JAVA PROGRAMMING

18MCA32C

Unit – I

INTRODUCTION

FACULTY

Dr. K. ARTHI MCA, M.Phil., Ph.D.,

Assistant Professor,

Postgraduate Department of Computer Applications,

Government Arts College (Autonomous),

Coimbatore 641018.

JAVA PROGRAMMING
18MCA32C
Syllabus

Objective: On successful completion of the course the students should have understood the Basic concept and fundamentals of core java classes, API, OOPS concept in Java and features of OOPS.

UNIT I: The Genesis of Java - The Java class Libraries - Data types, Variables - Operators - Arrays. Control Statements: Selection statements - Iteration statements - Jump statements. Introducing classes: Class Fundamentals - Declaring objects - Methods.

UNIT II: Constructors - this keyword - Garbage collection. Overloading Methods - Access controls - Nested and Inner classes. Inheritance: Inheritance basics - using Super - Method overriding - Dynamic method Dispatch - Abstract classes - using final with inheritance. Packages and Interfaces: Packages - Access protection - Importing Packages - Interfaces.

UNIT III: Exception Handling: Exception Handling Fundamentals - Java's Built in Exceptions - creating own Exception subclasses. Multithreaded Programming: The Java Thread Model - Creating a Thread - Synchronization - Inter Thread communication.

UNIT IV: I/O Basics - Reading console Input -Writing Console Output - Reading and writing Files - Exploring java.io. Applet Fundamentals - Applet Basics - Introducing the AWT.

UNIT V: Software Development using Java: Java Beans introduction - Servlets: Life cycle - A simple servlet - servlet API - Handling HTTP Request and Responses - Session tracking. Networking Basics - Remote Method Invocation (RMI) - Accessing Database with JDBC.

TEXT BOOKS:

1. Herbert Schildt, "The Complete Reference Java 2", 2nd Ed, Tata McGraw Hill (I) Pvt. Ltd.,2002.
2. H.M. Deitel and P. J. Deitel, "Java How to Program", 6th Ed, PHI/Pearson Education Asia 2005.

History of Java

1. History of Java
2. Java Version History

The history of Java is very interesting. Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time. The history of Java starts with the Green Team. Java team members (also known as **Green Team**), initiated this project to develop a language for digital devices such as set-top boxes, televisions, etc. However, it was suited for internet programming. Later, Java technology was incorporated by Netscape.

The principles for creating Java programming were "Simple, Robust, Portable, Platform-independent, Secured, High Performance, Multithreaded, Architecture Neutral, Object-Oriented, Interpreted, and Dynamic". Java was developed by James Gosling, who is known as the father of Java, in 1995. James Gosling and his team members started the project in the early '90s.

Currently, Java is used in internet programming, mobile devices, games, e-business solutions, etc. There are given significant points that describe the history of Java.

1) **James Gosling**, **Mike Sheridan**, and **Patrick Naughton** initiated the Java language project in June 1991. The small team of sun engineers called **Green Team**.

2) Initially designed for small, embedded systems in electronic appliances like set-top boxes.

3) Firstly, it was called **"Greentalk"** by James Gosling, and the file extension was .gt.

4) After that, it was called **Oak** and was developed as a part of the Green project.

Why Java named "Oak"?

5) **Why Oak?** Oak is a symbol of strength and chosen as a national tree of many countries like the U.S.A., France, Germany, Romania, etc.

6) In 1995, Oak was renamed as **"Java"** because it was already a trademark by Oak Technologies.

Why Java Programming named "Java"?

7) **Why had they chosen java name for Java language?** The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA", etc. They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell and fun to say.

According to James Gosling, "Java was one of the top choices along with **Silk**". Since Java was so unique, most of the team members preferred Java than other names.

8) Java is an island of Indonesia where the first coffee was produced (called java coffee). It is a kind of espresso bean. Java name was chosen by James Gosling while having coffee near his office.

9) Notice that Java is just a name, not an acronym.

10) Initially developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995.

11) In 1995, Time magazine called **Java one of the Ten Best Products of 1995**.

12) JDK 1.0 released in(January 23, 1996). After the first release of Java, there have been many additional features added to the language. Now Java is being used in Windows applications, Web applications, enterprise applications, mobile applications, cards, etc. Each new version adds the new features in Java.

History of Java

James Gosling initiated Java language project in June 1991 for use in one of his many set-top box projects. The language, initially called 'Oak' after an oak tree that stood outside Gosling's office, also went by the name 'Green' and ended up later being renamed as Java, from a list of random words.

