRE • JDK Tools • Compiling and executing basic java program • Datatypes • Java Tokens • Operators • Type casting • Decision Statements • Looping Statements • Jumping Statements • Array • Command Line Argument Array
classes and objects	• OOP Concept (class,objects,encapsulation,inheritance,polymorphism) • Creating and using class with members • Constructor • Finalize() method • Static and non-static members • Overloading(constructor & method) • VarArgs

History, Introduction and Language Basics (part – 1)

Java features. OR History of java



- Java is a programming language that:
  - ✓ Is exclusively **object oriented**
  - ✓ Has full **GUI support**
  - ✓ Has full **network support**
  - ✓ Is **platform independent**
  - ✓ Executes **stand-alone or "on-demand" in web browser as applets**

Features of JAVA

1. **Simple**
   - ❖ Java was designed to be **easy for the professional programmer to learn and use effectively**.
   - ❖ Assuming that you have some programming experience, you will not find java hard to master.
   - ❖ If you already understand the basic concepts of object oriented programming, learning java will be even easier.
   - ❖ If you are an experienced C++ programmer, moving to java will require very little effort. Because **java inherits the C/C++ syntax and many of the object oriented features of C++, most programmers have little trouble learning Java.**

2. **Security**
   - Security is the benefit of java. **Java system not only verifies all memory access but also ensure that no viruses are communicated with an applet.**

3. **Portable**
   - The most significant contribution of java over other language is its portability.
   - Java programs can be easily moved from one computer system to another.
   - Java ensures portability in two ways:
     1. Java compiler **generates byte code instruction that can be implemented on any machine.**
     2. The size of the primitive data types is **machine-independent**.

4. **Object-Oriented**
   - Java is a true object oriented language. All **program code and data reside within object and classes**. The object model in java is simple and easy to extend.

5. **Robust(healthy, strong)**
   - The multiplatform environment of the Web places extraordinary demands on a program, because the program must execute reliably in a variety of systems. Thus, the ability to create robust programs was given a high priority in the design of java.
   - To gain reliability, **java has strict compile time and run time checking for codes**.
   - To better understand how java is robust, consider two main reasons for program failure: **memory management mistakes and mishandled exceptional conditions.**

6. **Multithreaded**
   - Java was designed to meet the real-world requirement of creating interactive, networked programs.
   - To accomplish this, **java supports multithreaded programming, which allows you to write programs that do many things simultaneously.**
   - The java run-time system comes with an elegant yet sophisticated solution for **multiprocess synchronization that enables you to construct smoothly running interactive systems.**
   - Java's easy to use approach to multithreading allows you to think about the specific behavior of your program, not the multitasking subsystem.

7. **Architecture-Neutral**

- A central issue of java programmers was that code longevity and portability. One of the main problems facing programmers is that no guarantee exists that if you write a program today, it will run tomorrow- even on the same machine.
- Operating system upgrades, and changes in core system resources can all combine to make a program malfunction.
- The java designer made several hard decisions in the java language and the java virtual machine in an attempt to alter this situation. Their goal was "**write once; run anywhere, anytime, forever.**"

8. Interpreted

- Usually a computer language is either compiled or interpreted. **Java combines these approaches thus making java a two-stage system.**
- Java **compiler translates source code into byte code instructions**. Byte codes are not machine instructions and so java **interpreter generates machine code that can be directly executed by the machine that is running the java program.**
- We can thus say that java is both a compiled and an interpreted language.

9. High Performance

- Java performance is impressive for an interpreted language, mainly due to the use of intermediate byte code.

10. Distributed

- Java is designed for the distributed environment of the Internet, because it handles TCP/IP protocols.
- Java also **supports Remote Method Invocation (RMI).** This feature enables a program to invoke methods across a network.

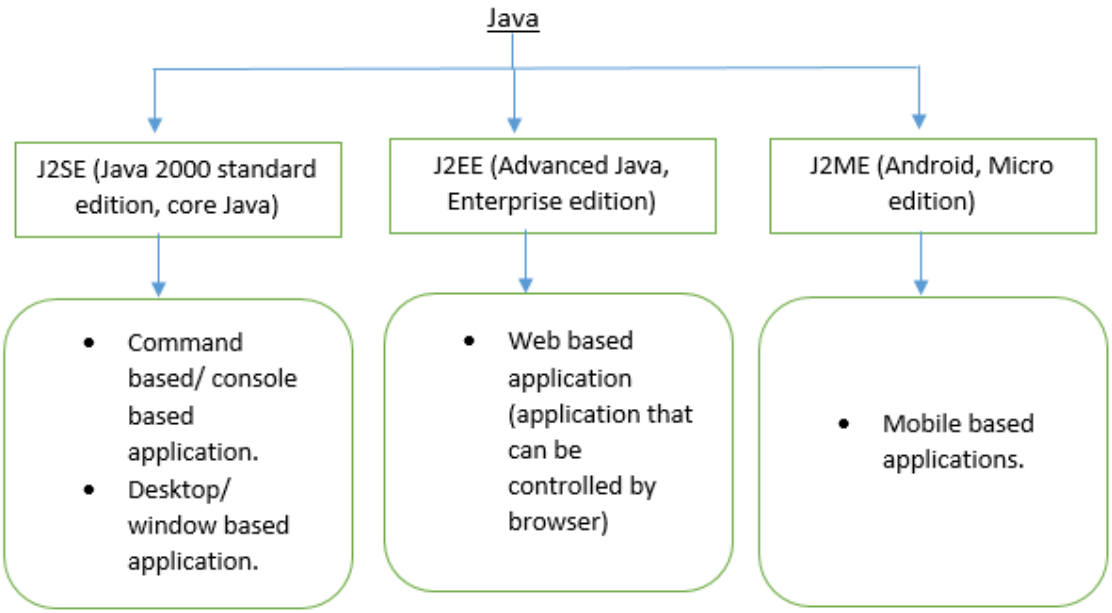
11. Dynamic

- Java is capable of dynamically linking in new **class libraries, methods and object**.
- Java can also determine the type of class through a query, making it possible to either dynamically link or abort the program.

1 WORD QUESTION ANSWER

Sr.no	Question	Answer
1	Which leads to the portability and security of java?	Bytecode is executed by JVM
2	_____ Is used to find and fix bugs in the java programs.	JDB
3	Which tool is used to generate api documentation in html format from doc comments in source code?	JAVADOC tool
4	Who is known as father of java programming language?	James Gosling
5	What is byte code in java?	Code generated by a java compiler
6	Java language was initially called as ___	Oak

Java Edition



➢ **Java Platform, Micro Edition(J2ME)**

❖ **Java ME, is designed for mobile phones (especially feature phones) and set-top boxes**. Java ME was formerly known as **Java 2 Platform, Micro Edition** (**J2ME**).

- Java ME was designed by Sun Microsystems, acquired by Oracle Corporation in 2010; the platform replaced a similar technology, Personal Java.
- The **most common of these are the Mobile Information Device Profile aimed at mobile devices**, such as cell phones, and the Personal Profile aimed at consumer products and embedded devices like set-top boxes and PDAs.

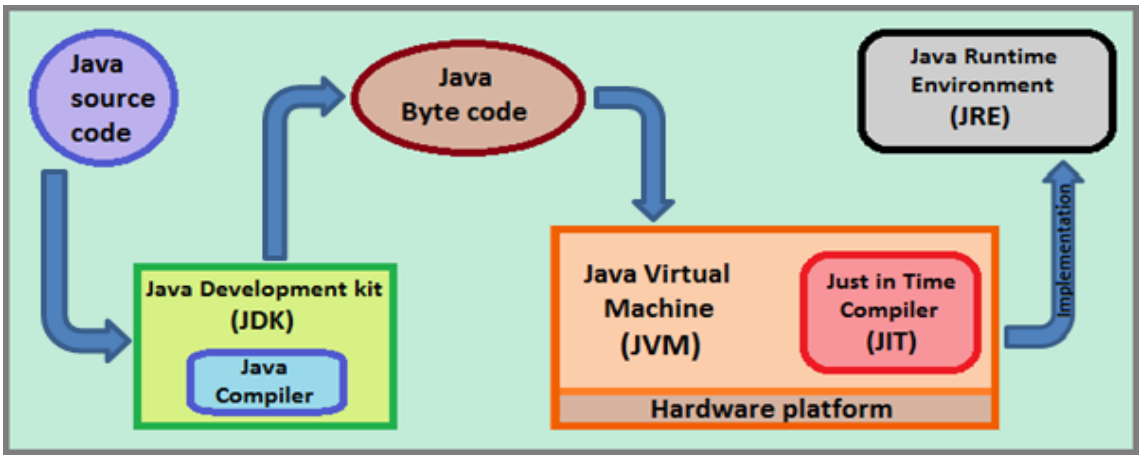
➢ **Java Platform, Enterprise Edition(J2EE)**

- **Java EE** is Oracle's enterprise Javacomputing platform.
- The platform provides an APIand runtime environment for developing and running enterprise software, including network and web services ,and other large-scale, multi-tiered, scalable, reliable, and secure network applications.
- **Java EE extends the Java Platform, Standard Edition(Java SE),**providing an API for object-relational mapping , distributed and multi-tier architectures, and web services .The platform incorporates a design based largely on modular components running on an application server.

