UNIT-1

Features of Java:

The primary objective of Java programming language creation was to make it portable, simple and secure programming language. Apart from this, there are also some excellent features which play an important role in the popularity of this language. The features of Java are also known as Java buzzwords.

1.Simple

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

2.Object-oriented

Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporate both data and behavior.

Basic concepts of OOPs are:

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

3.Platform Independent

Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.

Java code can be executed on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere (WORA).

4.Secured

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

- **Classloader:** Classloader in Java is a part of the Java Runtime Environment (JRE) which is used to load Java classes into the Java Virtual Machine dynamically. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.
- **Bytecode Verifier:** It checks the code fragments for illegal code that can violate access rights to objects.

- **Security Manager:** It determines what resources a class can access such as reading and writing to the local disk.

5.Robust

The English mining of Robust is strong. Java is robust because:

- It uses strong memory management.
- There is a lack of pointers that avoids security problems.
- There are exception handling and the type checking mechanism in Java. All these points make Java robust.

6.Portable

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

7.High-performance

Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

8.Multi-threaded

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

JVM (Java Virtual Machine) Architecture

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent)

What it does

The JVM performs following operation:

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

JVM is a virtual processor

1. **Interpreter:** Read bytecode stream then execute the instructions.

2. **Just-In-Time(JIT) compiler:** It is used to improve the performance. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here, the term "compiler" refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

The JVM is also known as a virtual machine as it does not exist physically.

JVM is essentially a part of the JRE (Java Run Environment). You cannot separately download and install it. You first need to install the JRE to install the JVM.

What is JRE?

JRE stands for Java Runtime Environment- also written as Java RTE. It is a set of software tools designed for running other software. It is an implementation of JVM, and JRE provides a runtime environment. In short, a user needs JRE to run any Java program. If not a programmer, the user doesn't need to install the JDK- JRE alone will help run the Java programs.

The JRE also exists physically. It consists of a library set + a few more files that the JVM (Java Virtual Machine) deploys at the runtime.

What is JDK?

The **JDK (Java Development Kit)** is a software development kit that develops applications in Java. Along with JRE, the JDK also consists of various development tools (Java Debugger, JavaDoc, compilers, etc.) The Java Runtime Environment (JRE) is an implementation of JVM.

Java APIs

Java APIs are integrated pieces of software that come with JDKs. APIs in Java provides the interface between two different applications and establish communication.

Types of Java APIs

There are four types of APIs in Java:

- Public
- Private
- Partner
- Composite

Public

Public (or open) APIs are Java APIs that come with the JDK. They do not have strict restrictions about how developers use them.

Private

Private (or internal) APIs are developed by a specific organization and are accessible to only employees who work for that organization.

Partner

Partner APIs are considered to be third-party APIs and are developed by organizations for strategic business operations.

Composite

Composite APIs are microservices, and developers build them by combining several service APIs.

User Interface Services

User interface service APIs are open-source APIs that allow developers to build user interfaces for mobile devices, computers, and other electronics.

First Java Program | Hello World Example

In this section, we will learn how to write the simple program of Java. We can write a simple hello Java program easily after installing the JDK.

To create a simple Java program, you need to create a class that contains the main method. Let's understand the requirement first.

The requirement for Java Hello World Example

- Install the JDK if you don't have installed it, download the JDK and install it.
- Set path of the jdk/bin directory. http://www.javatpoint.com/how-to-set-path-in-java
- Create the Java program
- Compile and run the Java program

Creating Hello World Example

```
1. Class Simple{
2. public static void main(String args[]){
3. System.out.println("Hello Java");
4. }
5. }
```