Sun released the first public implementation as Java 1.0 in 1995. It promised **Write Once, Run Anywhere** (WORA), providing no-cost run-times on popular platforms.

On 13 November, 2006, Sun released much of Java as free and open source software under the terms of the GNU General Public License (GPL).

On 8 May, 2007, Sun finished the process, making all of Java's core code free and open-source, aside from a small portion of code to which Sun did not hold the copyright.

Java class library.

The **Java Class Library** (**JCL**) is a set of dynamically loadable libraries that Java Virtual Machine (JVM) languages can call at run time. Because the Java Platform is not dependent on a specific operating system, applications cannot rely on any of the platform-native libraries. Instead, the Java Platform provides a comprehensive set of standard class libraries, containing the functions common to modern operating systems.

These libraries provide –

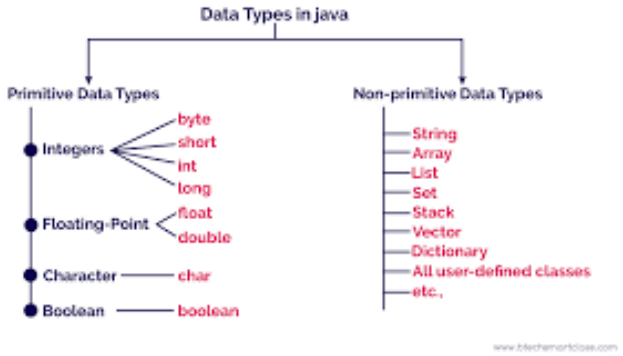
- Container classes and Regular Expressions.
- Interfaces for tasks that depend on the hardware of the OS such as network and file access.
- In case, the underlying platform does not support certain feature of Java then, these libraries surpass that specific feature if needed.

JCL serves three purposes within the JVM:

- Like other standard code libraries, they provide the programmer a well-known set of useful facilities, such as container classes and regular expression processing.
- The library provides an abstract interface to tasks that would normally depend heavily on the hardware and operating system, such as network access and file access.
- Some underlying platforms may not support all of the features a Java application expects. In these cases, the library implementation can either emulate those features or provide a consistent way to check for the presence of a specific feature.

Java Data Types

- Primitive data types - includes byte , short , **int** , long , float , double , boolean and char.
- Non-primitive data types - such as String, Arrays and Classes



Data Types in Java

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

Java Primitive Data Types

In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.

> Java is a statically-typed programming language. It means, all variables must be declared before its use. That is why we need to declare variable's type and name.

There are 8 types of primitive data types:

- boolean data type
- byte data type
- char data type
- short data type
- int data type
- long data type
- float data type
- double data type

Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Boolean Data Type

The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.

The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.

Example: Boolean one = false

Byte Data Type

The byte data type is an example of primitive data type. It isan 8-bit signed two's complement integer. Its value-range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0.

The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type.

Example: byte a = 10, byte b = -20

Short Data Type

The short data type is a 16-bit signed two's complement integer. Its value-range lies between -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.

The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer.

Example: short s = 10000, short r = -5000

Int Data Type

The int data type is a 32-bit signed two's complement integer. Its value-range lies between - 2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive). Its minimum value is - 2,147,483,648and maximum value is 2,147,483,647. Its default value is 0.

The int data type is generally used as a default data type for integral values unless if there is no problem about memory.

Example: int a = 100000, int b = -200000

Long Data Type

The long data type is a 64-bit two's complement integer. Its value-range lies between - 9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)(inclusive). Its minimum value is - 9,223,372,036,854,775,808and maximum value is 9,223,372,036,854,775,807. Its default value is 0. The long data type is used when you need a range of values more than those provided by int.

Example: long a = 100000L, long b = -200000L

Float Data Type

The float data type is a single-precision 32-bit IEEE 754 floating point.Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F.

Example: float f1 = 234.5f

Double Data Type

The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.

Example: double d1 = 12.3

Char Data Type

The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).The char data type is used to store characters.

Example: char letterA = 'A'

Java Variables

A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type.

Variable is a name of memory location. There are three types of variables in java: local, instance and static.

There are two types of data types in Java: primitive and non-primitive.

Variable

Variable is name of *reserved area allocated in memory*. In other words, it is a *name of memory location*. It is a

combination of "vary + able" that means its value can be changed.

```
1. int data=50;//Here data is variable
```

Types of Variables

There are three types of variables in Java:

- local variable
- instance variable
- static variable

1) Local Variable

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

2) Instance Variable

A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static.