➢ **Java Platform, Standard Edition(J2SE)**

- **Java SE** is a **widely used platform development and deployment of portable applications for desktop and server environments.**
- Java SE uses the object-oriented Java programming language. Java SE is a platform specification. It defines a wide range of general purpose APIs such as Java APIs for the Java Class Library

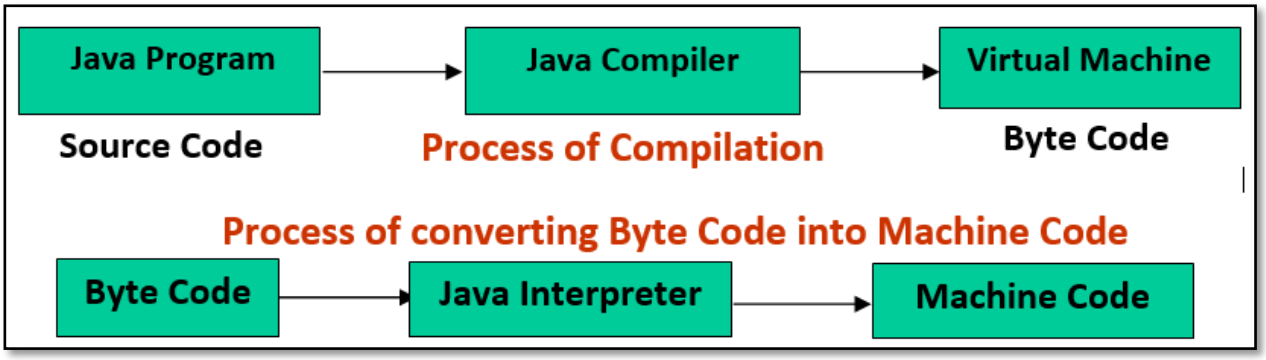
JDK, JVM, JRE



- The **Java Development Kit** (**JDK**) is an implementation of either one of the Java SE, Java EE or Java ME platforms **released by Oracle Corporation**in the form of a binary product aimed at Javadevelopers on Solaris, Linux, Mac OS or Windows.
- Since the introduction of the Javaplatform, **it has been by far the most widely used Software Development Kit (SDK) On 17 November 2006**, Sun announced that it would be released under the GNU **General Public License (GPL),** thus making it free software.
- This happened in large part on 8 May 2007, when Sun contributed the source code to the OpenJDK.

JVM

- JVM stands for **Java Virtual Machine.**
- All language compilers translate source code into machine code for a specific computer. Java compiler also does the same thing.
- Java compiler produces an intermediate code known as byte code for a machine that does not exist.
- **This machine is called the Java Virtual Machine** and it exists only inside the computer memory.



- The virtual machine code (Byte Code) is not machine specific.
- The machine specific code is generated by the Java Interpreter by acting as an intermediary between the virtual machine and the real machine as shown in fig. Interpreter is different for different machine.

JRE

- The Java Runtime Environment (JRE), also known as Java Runtime, is **part of the Java Development Kit (JDK)**, a set of programming tools for developing Java applications.
- The Java Runtime Environment provides the minimum requirements for executing a Java application; it consists of the Java Virtual Machine (JVM), core classes, and supporting files.

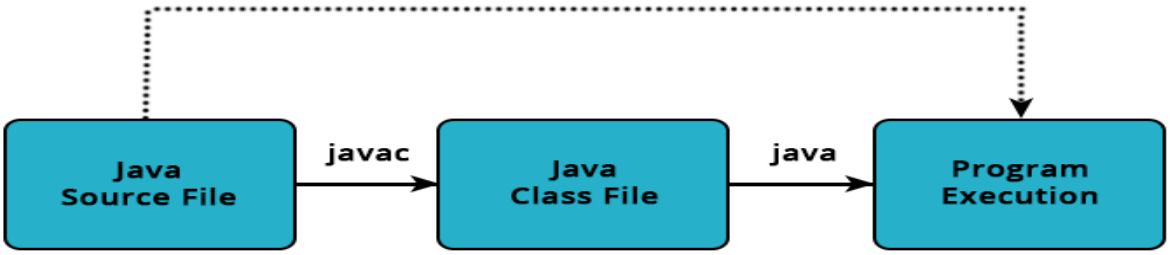
➢ JDK and its components.

- ❖ JDK stands for **Java Development Kit**.
- ❖ It is a collection of tools which are used for developing and running the java program.

➢ **Components:-**

1) **appletviewer** :- it used to view the java applets without using the browser.
2) **javac** :- the java compiler translate source code to byte code.
3) **java** :- java interpreter runs the applets and application by reading the byte code.
4) **javadoc** :- it creates html format documentation from java source code.
5) **javah** :- it produce header file.
6) **javap** :- it enables to convert byte code into program description.
7) **jdb** :- it is java debugger used for find errors from the programs.

Basic Java Program



HelloWorld.java - the source code for the "Hello, world!" program

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

- **How to save**
  - To run this program, save it in a file with the name HelloWorld.java. It must be sure that the file name must match the name of the class.
- **Compile the program**
  - **javac HelloWorld.java** command is used to compile the source code.
    When you compile the program you'll create a byte-code file named HelloWorld.class.
    You can confirm this with the dir command in the DOS/Windows world.

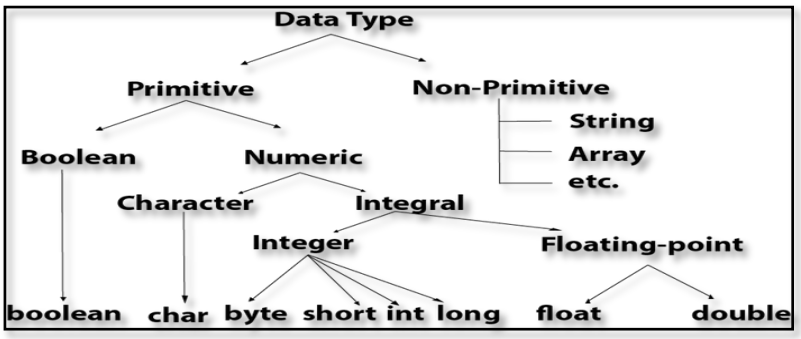
- **Execute the byte code**
  - Now you can execute the byte code in the Java interpreter with this command:
    java HelloWorld
- **Output of the program**
  - When you run the program at the command line, you'll see this output
    Hello, world!
- **Understanding the HelloWorld.java code**
  - Let's examine the HelloWorld.java file. Class is the basic building block of the java program, java codes are written in the java class.

1 WORD QUESTION ANSWER

no	Question	Answer
1	Which component is used to compile, debug and execute java program?	JDK
2	Which component is responsible for converting bytecode into machine specific code?	JVM
3	Which component is responsible to run java program?	JRE
4	Which component is responsible to optimize bytecode to machine code?	JIT
5	What is the extension of java code files?	.java
6	What is the extension of compiled java classes?	.class
7	How can we identify whether a compilation unit is class or interface from a .class file?	Java source file header
8	What is use of interpreter?	They read high level code and execute them

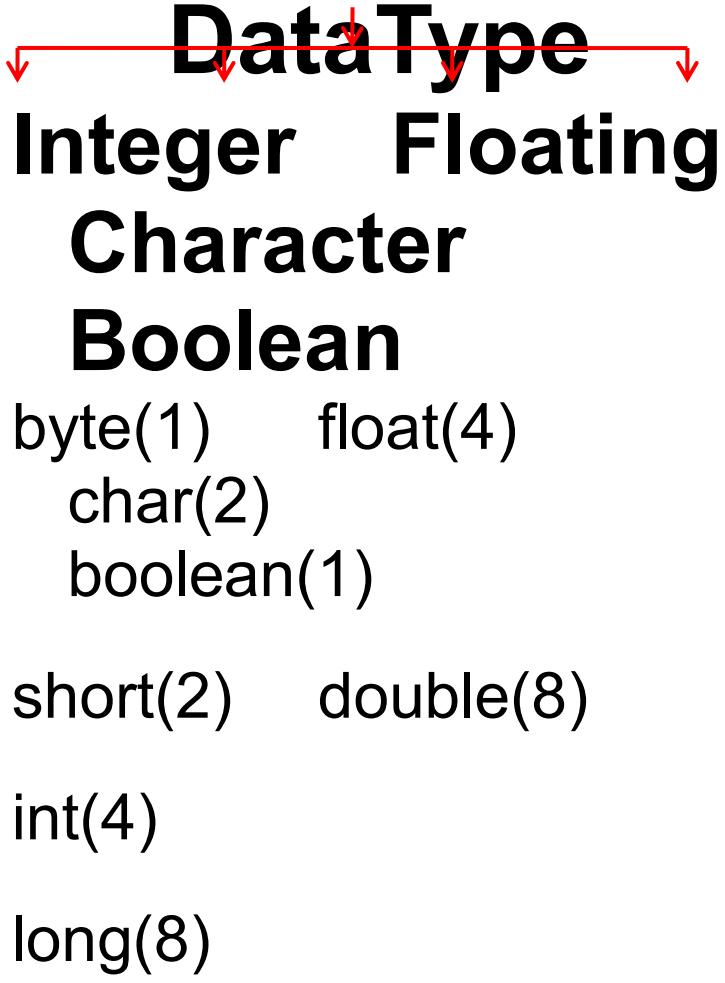
Data Type

- Every variable has a type, every expression has a type, and every type is strictly defined.
- All assignments, whether explicit or via parameter passing in method calls, are checked for type compatibility.
- There are no automatic coercions or conversions of conflicting types as in some languages.
- The java compiler checks all expressions and parameters to ensure that the types are compatible.