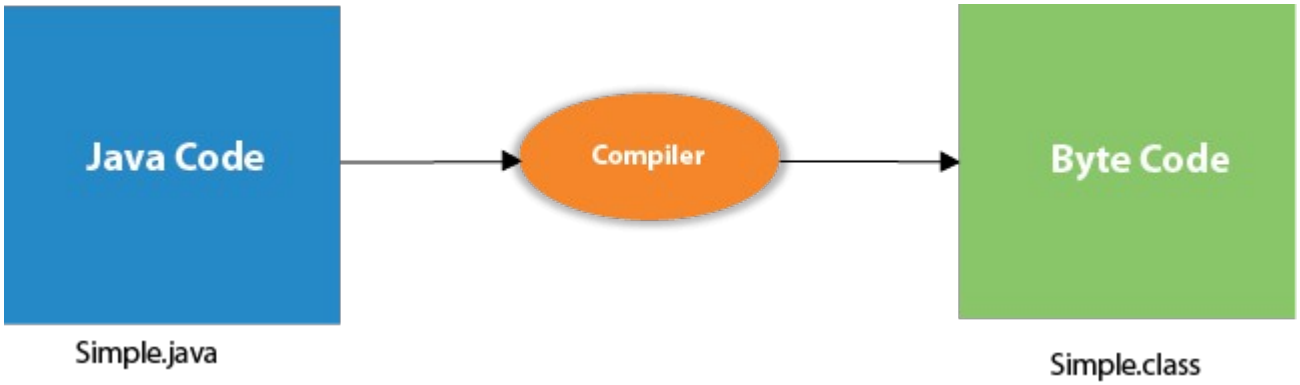
Save the above file as Simple.java.

To compile: javac Simple.java
To execute: java Simple
Output:

Hello Java

Compilation Flow:

When we compile Java program using javac tool, the Java compiler converts the source code into byte code.



Parameters used in First Java Program

- **class** keyword is used to declare a class in Java.
- **public** keyword is an access modifier that represents visibility. It means it is visible to all.
- **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main() method is executed by the JVM, so it doesn't require creating an object to invoke the main() method. So, it saves memory.
- **void** is the return type of the method. It means it doesn't return any value.
- **main** represents the starting point of the program.
- **String[] args** or **String args[]** is used for command line argument. We will discuss it in coming section.
- **System.out.println()** is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class. We will discuss the internal working of System.out.println() statement in the coming section.

Scanner:

Java User Input

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

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

Input Types

In the example above, we used the `nextLine()` method, which is used to read Strings. To read other types, look at the table below:

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

Java Tokens

What is token in Java?

The Java compiler breaks the line of code into text (words) is called **Java tokens**. These are the smallest element of the Java program. The Java compiler identified these words as tokens. These tokens are separated by the delimiters. It is useful for compilers to detect errors. Remember that the delimiters are not part of the Java tokens.

1. token<=identifier | keyword | separator | operator | literal | comment

For example, consider the following code.

```
1. Public class Demo
2. {
3. public static void main(String args[])
4. {
5. System.out.println("javatpoint");
6. }
7. }
```

In the above code snippet, **public, class, Demo, {, static, void, main, (, String, args, [,],), System, ., out, println, javatpoint**, etc. are the Java tokens.

The Java compiler translates these tokens into Java bytecode. Further, these bytecodes are executed inside the interpreted Java environment.

Types of Tokens

Java token includes the following:

- Keywords
- Identifiers
- Literals
- Operators
- Separators
- Comments

Java Constant

As the name suggests, a **constant** is an entity in programming that is immutable or Unchangeable. In other words, the value that cannot be changed. In this section, we will learn about **Java constant** and **how to declare a constant in Java**.

What is constant?

Constant is a value that cannot be changed after assigning it. Java does not directly support the constants. There is an alternative way to define the constants in Java by using the non-access modifiers static and final.

How to declare constant in Java?

In Java, to declare any variable as constant, we use static and final modifiers. It is also known as **non-access** modifiers. According to the java naming conventiontothe identifier name must be in **capital letters**.

When we use **static** and **final** modifiers together, the variable remains static and can be initialized once. Therefore, to declare a variable as constant, we use both static and final modifiers. It shares a common memory location for all objects of its containing class.

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.

Variable

A variable is the name of a reserved area allocated in memory. In other words, it is a name of the memory location. It is a combination of "vary + able" which means its value can be changed.

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 an instance variable. It is not declared as static.

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

3) Static variable

A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all thThe Scope of Variables in Java

The Scope of Variables in Java

Each variable has a scope that specifies how long it will be seen and used in a programme.

There are four scopes for variables in Java: local, instance, class, and method parameters. Examining each of these scopes in more detail will be helpful.