It is called instance variable because its value is instance specific and is not shared among instances.

3) Static variable

A variable which is declared as static is called static variable. It cannot be local. You can create a single copy of static variable and share among all the instances of the class. Memory allocation for static variable happens only once when the class is loaded in the memory.

Example to understand the types of variables in java

```
1. class A{
2. int data=50;//instance variable
3. static int m=100;//static variable
4. void method(){
5. int n=90;//local variable
6. }
7. }//end of class
```

Operators in Java

Operator in Java is a symbol which is used to perform operations. For example: +, -, *, / etc.

There are many types of operators in Java which are given below:

- Unary Operator,
- Arithmetic Operator,
- Shift Operator,
- Relational Operator,
- Bitwise Operator,
- Logical Operator,
- Ternary Operator and
- Assignment Operator.

Java Operator Precedence

Operator Type	Category	Precedence
Unary	postfix	*expr*++ *expr*--
	prefix	++*expr* --*expr* +*expr* -*expr* ~ !
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||

Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Java Arrays

Normally, an array is a collection of similar type of elements which has contiguous memory location.

Java array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

Unlike C/C++, we can get the length of the array using the length member. In C/C++, we need to use the sizeof operator.

In Java, array is an object of a dynamically generated class. Java array inherits the Object class, and implements the Serializable as well as Cloneable interfaces. We can store primitive values or objects in an array in Java. Like C/C++, we can also create single dimentional or multidimentional arrays in Java.

Moreover, Java provides the feature of anonymous arrays which is not available in C/C++.

Advantages

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

Disadvantages

- **Size Limit:** We can store only the 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 which grows automatically.

Types of Array in java

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Single Dimensional Array in Java

Syntax to Declare an Array in Java

```
1. dataType[] arr; (or)
2. dataType []arr; (or)
3. dataType arr[];
```

Instantiation of an Array in Java

```
1. arrayRefVar=new datatype[size];
```

Example of Java Array

Let's see the simple example of java array, where we are going to declare, instantiate, initialize and traverse an array.

```
1. //Java Program to illustrate how to declare, instantiate, initialize
2. //and traverse the Java array.
3. class Testarray{
4. public static void main(String args[]){
5. int a[]=new int[5];//declaration and instantiation
6. a[0]=10;//initialization
7. a[1]=20;
8. a[2]=70;
9. a[3]=40;
10. a[4]=50;
11. //traversing array
12. for(int i=0;i<a.length;i++)//length is the property of array
13. System.out.println(a[i]);
14. }}
```

Test it Now

Output:

```
10
20
70
40
50
```

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
```

Example to initialize Multidimensional Array in Java

```
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;
```

Example of Multidimensional Java Array

Let's see the simple example to declare, instantiate, initialize and print the 2Dimensional array.

```
//Java Program to illustrate the use of multidimensional array
class Testarray3{
public static void main(String args[]){
//declaring and initializing 2D array
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
//printing 2D array
for(int i=0;i<3;i++){
 for(int j=0;j<3;j++){
   System.out.print(arr[i][j]+" ");
 }
 System.out.println();
}
}}
```

Java if-else Statement

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

Syntax:

```
if(condition){
//code if condition is true
}else{
//code if condition is false
}
```

Example:

```
//A Java Program to demonstrate the use of if-else statement.
//It is a program of odd and even number.
public class IfElseExample {
public static void main(String[] args) {
   //defining a variable
   int number=13;
   //Check if the number is divisible by 2 or not
   if(number%2==0){
      System.out.println("even number");
   }else{
      System.out.println("odd number");
   }
}
}
```

Using Ternary Operator

We can also use ternary operator (? :) to perform the task of if...else statement. It is a shorthand way to check the condition. If the condition is true, the result of ? is returned. But, if the condition is false, the result of : is returned.

Example:

```
public class IfElseTernaryExample {
public static void main(String[] args) {
   int number=13;
   //Using ternary operator
   String output=(number%2==0)?"even number":"odd number";
   System.out.println(output);
}
}
```

Output:

```
odd number
```

```
Selection statements
```

Java if-else-if ladder Statement

The if-else-if ladder statement executes one condition from multiple statements.

Syntax:

```
if(condition1){
//code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
else{
//code to be executed if all the conditions are false
}
```

Example:

```
//Java Program to demonstrate the use of If else-if ladder.
//It is a program of grading system for fail, D grade, C grade, B grade, A grade and A+.
public class IfElseIfExample {
public static void main(String[] args) {
    int marks=65;

    if(marks<50){
        System.out.println("fail");
    }
    else if(marks>=50 && marks<60){
        System.out.println("D grade");
    }
    else if(marks>=60 && marks<70){
        System.out.println("C grade");
    }
    else if(marks>=70 && marks<80){
        System.out.println("B grade");
    }
    else if(marks>=80 && marks<90){
        System.out.println("A grade");
    }else if(marks>=90 && marks<100){
        System.out.println("A+ grade");
    }else{
```

```
        System.out.println("Invalid!");
    }
}
}
```

Output:

```
C grade
```

Program to check POSITIVE, NEGATIVE or ZERO:

```
public class PositiveNegativeExample {
public static void main(String[] args) {
    int number=-13;
    if(number>0){
    System.out.println("POSITIVE");
    }else if(number<0){
    System.out.println("NEGATIVE");
    }else{
    System.out.println("ZERO");
   }
}
}
```

Output:

```
NEGATIVE
```

Java Nested if statement

The nested if statement represents the *if block within another if block*. Here, the inner if block condition executes only when outer if block condition is true.

Syntax:

```
if(condition){
    //code to be executed
        if(condition){
          //code to be executed
    }
}
```

Example:

```
//Java Program to demonstrate the use of Nested If Statement.
public class JavaNestedIfExample {
public static void main(String[] args) {
    //Creating two variables for age and weight
    int age=20;
```

```
    int weight=80;
    //applying condition on age and weight
    if(age>=18){
       if(weight>50){
          System.out.println("You are eligible to donate blood");
       }
    }
}}
```

Test it Now

Output:

```
You are eligible to donate blood
```

Example 2:

```
//Java Program to demonstrate the use of Nested If Statement.
public class JavaNestedIfExample2 {
public static void main(String[] args) {
    //Creating two variables for age and weight
    int age=25;
    int weight=48;
    //applying condition on age and weight
    if(age>=18){
       if(weight>50){
          System.out.println("You are eligible to donate blood");
       } else{
          System.out.println("You are not eligible to donate blood");
       }
    } else{
      System.out.println("Age must be greater than 18");
    }
} }
```

Test it Now

Output:

```
You are not eligible to donate blood
```

Java Switch Statement

The Java *switch statement* executes one statement from multiple conditions. It is like if-else-if ladder statement. The switch statement works with byte, short, int, long, enum types, String and some wrapper types like Byte, Short, Int, and Long. Since Java 7, you can use strings in the switch statement.

In other words, the switch statement tests the equality of a variable against multiple values.

Points to Remember

- There can be *one or N number of case values* for a switch expression.
- The case value must be of switch expression type only. The case value must be *literal or constant*. It doesn't allow variables.

- The case values must be *unique*. In case of duplicate value, it renders compile-time error.
- The Java switch expression must be of *byte, short, int, long (with its Wrapper type), enums and string*.
- Each case statement can have a *break statement* which is optional. When control reaches to the break statement, it jumps the control after the switch expression. If a break statement is not found, it executes the next case.
- The case value can have a *default label* which is optional.

Syntax:

```
switch(expression){
case value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
......

default:
 code to be executed if all cases are not matched;
}
```

Example:

```
public class SwitchExample {
public static void main(String[] args) {
    //Declaring a variable for switch expression
    int number=20;
    //Switch expression
    switch(number){
    //Case statements
    case 10: System.out.println("10");
    break;
    case 20: System.out.println("20");
    break;
    case 30: System.out.println("30");
    break;
    //Default case statement
    default:System.out.println("Not in 10, 20 or 30");
    }
}
}
```

Test it Now

Output:

```
20
```

Iteration statements.

Loops in Java

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

- for loop
- while loop
- do-while loop

Java For Loop vs While Loop vs Do While Loop

Comparison	for loop	while loop	do while loop
Introduction	The Java for loop is a control flow statement that iterates a part of the programs multiple times.	The Java while loop is a control flow statement that executes a part of the programs repeatedly on the basis of given boolean condition.	The Java do while loop is a control flow statement that executes a part of the programs at least once and the further execution depends upon the given boolean condition.
When to use	If the number of iteration is fixed, it is recommended to use for loop.	If the number of iteration is not fixed, it is recommended to use while loop.	If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use the do-while loop.
Syntax	`for(init;condition;incr/decr){ // code to be executed }`	`while(condition){ //code to be executed }`	`do{ //code to be executed }while(condition);`
Example	`//for loop for(int i=1;i<=10;i++){ System.out.println(i);`	`//while loop int i=1; while(i<=10){`	`//do-while loop int i=1; do{`

	}
	
		System.out.println(i);
		i++;
		}
			System.out.println(i);
			i++;
			}while(i<=10);

Syntax for infinitive loop	for(;;){
	//code to be executed
	}
		while(true){
		//code to be executed
		}
			do{
			//code to be executed
			}while(true);

Java For Loop

The Java *for loop* is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.

There are three types of for loops in java.

- Simple For Loop
- For-each or Enhanced For Loop
- Labeled For Loop

Java Simple For Loop

A simple for loop is the same as C/C++. We can initialize the variable, check condition and increment/decrement value. It consists of four parts:

1. **Initialization**: It is the initial condition which is executed once when the loop starts. Here, we can initialize the variable, or we can use an already initialized variable. It is an optional condition.
2. **Condition**: It is the second condition which is executed each time to test the condition of the loop. It continues execution until the condition is false. It must return boolean value either true or false. It is an optional condition.
3. **Statement**: The statement of the loop is executed each time until the second condition is false.
4. **Increment/Decrement**: It increments or decrements the variable value. It is an optional condition.

Syntax:

```
for(initialization;condition;incr/decr){
//statement or code to be executed
}
```

Flowchart:

Example:

```
//Java Program to demonstrate the example of for loop
```

```
2.  //which prints table of 1
3.  public class ForExample {
4.  public static void main(String[] args) {
5.     //Code of Java for loop
6.     for(int i=1;i<=10;i++){
7.        System.out.println(i);
8.     }
9.  }
10. }
```

Test it Now

Output:

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

Java Nested For Loop

If we have a for loop inside the another loop, it is known as nested for loop. The inner loop executes completely whenever outer loop executes.

Example:

```
1.  public class NestedForExample {
2.  public static void main(String[] args) {
3.  //loop of i
4.  for(int i=1;i<=3;i++){
5.  //loop of j
6.  for(int j=1;j<=3;j++){
7.        System.out.println(i+" "+j);
8.  }//end of i
9.  }//end of j
10. }
11. }
```

Output:

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
```

```
3 3
```

Pyramid Example 1:

```
1.  public class PyramidExample {
2.  public static void main(String[] args) {
3.  for(int i=1;i<=5;i++){
4.  for(int j=1;j<=i;j++){
5.  System.out.print("* ");
6.  }
7.  System.out.println();//new line
8.  }
9.  }
10. }
```

Output:

```
*
* *
* * *
* * * *
* * * * *
```

Loops in Java

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

- for loop
- while loop
- do-while loop

Java For Loop vs While Loop vs Do While Loop

Comparison	for loop	while loop	do while loop
Introduction	The Java for loop is a control flow statement that iterates a part of the programs multiple times.	The Java while loop is a control flow statement that executes a part of the programs repeatedly on the basis of given boolean condition.	The Java do while loop is a control flow statement that executes a part of the programs at least once and the further execution depends upon the given boolean condition.

When to use	If the number of iteration is fixed, it is recommended to use for loop.	If the number of iteration is not fixed, it is recommended to use while loop.	If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use the do-while loop.

Syntax	`for(init;condition;incr/decr){`
	`// code to be executed`
	`}`
		`while(condition){`
		`//code to be executed`
		`}`
			`do{`
			`//code to be executed`
			`}while(condition);`

Example	`//for loop`
	`for(int i=1;i<=10;i++){`
	`System.out.println(i);`
	`}`
		`//while loop`
		`int i=1;`
		`while(i<=10){`
		`System.out.println(i);`
		`i++;`
		`}`
			`//do-while loop`
			`int i=1;`
			`do{`
			`System.out.println(i);`
			`i++;`
			`}while(i<=10);`

Syntax for infinitive loop	`for(;;){`
	`//code to be executed`
	`}`
		`while(true){`
		`//code to be executed`
		`}`
			`do{`
			`//code to be executed`
			`}while(true);`

Java For Loop

The Java *for loop* is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.

There are three types of for loops in java.

- Simple For Loop
- For-each or Enhanced For Loop
- Labeled For Loop

Java Simple For Loop

A simple for loop is the same as C/C++. We can initialize the variable, check condition and increment/decrement value. It consists of four parts:

1. **Initialization**: It is the initial condition which is executed once when the loop starts. Here, we can initialize the variable, or we can use an already initialized variable. It is an optional condition.
2. **Condition**: It is the second condition which is executed each time to test the condition of the loop. It continues execution until the condition is false. It must return boolean value either true or false. It is an optional condition.