- **There are two types of data types**
  1. **Primitive types**
  2. **Non primitive types**

1. **Primitive Types**
   - **Java provides eight *primitive* types of data:**



- The primitive types are also commonly referred to as primary/simple types.
- **These can be put in four groups:**
  1. Integer
  2. Floating-point numbers
  3. Characters
  4. Boolean

1. Integer

- Java provides four integer types: byte, short, int, long.
- All of these are signed, positive and negative values.

Type	Size/bits	Range
byte	8	-128 to 127

short	16	-32,768 to 32,767
int	32	-2,147,483,648 to 2,147,483,647
long	64	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

2. Floating-Point Types

- Floating-Point numbers, also known as real numbers.
- There are two kinds of floating-point types, float and double, which represent single and double-precision numbers, respectively.

Type	Size/bytes	Range
Float	32	1.4e – 045 to 3.4e + 038
Double	64	4.9e - 324 to 1.8e + 308

3. Character

- The data type used to store characters is char.

Type	Size/bytes	Range
Char	16	0 to 65,536

4. Boolean

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

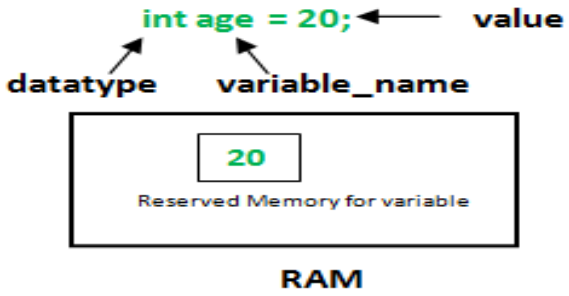
Type	Size/bytes	Range
Boolean	1	True/False , Yes/No , 0/1

1 WORD QUESTION ANSWER

no.	Question	Answer
1	What is the order of variables in enum?	Ascending order
2	Can we create an instance of enum outside of enum itself? (true or false)	False
3	What is the range of short data type in java?	-32768 to 32767
4	What is the range of byte data type in java?	-128 to 127
5	Which data type value is returned by all transcendental math functions?	Double
6	What is the numerical range of a char data type in java?	0 to 65535
7	Which coding types is used for data type characters in java?	Unicode

8	Which value can a boolean variable contain?	True & false
9	What is a valid declaration of a boolean?	Boolean b3 = false;

Variable



- The variable is **the basic unit of storage** in a java program.
- A variable is **defined by the combination of identifiers, a type and an optional initialize.**
- All variables have a scope, which defines their visibility and a life time.

✓ **Declaring a Variable: -**

- All variables must be declared before they can be used. The basic form of a variable declaration is shown here.

```
Type identifier [= value] [, identifier [=value]...];

int a,b,c;      // declare 3 integers
byte z = 22;  // initialize z
char x = 'X';// the variable x has the value 'X'
```

✓ **Dynamic Initialization:**

- Java allows variables to be initialized dynamically using any valid expression at the time the variable is declared.

Example:

```
class  DynamicInt
{         public static void main(String args[])
          {       double a= 3.0, b= 5.0;
                  // c is dynamically initialized
                  double c = Math.sqrt (a * a+b * b);
                  System.out.println("The value of C is: - " + c);
          }
}
```

In above example method **sqrt ()**, is the member of the **Math** class.

- ✓ **The Scope & Lifetime of Variable**
  - ❖ All the variables used have been declared at the start of the main( ) method. Java allows variables to be declared within any block.
  - ❖ A block defines a scope. Thus, each time you start a new block, you are creating a new scope.
  - ❖ A scope determines what objects are visible to other parts of your program. It also determines the lifetime of those objects.
  - ❖ Many other computer languages define two general categories of scopes: Global and Local. In java, the two major scopes are those defined by a class and those defined by method.
  - ❖ As a general rule, **variables declared inside a scope are not visible (accessible) to code that is defined outside that scope.**
  - ❖ Thus, when you declare a variable within a scope you are localizing that variable and the scope rules provide the foundation for encapsulation.
  - ❖ Scopes can be nested. For example, each time you create a block of code, you are creating a new rested scope. When this occurs, the outer scope encloses the inner scope. This means that objects declared in the outer scope will be visible to code within the inner scope. However, the reverse is not true. Objects declared within the inner scope will not be visible outside it.
  - ❖ To understand the effect of nested scopes consider the following program.

```
class Scope
{    public static void main(String args[])
     {        int x;
              x = 10;   // known to all code within main
              if (x == 10)
              {        int y =20;  // known to only this block
                       system.out.println ("x and y: - "+ x+" "+y);
                       x = y*2;
              }
              y = 100;  // error y is not known here
              system.out.println ("x is "+x);
     }
}
```

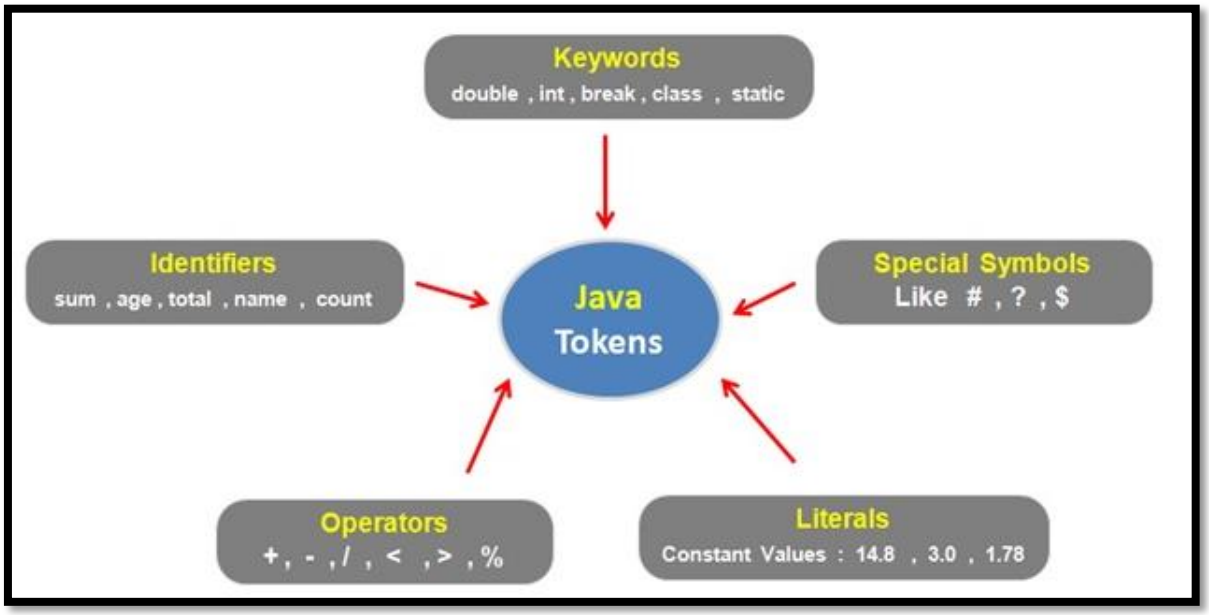
  - ❖ Variables are created when their scope is entered and destroyed when their scope is left.
  - ❖ This means that a variable will not hold its value once it has gone out of scope.

- Therefore, **variables declared within a method will not hold their values between calls to that method.**
- A variable **declared within a block will lose its value when the block is left**. Thus, the lifetime of a variable is confined to its scope.
- If a variable declaration includes an initializer then that variable will be reinitialized each time the block in which it is declared is entered

1 WORD QUESTION ANSWER

Sr.no	Question	Answer
1	Which of these cannot be used for a variable name in java?	Keyword

Tokens



- A java Program is made up of Classes and Methods and in the Methods are the Container of the various Statements And a **Statement is made up of Variables, Constants, operators etc .**

- Tokens are the various Java program elements which are identified by the compiler. A token is **the smallest element of a program that is meaningful to the compiler**. Tokens supported in Java **include keywords, variables, constants, special characters, operations** etc.

- When you compile a program, the compiler scans the text in your source code and extracts individual tokens. While tokenizing the source file, the compiler recognizes and subsequently removes whitespaces (spaces, tabs, newline and form feeds) and the text enclosed within comments. Now let us consider a program

```
//Print Hello
Public class Hello
{
Public static void main(String args[])
{
System.out.println("Hello Java");
}
}
```

- The source code contains tokens such as public, class, Hello, {, public, static, void, main, (, String, [], args, {, System, out, println, (, "Hello Java", }, }.
- The resulting tokens· are compiled into Java bytecodes that is capable of being run from within an interpreted java environment. Token are useful for compiler to detect errors. When tokens are not arranged in a particular sequence, the compiler generates an error message.
- Tokens are the smallest unit of Program

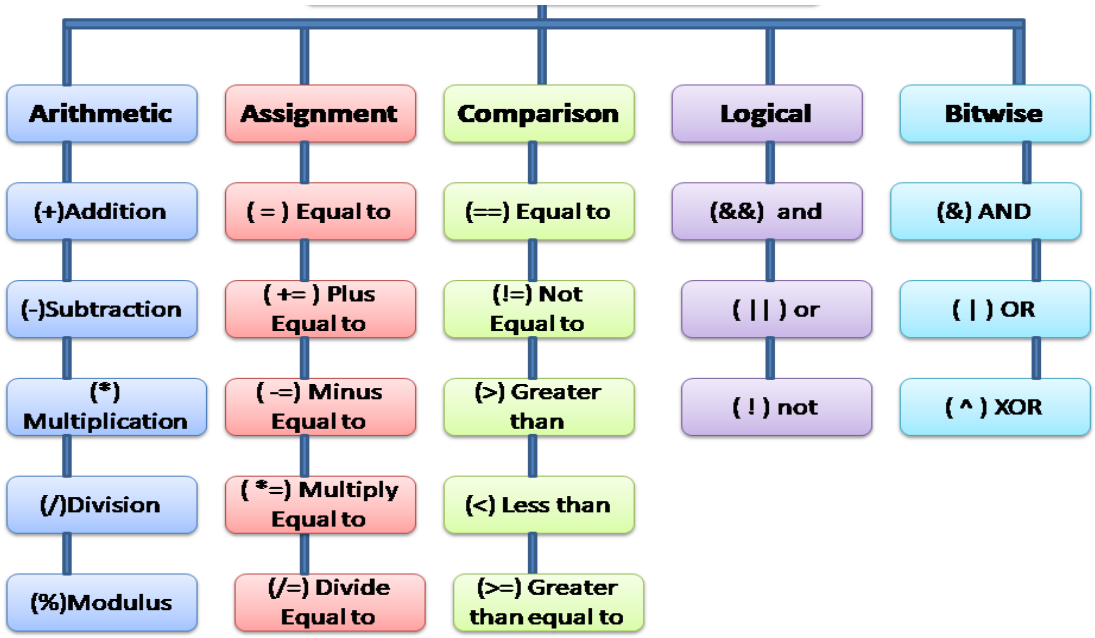
  There is Five Types of Tokens
  (1) Reserve Word or Keywords
  (2) Identifier
  (3) Literals
  (4) Operators
  (5) Separators

1 WORD QUESTION – ANSWER

NO.	QUESTION	ANSWER
1	Smallest individual units in c language is called?	Token
2	Tokens must be recognized by _____	Compiler
3	Keyword means	Reserve words
4	Can we use keyword as variable name?(yes/no)	No
5	Constant means	Fixed value
6	Value of constant can never change during execution of program?(true/false)	True
7	Types of constant	2(two) literal and symbolic

8	Is following declaration true in case of literal constant? Ex :- int i=90;	True

Operators in JAVA



- An operator is a symbol that tells the computer to perform certain mathematical or logical calculations.
- Operators are used in programs to manipulate data and variables.
- Java operators can be classified into a number of related categories as below:
  1. Arithmetic operator
  2. Relational operator
  3. Logical operator
  4. Assignment operators
  5. Increment and decrement operator
  6. Conditional operators
  7. Bitwise operators
  8. Special operators

1. Arithmetic operator

- Java provides all the basic arithmetic operators.

Operator	Meaning	Example
+	Addition or unary plus	a+b=18
-	Subtraction or unary minus	a-b=10
*	Multiplication	a*b=56
/	Division	a/b=3 (decimal part truncated)
%	Modulo division	a%b=2 (remainder of integer)

2. Relational Operator

- For comparing two quantities, and depending on their relation, we take certain decision.
- For example, we may compare the age of two persons, or the price of two items, and so on. These comparison can be done with the help of relational operators.

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

3. Logical Operator

- Java has three logical operators:

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

4. Assignment Operator

- Assignment operators are used to assign the value of an expression to a variable.
- Java has a set of 'shorthand' assignment operators which are used in the form

- For the statement:
  - ✓ x=x+(y+1);
  - ✓ x+=y+1;

Simple assignment operators	**Shorthand operator**
a=a+1	a+=1
a=a*1	a*=1
a=a*(n+1)	a*=n+1
a=a/(n+1)	a/=n+1
a=a%b	a%=b

5. Increment and Decrement Operator

- java has two increment and decrement operators: ++ and - -
- The operator ++ adds 1 to the operand while - - subtracts 1.
- both are used in the following format:
  - ✓ ++m; or m++;
  - ✓ --m or m--;
- Where:
  - ✓ ++m is equivalent to m=m+1;
  - ✓ -- m is equivalent to m=m-1;

6. Conditional Operator

- The character pair ? : is a ternary operator available in java. This operator is used to construct conditional expressions of the form
  Exp1 ? Exp2: Exp3
- Consider the following example:

```
a=10;
b=15;
x=(a>b) ? a: b;
It is same as:
    if (a>b)
        x=a;
    else
        x=b;
```

7. Bitwise Operators

- Java has a distinction of supporting special operators known as bitwise operators for manipulation of data at values of bit level.
- These operators are used for testing the bits, or shifting them to the right or left.
- Bitwise operators may not be applied to float or double.

Operator	Meaning
&	bitwise AND
!	Bitwise OR
^	Bitwise exclusive OR
~	one's complement
<<	shift left
>>	shift right

8. Special Operator

- Java supports some special operators of interest such as instanceof operator and member selection operator(.)

I. Instanceof Operator

- ✓ The instanceof is an object reference operator and returns true if the object on the left-hand side is an instance of the class given on the right-hand side. This operator allows us to determine whether the object belongs to a particular class or not.
- ✓ For example:
- ✓ **person instanceof student** is true if the object person belongs to the class student; otherwise it is false.

II. Dot operator

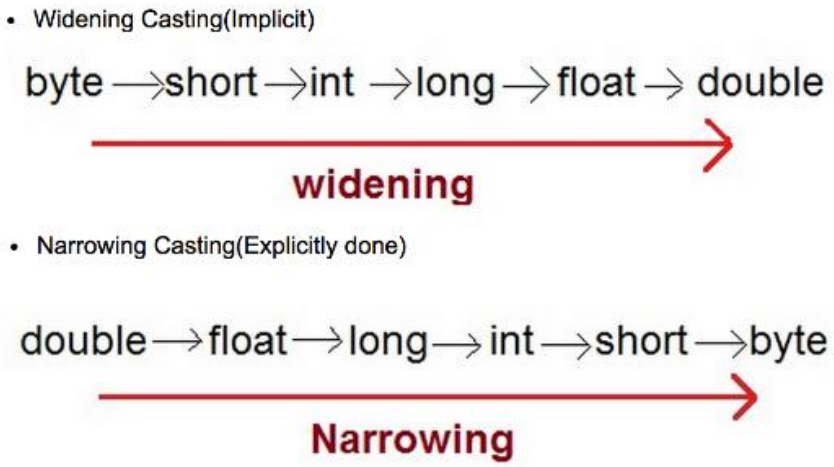
- ✓ The dot operator(.) is used to access the instance variables and methods of class objects.
- ✓ **For example:**
- ✓ person1.age //reference to the variable age
- ✓ person1.salary() //reference to the method salary()
- ✓ It is used to access classes and sub-packages form a package.

1 WORD QUESTION – ANSWER

Sr.no.	Question	Answer

1	Conditional operator in c language can be ____ and ____	? And :
2	_____ Operator can find how many bytes an operand can occupies	Sizeof
3	_____ Operator can used to link related expression together.	Comma
4	_____ Operator is known as assignment operator	=(Equal to)
5	If condition is false , logical operator can return _____	0
6	Precedence of operator is known as _____	Hierarchy

Type Casting



Widening Casting (Implicit)

- ✓ Widening casting is done automatically when passing a smaller size type to a larger size type:

Example

```
public class Test
{
        public static void main(String[] args)
        {
                int myInt = 9;
                double myDouble = myInt; // Automatic casting: int to double
                System.out.println(myInt);      // Outputs 9
                System.out.println(myDouble);   // Outputs 9.0
         }
}
```

Narrowing Casting (Explicit)

- ✓ Narrowing casting must be done manually by placing the type in parentheses in front of the value:

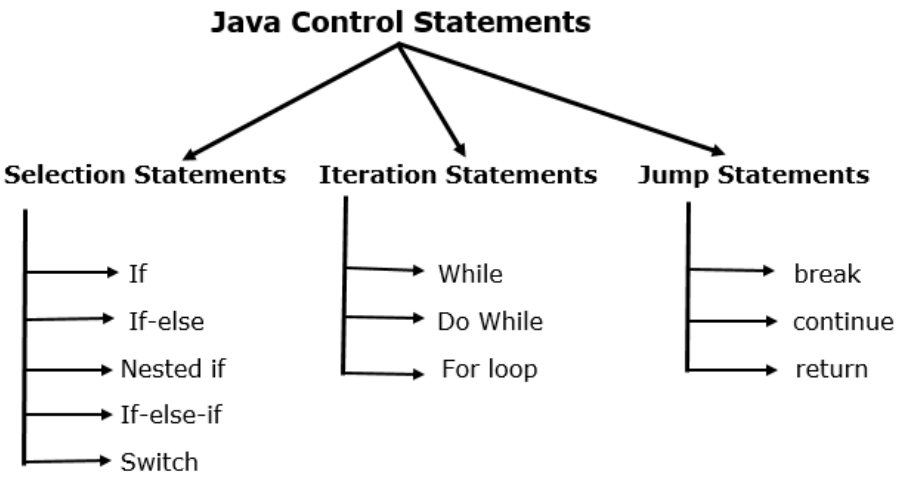
Example

```
public class Main
{           public static void main(String[] args)
        {
                double myDouble = 9.78;
                int myInt = (int) myDouble; // Manual casting: double to int
                System.out.println(myDouble);   // Outputs 9.78
                System.out.println(myInt);      // Outputs 9
        }
}
```