3. **Statement**: The statement of the loop is executed each time until the second condition is false.
4. **Increment/Decrement**: It increments or decrements the variable value. It is an optional condition.

Syntax:

```
for(initialization;condition;incr/decr){
//statement or code to be executed
}
```

Flowchart:

Example:

```
//Java Program to demonstrate the example of for loop
//which prints table of 1
public class ForExample {
public static void main(String[] args) {
    //Code of Java for loop
    for(int i=1;i<=10;i++){
        System.out.println(i);
    }
}
}
```

Test it Now

Output:

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

Java Nested For Loop

If we have a for loop inside the another loop, it is known as nested for loop. The inner loop executes completely whenever outer loop executes.

Example:

```
public class NestedForExample {
public static void main(String[] args) {
//loop of i
for(int i=1;i<=3;i++){
```

```
5.  //loop of j
6.  for(int j=1;j<=3;j++){
7.      System.out.println(i+" "+j);
8.  }//end of i
9.  }//end of j
10. }
11. }
```

Output:

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
```

Pyramid Example 1:

```
1.  public class PyramidExample {
2.  public static void main(String[] args) {
3.  for(int i=1;i<=5;i++){
4.  for(int j=1;j<=i;j++){
5.      System.out.print("* ");
6.  }
7.  System.out.println();//new line
8.  }
9.  }
10. }
```

Output:

```
*
* *
* * *
* * * *
* * * * *
```

Jump statements

Java Break Statement

When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.

The Java *break* statement is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.

We can use Java break statement in all types of loops such as for loop, while loop and do-while loop.

Syntax:

```
jump-statement;
break;
```

Java Continue Statement

The continue statement is used in loop control structure when you need to jump to the next iteration of the loop immediately. It can be used with for loop or while loop.

The Java *continue statement* is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition. In case of an inner loop, it continues the inner loop only.

We can use Java continue statement in all types of loops such as for loop, while loop and do-while loop.

Syntax:

```
jump-statement;
continue;
```

Java Continue Statement Example

Example:

```
//Java Program to demonstrate the use of continue statement
//inside the for loop.
public class ContinueExample {
public static void main(String[] args) {
   //for loop
   for(int i=1;i<=10;i++){
      if(i==5){
         //using continue statement
         continue;//it will skip the rest statement
      }
      System.out.println(i);
   }
}
}
```

Test it Now

Output:

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

Objects and Classes in Java

In this page, we will learn about Java objects and classes. In object-oriented programming technique, we design a program using objects and classes.

An object in Java is the physical as well as a logical entity, whereas, a class in Java is a logical entity only.

What is an object in Java

An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system.

An object has three characteristics:

- **State:** represents the data (value) of an object.
- **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
- **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

For Example, Pen is an object. Its name is Reynolds; color is white, known as its state. It is used to write, so writing is its behavior.

An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

Object Definitions:

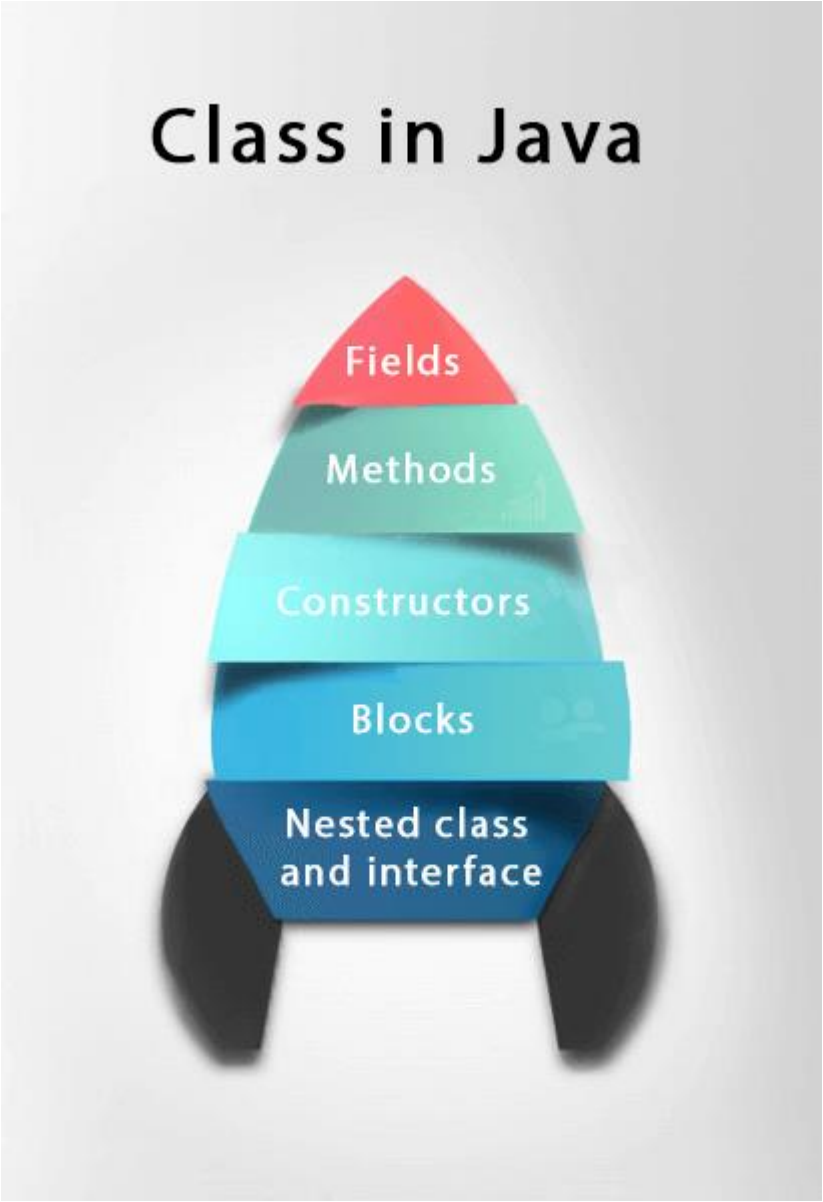
- An object is *a real-world entity*.
- An object is *a runtime entity*.
- The object is *an entity which has state and behavior*.
- The object is *an instance of a class*.

What is a class in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

- **Fields**
- **Methods**
- **Constructors**
- **Blocks**
- **Nested class and interface**



Syntax to declare a class:

```
class <class_name>{
    field;
    method;
}
```

Instance variable in Java

A variable which is created inside the class but outside the method is known as an instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.

Method in Java

In Java, a method is like a function which is used to expose the behavior of an object.

Advantage of Method

- Code Reusability
- Code Optimization

---

new keyword in Java

The new keyword is used to allocate memory at runtime. All objects get memory in Heap memory area.

Object and Class Example: main within the class

In this example, we have created a Student class which has two data members id and name. We are creating the object of the Student class by new keyword and printing the object's value.

Here, we are creating a main() method inside the class.

File: Student.java

```
1. //Java Program to illustrate how to define a class and fields
2. //Defining a Student class.
3. class Student{
4.  //defining fields
5.  int id;//field or data member or instance variable
6.  String name;
7.  //creating main method inside the Student class
8.  public static void main(String args[]){
9.   //Creating an object or instance
10.  Student s1=new Student();//creating an object of Student
11.  //Printing values of the object
12.  System.out.println(s1.id);//accessing member through reference variable
13.  System.out.println(s1.name);
14.  }
15. }
```

Test it Now

Output:

```
0
null
```

3 Ways to initialize object

There are 3 ways to initialize object in Java.

1. By reference variable
2. By method
3. By constructor

1) Object and Class Example: Initialization through reference

Initializing an object means storing data into the object. Let's see a simple example where we are going to initialize the object through a reference variable.

File: TestStudent2.java

```
1. class Student{
2.  int id;
3.  String name;
4. }
5. class TestStudent2{
6.  public static void main(String args[]){
7.   Student s1=new Student();
8.   s1.id=101;
9.   s1.name="Sonoo";
10.  System.out.println(s1.id+" "+s1.name);//printing members with a white space
11.  }
12. }
```

Test it Now

Output:

```
101 Sonoo
```

We can also create multiple objects and store information in it through reference variable.

File: TestStudent3.java

```
1. class Student{
2.  int id;
3.  String name;
4. }
5. class TestStudent3{
6.  public static void main(String args[]){
7.   //Creating objects
8.   Student s1=new Student();
9.   Student s2=new Student();
10.  //Initializing objects
```

```
  s1.id=101;
  s1.name="Sonoo";
  s2.id=102;
  s2.name="Amit";
  //Printing data
  System.out.println(s1.id+" "+s1.name);
  System.out.println(s2.id+" "+s2.name);
 }
}
```

Test it Now

Output:

```
101 Sonoo
102 Amit
```

2) Object and Class Example: Initialization through method

In this example, we are creating the two objects of Student class and initializing the value to these objects by invoking the insertRecord method. Here, we are displaying the state (data) of the objects by invoking the displayInformation() method.

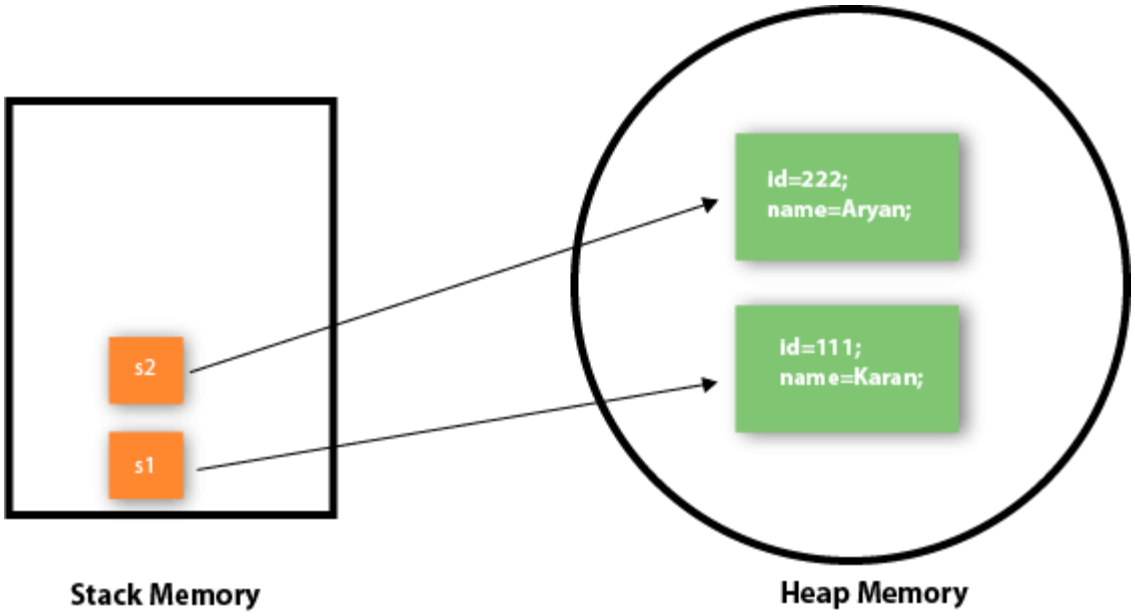
File: TestStudent4.java

```
class Student{
 int rollno;
 String name;
 void insertRecord(int r, String n){
  rollno=r;
  name=n;
 }
 void displayInformation(){System.out.println(rollno+" "+name);}
}
class TestStudent4{
 public static void main(String args[]){
  Student s1=new Student();
  Student s2=new Student();
  s1.insertRecord(111,"Karan");
  s2.insertRecord(222,"Aryan");
  s1.displayInformation();
  s2.displayInformation();
 }
}
```

Test it Now

Output:

```
111 Karan
222 Aryan
```



As you can see in the above figure, object gets the memory in heap memory area. The reference variable refers to the object allocated in the heap memory area. Here, s1 and s2 both are reference variables that refer to the objects allocated in memory.

---

3) Object and Class Example: Initialization through a constructor

We will learn about constructors in Java later.

Object and Class Example: Employee

Let's see an example where we are maintaining records of employees.

File: TestEmployee.java

```
1.  class Employee{
2.     int id;
3.     String name;
4.     float salary;
5.     void insert(int i, String n, float s) {
6.        id=i;
7.        name=n;
8.        salary=s;
9.     }
10.    void display(){System.out.println(id+" "+name+" "+salary);}
11. }
12. public class TestEmployee {
13. public static void main(String[] args) {
14.    Employee e1=new Employee();
15.    Employee e2=new Employee();
```

```
16.     Employee e3=new Employee();
17.     e1.insert(101,"ajeet",45000);
18.     e2.insert(102,"irfan",25000);
19.     e3.insert(103,"nakul",55000);
20.     e1.display();
21.     e2.display();
22.     e3.display();
23. }
24. }
```

Test it Now

Output:

```
101 ajeet 45000.0
102 irfan 25000.0
103 nakul 55000.0
```

THANK YOU

The content for this material are taken from the prescribed text books & reference books.