1 WORD QUESTION – ANSWER

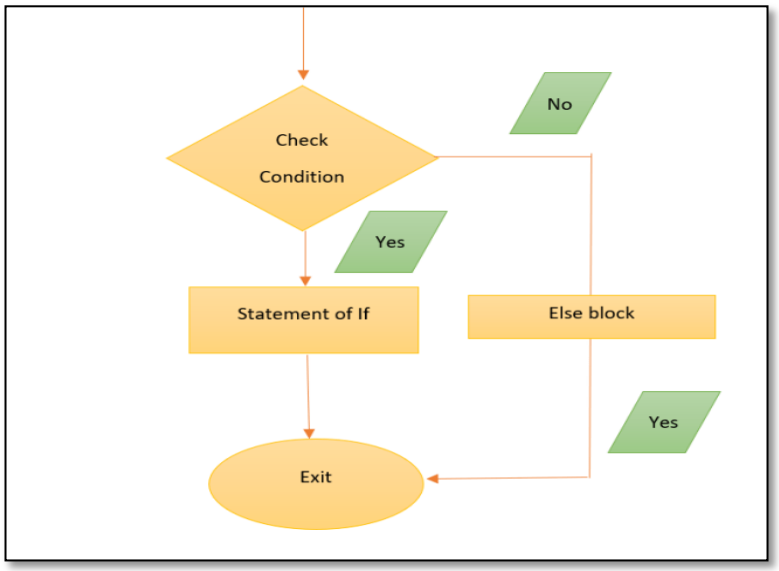
Sr.no.	Question	Answer
1	Type casting also known as _____	Type conversion
2	Type casting can be of _____ & _____ types	Implicit & explicit
3	Implicit conversion also known as _____	Automatic conversion
4	In explicit conversion , we have to use _____ operator to perform type casting	Type cast

Control statements in java
Decision Statement (Selection Statements)



- There are two types of decision making statements in Java. They are:
  - if statements
  - switch statements

1. The if Statement:



- An if statement consists of a Boolean expression followed by one or more statements.
- **Syntax:**

The syntax of an 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.

Example:

```
public class Test
{       public static void main(String args[])
        {   int x =10;
            if( x <20)
            {
                System.out.print("This is if statement");     }}}
```

2. The if...else Statement:

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

Syntax:

The syntax of an if...else is:

```
if(Boolean_expression)
{
    //Executes when the Boolean expression is true
}else
{
    //Executes when the Boolean expression is false
}
```

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");
                }
        }
}
```

Output:

This is else statement

3. The if...else if...else Statement:

- An 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.
- An if can have zero or one else's and it must come after any else if's.
- An 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.

Syntax:

The syntax of an if...else is:

```
if(Boolean_expression1)
{       //Executes when the Boolean expression 1 is true
}
elseif(Boolean_expression2)
{       //Executes when the Boolean expression 2 is true
}
elseif(Boolean_expression3)
{       //Executes when the Boolean expression 3 is true
}
else
{       //Executes when the none of the above condition is true.
}
```

Example:

```
public class Test
{       public static void main(String args[])
        {
        int x =30;
        if( x ==10)
        {
                System.out.print("Value of X is 10");
```

```
        }
        elseif( x ==20)
        {
                System.out.print("Value of X is 20");
        }
        elseif( x ==30)
        {
                System.out.print("Value of X is 30");
        }
        else
        {
                System.out.print("This is else statement");
        }
        }
}
```

output:

Value of X is30

4. 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_expression1)
{
    //Executes when the Boolean expression 1 is true
    if(Boolean_expression2)
    {
    //Executes when the Boolean expression 2 is true
    }
}
```

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

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");
                    }
            }
    }
}
```

Output:

X =30and Y =10

5. 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:

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

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.**

Example:

```
public class Test
{
 public static void main(String  args[])
 {     //char grade = args[0].charAt(0);
       char grade ='C';
       switch(grade)
       {case 'A':
              System.out.println("Excellent!");
              break;
       case 'B':
       case 'C':
              System.out.println("Well- done");
              break;
       case 'D':
              System.out.println("You passed");
       case 'F':
              System.out.println("Better try again");
              break;
       default:
              System.out.println("Invalid grade");
       }
       System.out.println("Your grade is "+ grade);
 }
}
```

1 WORD QUESTION – ANSWER

Sr.no	Question	Answer
1	How many control structures available in Java ? Give name.	(Two) If statement Switch statement
2	If any condition become false ,statement following _____ will be execute.	Else
3	In nested if first of all _____ condition will be checked.	Outer
4	How mamy flavors/types of if statement.give name.	4(four) Simple if If….else If…else..if…else Nested if
5	Which indicate easy way to represent multiple conditions at the same time?	Else…if
6	Switch represent _____	Multiway decision statement
7	If no any case value match with condition then statement following _____ will execute.	Default
8	_____ Statement is used to terminate particular case.	: (Colon)
9	Write down syntax to represent switch statement	Switch(expression)

Looping statements

for Loop	while Loop	do-while Loop
• If number of iteration is fixed, this loop is recommended.	• If number of iteration is not fixed, this loop is recommended	• It is used, If number of iteration is not fixed and you must have to execute the loop at least once.

- 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:
  - while Loop
  - do...while Loop
  - for Loop

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

1. The while Loop:

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

Syntax:

```
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.

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. The 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:

```
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.
- **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);
        }
}
```

- **Output:**

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. The 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:

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

- Here is the flow of control in a for loop:
- The initialization step is executed first, and only once.
- 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.

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");
                }
        }
}
```

- **Output:**

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

4. Enhanced for loop in Java:

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

Syntax:

```
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.

Example:

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

```
                for(int x : numbers )
                {       System.out.print( x );
                        System.out.print(",");
                }
                System.out.print("\n");
                String[] names ={"James","Larry","Tom","Lacy"};
                for(String name : names )
                {       System.out.print( name );
                        System.out.print(",");
                }
        }
}
```

- **Output:**
  10,20,30,40,50,
  James,Larry,Tom,Lacy,

1 WORD QUESTION – ANSWER

No.	Question	Answer
1.	How many loops are there in java	3
2.	Give the correct syntax of for loop?	for(initialization; condition; increment/decrement)
3.	Give the name of entry control loop	while
4.	Give the name of exit control loop	do…while

Jumping statements

- There are three jump statements in java:
  1. break
  2. continue
  3. return
- These statements transfer control to another part of your program. The break statements have three uses, they are:
- It determines a statement sequence in a switch statement.
- It can be used to exit a loop
- It is another form of goto.

1. 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:

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");
                }
        }
}
```

2. 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:

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");
                }
            }
}
```

Output:

10
20
40
50

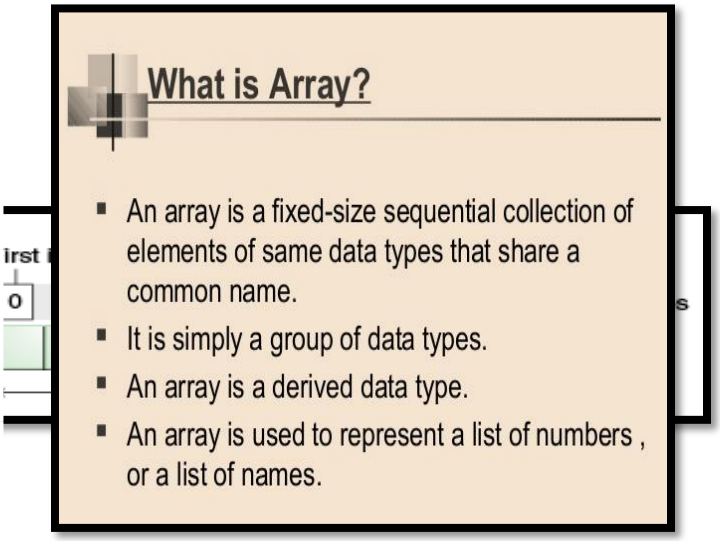
```
class BrDemoAppl
{   public static void main(String args[])
   {   for (int count = 1; count <= 100; count++)
      {   if (count == 10)
          break;
        System.out.println("The value of num is : " + count);
      }
      System.out.println("The loop is over");
   }
}
```

- The for loop is designed to run for 100 times but it determines when the value of count is equal to 10 by calling the break statement.

1 WORD QUESTION – ANSWER

Sr.no.	Question	Answer
1	How many jumping statements available in c?	3(break ,continue, goto)
2	_____ Statement is used to terminate from the loop.	Break
3	_____Statement is used to pass control to the next iteration.	Continue
4	_____ Statement is used to jump from one point to another point.	Goto

Array

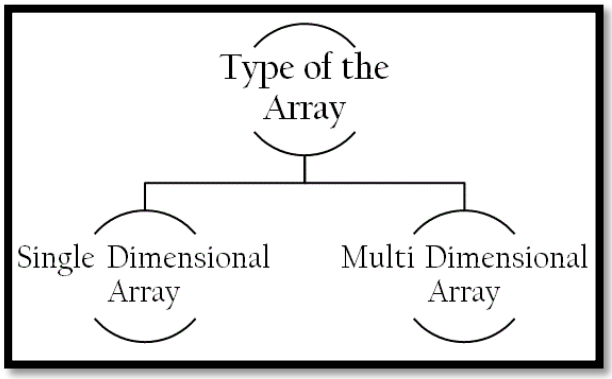


❖ An array is a group of like-typed variables that are referred by a common name.

❖ Arrays of any type can be created and may have one or more dimensions.

❖ A specific element in an array is accessed by its index.

❖ Arrays offer a convenient means of grouping related information.



1. One-Dimensional Arrays (Single-Dimensional Arrays)

- A one-dimensional array is a list of like typed variables.
- To create an array, you first must create an array variable of the desired type.
- Syntax: type var-name[ ];
- **Example:** int month_days[ ];
- The general form of new as it applies to one-dimensional arrays appears as follows:

  array-var=new type[size];

```
month_days=new int[12];
```

- after this statement executes, month_days will refer to an array of 12 integers.

- In short, obtaining an array is a two step process.
  - you must declare a variable of the desired array type.
  - you must allocate the memory that will hold the array, using new, and assign it to the array variable.

```
class Array_Ex
{   public static void main(String args[])
    {       int month_days[ ];
            month_days= new int[12];
            month_days[0]=31;
            month_days[1]=28;
            month_days[2]=31
            month_days[3]=30
            month_days[4]=31
            month_days[5]=30
            month_days[6]=31
            month_days[7]=31;
            month_days[8]=30;
            month_days[9]=31;
            month_days[10]=30;
            month_days[11]=31;
            System.out.println ("April has" + month_days[3] + "days.");
    }
}
```

- it is possible to combine the declaration of the array variable with the allocation of the array itself, as shown here:
- int month_days[ ]=new int[12];
- Arrays can be initialized when they are declared.
- An array initializer is a list of comma-separated expressions surrounded by curly braces. The comma separates the values of the array elements.
- The array will automatically be created large enough to hold the number of elements you specify in the array initializer. There is no need to use new.
- For example,

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

```
            int month_days[]={31,28,31,30,31,30,31,31,30,31,30,31};
            System.out.println("April has" + month_days[3]    + "days.");
    }
}
```

2. Multidimensional Arrays

- Multidimensional arrays are actually arrays of arrays.
- To declare a multidimensional array variable, specify each additional index using another set of square brackets.
- For example, following declares two dimensional array:
- inttwoD[ ] [ ] = new int [4] [5];
- This allocates a 4 by 5 array and assigns it to twoD. Internally this matrix is implemented as an array of arrays of int.

Alternative Array Declaration

- There is a second form that may be used to declare an array:
  type[ ] var-name;
- The square brackets follow the type specifier, and not the name of the array variable.
- For example, the following two declarations are equivalent:

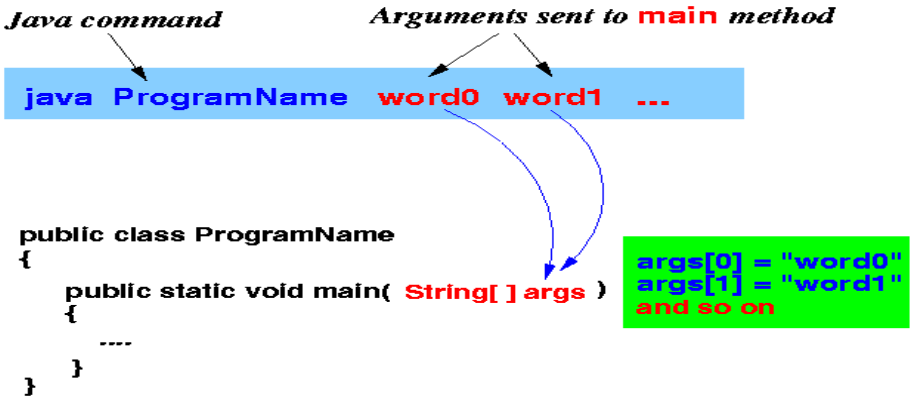
```
int a1[ ] = new int[4];
int [ ] a1= new int[4];
char twod [ ] [ ] = new char [3] [4];
char [ ] [ ] twod = new char [3] [4];
```

- This alternative declaration form offers convenience when declaring several arrays at the same time. For example,
- int [ ] nums1, nums2, nums3;
- This creates 3 array variables of int type.

1 WORD QUESTION – ANSWER

No	Question	Answer
1	What is array?	Group of elements having same name and type.
2	Array is _____ data type.	Derived
3	Array is used to represent _____	Collection
4	Types of array can be _____ & _____	Single/one dimension& multi/two dimension

Command line Arguments



- A Java application **can accept any number of arguments from the command line**. This allows the user to specify configuration information when the application is launched.
- The **user enters command-line arguments when invoking the application and specifies them after the name of the class to be run**. For example, suppose a Java application called Sort sorts lines in a file. To sort the data in a file named friends.txt, a user would enter:

  java Sort friends.txt
- When an application is launched, the runtime system passes the command-line arguments to the application's main method via an array of Strings. In the previous example, the command-line arguments passed to the Sort application in an array that contains a single String: "friends.txt".

Echoing Command-Line Arguments

```
public class Echo
{
   public static void main (String[] args)
  {     for (String s: args)
        {
              System.out.println(s);
      }
   }
}
```

The following example shows how a user might run Echo. User input is in italics.

```
java Echo Drink Hot Java
Drink
Hot
Java
```

- To have Drink, Hot, and Java interpreted as a single argument, the user would join them by enclosing them within quotation marks.

```
java Echo "Drink Hot Java"
Drink Hot Java
```

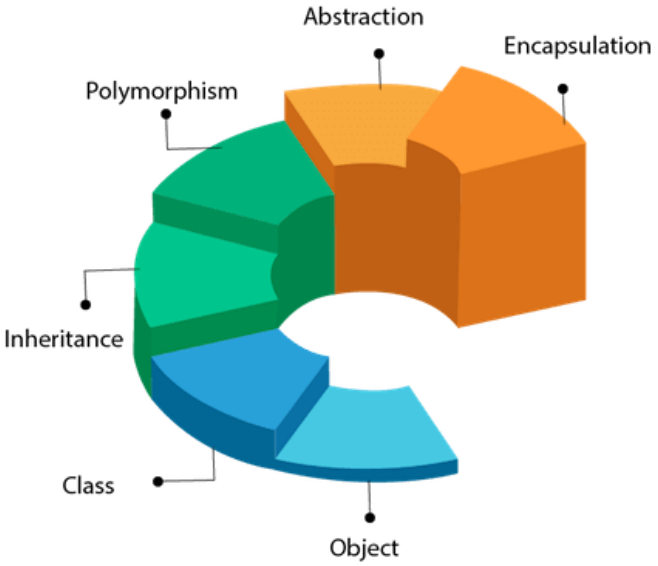
1 WORD QUESTION – ANSWER

SR. NO.	Question	Answer
1	Which method is given parameter via command line arguments?	main()
2	Which data types is used to store command line arguments?	String
3	How many arguments can be passed to main()?	Infinite
4	Which is correct statement about args in the following line of code? Public static void main(string args[])	args is an array of string
5	Can command line arguments be converted into int automatically if required? (yes / no)	No

End of the PART -1

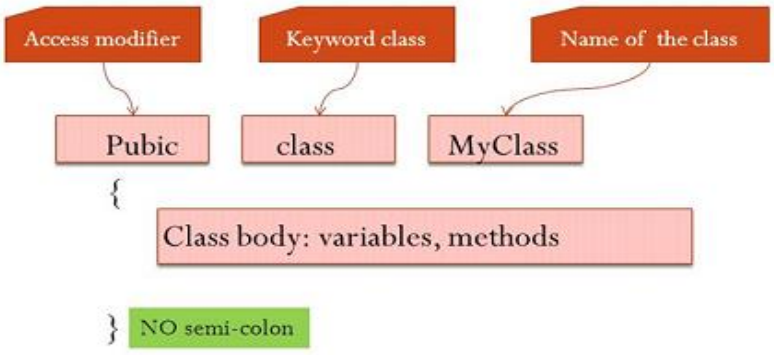
Class Fundas (Part 2)

OOPs (Object-Oriented Programming System)

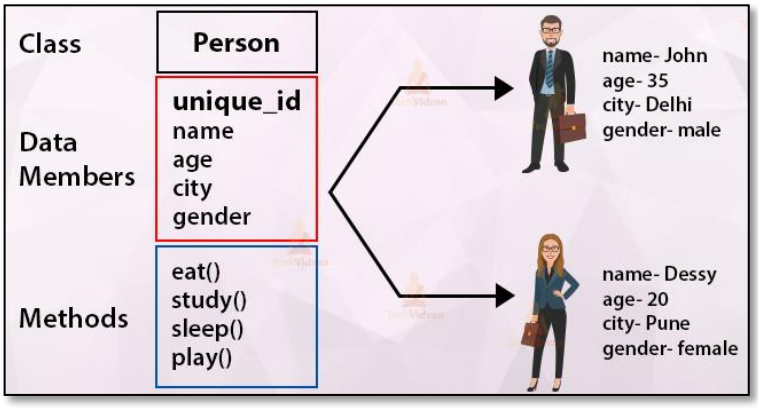


Class and Object

- **Class**
  - A class can be defined as a template that describes the behaviors /states that object of its type support.
  - Anatomy of a Java Class:



- A class is a **template from which objects are created**. That is objects are instance of a class.
- **When you create a class, you are creating a new data-type**. You can use this type to declare objects of that type.



- Class **defines structure and behavior (data & code)** that will be shared by a set of objects
- A class is declared by use of the **class keyword**. Classes may contain data and code both.
- The general form of al class definition:

```
class ClassName
{
        type instance variable1;
        type instance variable2;
        type methodname1 (parameter list)
        {
                body of method;
        }
        type methodname2 (parameter list)
        {
                body of method;
        }
}
```

- The data or variable are called **instance variable**. The code is contained within methods.
- The method and variables defined within a class are called **member of the class**.

```
class Box
{       double width;
        double height;
        double depth;
}
classBoxDemo
{       public static void main (String args[])
        {       Box mybox = new Box ();
                doublevol;
                mybox.width =10;
                mybox.height = 20;
                mybox.depth = 30;
                vol = mybox.width * mybox.height * mybox.depth;
                System.out.println ("Volume is: - "+vol);
        }
}
```

- ❖ Each object contains its own copy of each variable defined by the class.
- ❖ So, every Box object contains its own copy of the instance variables width, height and depth.
- ❖ To access these variables, you will use the **dot (.) operator**. The dot operator links the name of the object with the name of an instance variable.

- **Declaring Object**

```
Box mybox;              // declare ref. to object which contains null value.
mybox = new Box (); // allocate a Box object.
```

- **General form of a new**

```
classvar = new classname ();
mybox = new Box ();
Where:
        classvar =  variable of the class type.
        classname = name of the class.
```

- ❖ The **classname followed by parentheses** specifies the constructor for the class.
- ❖ A **constructor defines what occurs when an object of a class is created.**

- **Assigning Object Reference Variable**

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

- ❖ After this executes, b1 and b2 will both refer to the same object.
- ❖ The assignment of b1 to b2 did not allocate any memory or copy any part of the original object.
- ❖ It simply makes b2 refer to the same object as does b1. Thus, any changes made to the object through b2 will affect the object to which b1 is referring, since they are the same object.

1 WORD QUESTION ANSWER

NO.	Question	Answer
1.	_____ Are an expanded version of structures. Structure can contain multiple variables.	Class
2.	_____ Is used to access class members.	Object
3.	_____ Access the properties and features of one class into another class.	Inheritance
4.	_____ More than one function with same name, with different working. It can be static or dynamic.	Polymorphism
5.	_____Is to visible only the necessary information, unnecessary information will be hidden from the outside world.	Data abstraction
6.	_____ Is a process of wrapping data members and member functions in a single unit called class	Encapsulation

Creating and using class with members OR Introducing method

Methods in Java

```
type name (parameter-list)
{
        body of method
}
```

Where:

Type : Specifies the type of data returned by the method. If the method does not return a value its return type must be void.

Name: Specifies the name of the method.

Parameter-list: It is a sequence of type & identifiers pairs separated by commas.

- ✓ Parameters are variables that receive the value of the argument passed to the method when it is called. If the method has no parameters, then the parameter list will be empty.

- ✓ Methods that have a return type other than void return a value to the calling routine using the following form of the return statement;

```
Return value;
```

- **Types of Methods**
  1. Does not return value – void
  2. Returning a value
  3. Method which takes parameter

1. **Does not return value – void**

```
class Box
{   double width, height, depth;
    void volume()
    {
            System.out.println("Volume is: -"+width*height*depth);
    }
}
class BoxDemo
{   public static void main (String args[])
    {
            Box mybox1 = new Box ();
            Box mybox2 = new Box ();

            mybox1.width =10;
            mybox1.height =20;
            mybox1.depth =15;
            mybox2.width =10;
            mybox2.height =15;
            mybox2.depth =25;
            mybox1.volume ();
            mybox2.volume ();
    }
}
```

2. Returning a value

```
class Box
{
    double width, height, depth;
    void volume()
    {
            return width*height*depth;
    }
}
class BoxDemo
{
    public static void main (String args[])
    {
            Box mybox1 = new Box ();
            Box mybox2 = new Box ();
            double vol;

            mybox1.width =10;
            mybox1.height =15;
            mybox1.depth =20;
            mybox2.width =2;
            mybox2.height =3;
            mybox2.depth =5;
            vol = mybox1.volume ();
            System.out.println ("Volume is: -"+vol);

            vol = mybox2.volume ();
            System.out.println ("Volume is: -"+vol);
    }
}
```

3. Method which takes parameter

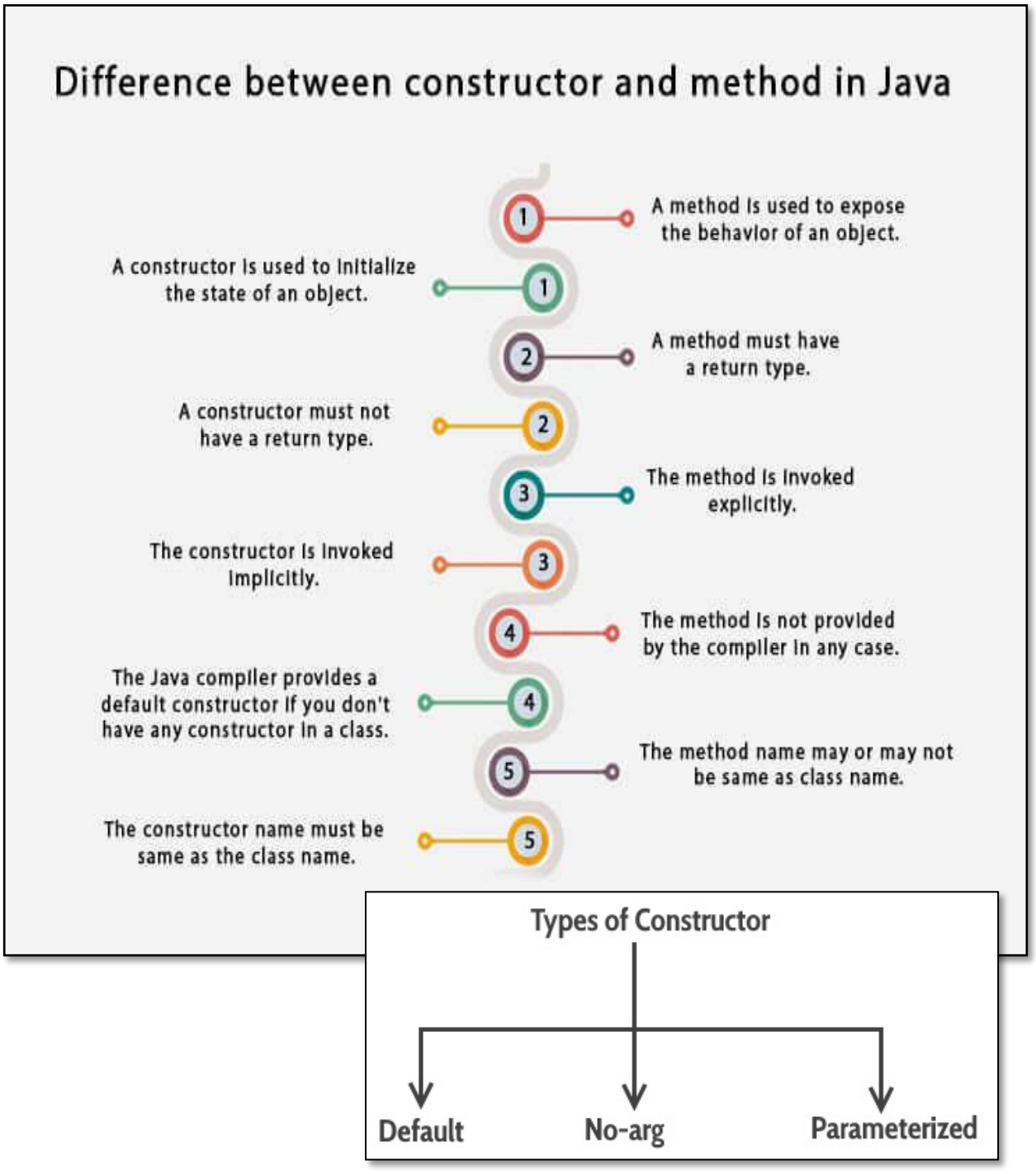
- A parameter is a variable defined by a method that receives a value when the method is called.
- An argument is value that is passed to a method when it is invoked.

```
class Box
{       double width, height, depth;
        double volume()
        {
                return width*height*depth;
        }
        void setDim (double w, double h, double d)
        {       width = w;
                height = h;
                depth = d;
        }
}
class BoxDemo
{       public static void main (String args[])
        {       double vol;
                Box mybox1 = new Box ();
                Box mybox2 = new Box ();
                mybox1.setDim (10,15,25);
                mybox2.setDim (3,5,7);
                vol = mybox1.volume ();
                System.out.println ("Volume is: -"+vol);
                vol = mybox2.volume ();
                System.out.println ("Volume is: -"+vol);
        }
}
```

1 WORD QUESTION ANSWER

NO.	Question	Answer
1.	What is the return type of a method that does not return any value?	Void
2.	What is the process of defining more than one method in a class differentiated by method signature?	Method overloading
3.	Which method having same name as that of its class?	Constructor
4.	Which method can be defined only once in a program?	main method

Constructor



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

Rules for creating java constructor

- Constructor name must be same as its class name
- Constructor must have no explicit return type not even void

Types of java constructors

- Default constructor
- no-argument constructor
- Parameterized constructor

1. **Default Constructor**
   If there is **no constructor** in a class, **compiler automatically** creates a default constructor.
2. **no-argument constructor**

```
class Box
{       double width, height, depth;
        Box( )
        {       width = 10;
                height = 10;
                depth = 10;
        }
        double volume( )
        {       return width*height*depth;
        }
}
class BoxDemo
{       public static void main (String args[])
        {       double vol;
                Box mybox1 = new Box ();
                Box mybox2 = new Box ();
                vol = mybox1.volume ();
                System.out.println ("Volume is: -"+vol);
                vol = mybox2.volume ();
                System.out.println ("Volume is: -"+vol);
        }
}
```

- mybox1 and mybox2 were initialized by the Box () constructor when they were created.
- Both will print the same value 1000.
- Constructor for the class is being called. New Box() is calling the Box() constructor.

3. Parameterized Constructor

```
class Box
{       double width, height, depth;
        Box (double w, double h, double d)
        {       width = w;
                height = h;
                depth = d;
        }
        double volume()
        {
                return width*height*depth;
        }
}
class BoxDemo
{       public static void main (String args[])
        {       double vol;
                Box mybox1 = new Box (10,20,15);
                Box mybox2 = new Box (3,5,7);
                vol = mybox1.volume ();
                System.out.println ("Volume is: -"+vol);
                vol = mybox2.volume ();
                System.out.println ("Volume is: -"+vol);
} }
```

1 WORD QUESTION ANSWER

NO.	QUESTION	ANSWER
1.	Java provides a special member function called the constructor that enables an _____to be initialized when it is created.	Object
2.	A _____is different than other member function of the class it has the same name as its class without return type even not void.	Constructor
3.	The _____of a class is the first member function to be executed automatically when object of class is created.	Constructor

4	_____ Has the same name as a class name.	Constructor
5	_____ Is executed automatically whenever the class object is created.	Constructor
6	The _____is a special member function with no arguments which initialize the data members.	Default constructor
7	The constructor with arguments is called_____.	Parameterized constructors
8	When the constructor is parameterized, must be provide appropriate _____ to the constructor.	Argument
9	When the class has multiple constructors, is called _____.	Constructor overloading
10	Each constructor should be differ from their _____of arguments and _____of arguments.	Number, types
11	When constructor has object of its own class is called_____ .	Copy constructor

Finalize() method

- Sometimes an object will need to perform **some action when it is destroyed**.
- For Ex, if an **object is holding some non-Java resource such as a file handle or window character font, then you might want to make sure these resources are freed before an object is destroyed.**
- To handle such situations, java provides a mechanism **called *finalization***.
- Finalize() is the method of **Object class**.
- This method is called just **before an object is garbage collected**.
- finalize() method overrides to dispose system resources, perform clean-up activities and minimize memory leaks.

```
Protected void finalize ()
{
        // finalize code
}
```

- Following is the example:

```
public class Test {
        int a;
  public static void main(String[] args)
  {   Test obj = new Test();
```

```
        obj.a = 10;
        System.out.println(obj.a);
        obj=null;
       System.gc(); // calling garbage collector
       System.out.println("end of garbage collection");
    }
    protected void finalize()
    {
       System.out.println("finalize method called");
    }
}
```

Output:

```
10
end of garbage collection
finalize method called
```

Static & Non-Static members

Difference Between Static and non-Static Variable in Java

Non-static variable	Static variable
• Non-static variable also known as instance variable while because memory is allocated whenever is created. • Non-static variable are specific to an object. • Non-static variable can access with object reference. • **Syntax**	• Memory is allocated at the time of loading of class so that these are also known as class variables. • Static variable are common for every object that mean these memory location can be shareable by every object reference or same class. • static variable can access with class reference. • **Syntax**
Obj_ref.variable_name	class_name.variable_name

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 can be:
  1. **Variable (also known as a class variable)**
  2. **Method (also known as a class method)**
  3. Block
  4. Nested class
- The most common example of a static member is **main()**.main() is declared as static because it must be called before any object exist.

1. 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.
- The static variable gets memory only once in the class area at the time of class loading.

Advantages of static variable

- It makes your program **memory efficient** (i.e., it saves memory).
- **Example:**

```
class Student
{       int rollno;
        String name;
        static String college =“HNS";
        Student(int r,String n)
        {       rollno = r;
                name = n;
         }
        void display ()
        {       System.out.println(rollno+" "+name+" "+college);    }
        public static void main(String args[])
        {       Student s1 = new Student(1,”Ashish");
                Student s2 = new Student(2,“Amar");
                s1.display();
                s2.display();
        }
```

```
}
```

2. **Java static method**
   - If you apply static keyword with any method, it is known as static method.
   - A static method can be invoked without the need for creating an instance of a class.
   - static method can access static data member and can change the value of it.
   - **Restrictions for the static method**
     1. The static method cannot use non static data member or call non-static method directly.
     2. **this and super** cannot be used in static context.
   - **Example:**

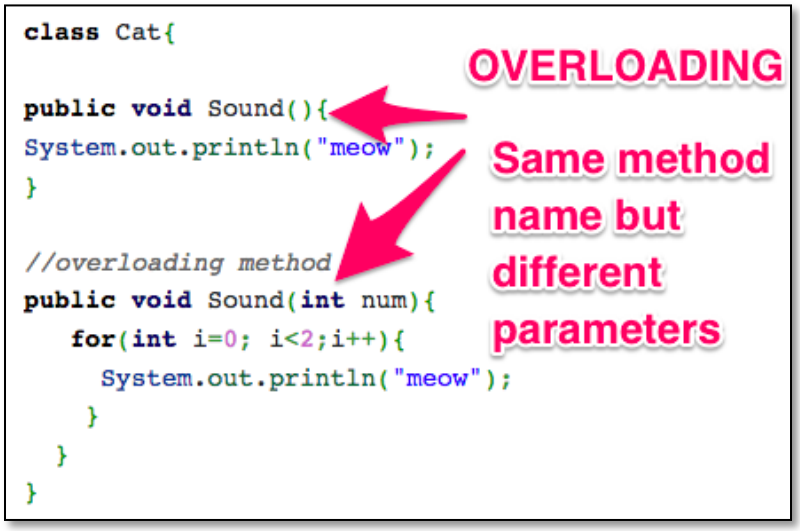
```
class Calculate
{
        static int cube(int x)
        {    return x*x*x;    }
        public static void main(String args[])
        {
                int result=Calculate.cube(5);
                System.out.println(result);
                 result=Calculate.cube(10);
                System.out.println(result);
         }
}
```

Static and Non Static members:

```
class A {
        int a=40;//non static
   public static void main(String args[])
        {  System.out.println(a);   }  X //Compile time error
}
=====
class A {
        static int a=40;//static
   public static void main(String args[])
        {  System.out.println(A.a);   }
}
```

Overloading of Methods and Constructor

Method overloading:



- In Java it is possible to define **two or more methods within the same class that share the same name, as long as their parameter declarations are different**.
- When this is the case, the methods are **said to be *overloaded***, and the process is referred to as ***method overloading***.
- **Method overloading is one of the ways that Java implements Polymorphism.**
- There are two ways to overload the method in java
  1. By changing number of arguments
  2. By changing the data type

1. By changing number of arguments:

Example:

```
class Calculation
{
        void sum(int a,int b)
        {
                System.out.println(a+b);
        }
        void sum(int a,int b,int c)
        {
                System.out.println(a+b+c);
        }
```

```
}
class CalDemo
{
        public static void main(String args[])
        {
                Calculation c=new Calculation();
                c.sum(10,10,10);
                c.sum(20,20);
         }
}
```

2. By changing the data type

Example:

```
class Calculation
{
        void sum(int a,int b)
        {
                System.out.println(a+b);
        }
        void sum(double a, double b)
        {
                System.out.println(a+b);
        }
        void sum(float a, float b)
        {       System.out.println(a+b);
        }
}
class CalDemo
{       public static void main(String args[])
        {
           Calculation c=new Calculation();
           c.sum(10,20);
           c.sum(20.5,30.5);
           c.sum(10.5f,20.6);
         }
}
```

1 WORD QUESTION ANSWER

Sr no	Question	Answer
1.	Which keyword can be used in a subclass to call the constructor of super class?	Super
2.	What is the process of defining a method in a subclass having same name & type signature as a method in its super class?	Method overriding
3.	Which keywords can be used to prevent method overriding?	Final
4	Which is correct way of calling a constructor having no parameters, of super class a by subclass b?	Super();
5	Which is supported by method overriding in java?	Polymorphism

Constructor Overloading

Example:

```
class Student
{        int id, age;
         String name;
         Student( )  //no args or Zero parameterized constructor
         {  id = 0;
            name = ” ”;
            age=0
          }
         Student(int i, String n)  //Two parameterized constructor
         {  id = i;
            name = n;
            age=18;
         }
         Student(int  i, String n, int a) //Three parameterized constructor
         {        id = i;
                  name = n;
                  age=a;
          }
         void display()
         {
                  System.out.println(id+" "+name+" "+age);
         }
```

```
        public static void main(String args[])
        {
            Student s1 = new Student(1,“XYZ");
            Student s2 = new Student(2,"ABC",25);
            Student s3 = new Student();
            s1.display();
            s2.display();
            s3.display();
         }
}
```

1 WORD QUESTION ANSWER

Sr no.	Question	Answer
1.	A java constructor is like a method without ___.	Return type
2.	The name of a constructor and the name of a class are ___.	Same
3.	The placement of a constructor inside a class should be ___.	Anywhere in the class
4	Memory is allocated to an object once the execution of ___ is over in java language.	Constructor
5	When is the constructor called for an object?	As soon as object is created
6	Why do we use constructor overloading?	To initialize the object in different ways

VarArgs

- In Java var args is a unique concept.
  - **What will happen if you don't know how many arguments you need to pass to a method?**
  - **What will happen if you want to pass unlimited variables to a method?**
- There is an answer for these questions in Java. It is simple, and there is varargs.

Normal Method

public void MyMeth(int a,int b){....}

VarArg Method

public void MyMeth(int... a) {....}

Explanation

- The normal method above takes no more than 2 parameters of *int* type. The varargs method **takes as many number of parameters as you give of *int* type only***
- Not only for primitive types, varargs in java is applicable for classes also. Simply, applicable for all kinds of parameters.

How it works?

- In the above varargs method, **parameter acts as an int array with a reference name *a***.
- **To check this, you can call the *length* property of the arrays and that works.**
- **Example:**

```
class Test
{
    static void display(String... values)
    {
            System.out.println("display method invoked ");
    }
    public static void main(String args[])
    {
            display();//zero argument
            display("my","name","is","varargs");//four arguments
    }
}
```