Local Variables:

Local variables are those that are declared inside of a method, constructor, or code block. Only the precise block in which they are defined is accessible. The local variable exits the block's scope after it has been used, and its memory is freed. Temporary data is stored in local variables, which are frequently initialised in the block where they are declared. The Java compiler throws an error if a local variable is not initialised before being used. The range of local variables is the smallest of all the different variable types.

Example:

public SumExample

```
{
public void calculateSum(){
int a=5;//local variable
int b=10;//local variable
int sum=a+b;
System.out.println("The sum is:"+sum);
}/a,b,and sum go out of scope here
}
```

Instance Variables:

Within a class, but outside of any methods, constructors, or blocks, instance variables are declared. They are accessible to all methods and blocks in the class and are a part of an instance of the class. If an instance variable is not explicitly initialised, its default values are false for boolean types, null for object references, and 0 for numeric kinds. Until the class instance is destroyed, these variables' values are retained.

Example:

public class Circle {

```
1. double radius;   //instance variable
2. public double calculateArea(){
3. returnMath.PI*radius *radius;
4. }
5. }
```

Class Variables (Static Variables):

In a class but outside of any method, constructor, or block, the static keyword is used to declare class variables, also referred to as static variables. They relate to the class as a whole, not to any particular instance of the class. Class variables can be accessed by using the class name and are shared by all instances of the class. Like instance variables, they have default values, and they keep those values until the programme ends.

Example:

public class Bank{

```
1. static double interestRate;      //class variable
2. //...
3. }
```

Keywords: These are the **pre-defined** reserved words of any programming language. Each keyword has a special meaning. It is always written in lower case. Java provides the following keywords:

01. abstract	02. boolean	03. byte	04. break	05. class
06. case	07. catch	08. char	09. continue	10. default
11. do	12. double	13. else	14. extends	15. final
16. finally	17. float	18. for	19. if	20. implements
21. import	22. instanceof	23. int	24. interface	25. long
26. native	27. new	28. package	29. private	30. protected
31. public	32. return	33. short	34. static	35. super
36. switch	37. synchronized	38. this	39. thro	40. throws
41. transient	42. try	43. void	44. volatile	45. while
46. assert	47. const	48. enum	49. goto	50. strictfp

Identifier: Identifiers are used to name a variable, constant, function, class, and array. It usually defined by the user. It uses letters, underscores, or a dollar sign as the first character. The label is also known as a special kind of identifier that is used in the goto statement. Remember that the identifier name must be different from the reserved keywords. There are some rules to declare identifiers are:

- The first letter of an identifier must be a letter, underscore or a dollar sign. It cannot start with digits but may contain digits.
- The whitespace cannot be included in the identifier.
- Identifiers are case sensitive.

Some valid identifiers are:

1. PhoneNumber
2. PRICE
3. radius
4. a
5. a1
6. _phonenumber
7. $circumference
8. jagged_array
9. 12radius//invalid

Literals: In programming literal is a notation that represents a fixed value (constant) in the source code. It can be categorized as an integer literal, string literal, Boolean literal, etc. It is defined by the programmer. Once it has been defined cannot be changed. Java provides five types of literals are as follows:

- Integer
- Floating Point
- Character
- String
- Boolean

Operators:

In programming, operators are the special symbol that tells the compiler to perform a special operation. Java provides different types of operators that can be classified according to the functionality they provide. There are eight types of operators in Java, are as follows:

- Arithmetic Operators
- Assignment Operators
- Relational Operators
- Unary Operators
- Logical Operators
- Ternary Operators
- Bitwise Operators
- Shift Operators

Operator	Symbols
Arithmetic	+ , - , / , * , %
Unary	++ , - - , !
Assignment	= , += , -= , *= , /= , %= , ^=
Relational	==, != , < , >, <= , >=

Logical	&& , ||
Ternary	(Condition) ? (Statement1) : (Statement2);
Bitwise	& , | , ^ , ~
Shift	<< , >> , >>>

Separators: The separators in Java is also known as **punctuators**. There are nine separators in Java, are as follows:

separators <=;|,|.|(|)|{|}|[]

Square Brackets []: It is used to define array elements. A pair of square brackets represents the single-dimensional array, two pairs of square brackets represent the two-dimensional array.

- **Parentheses ():** It is used to call the functions and parsing the parameters.
- **Curly Braces {}:** The curly braces denote the starting and ending of a code block.
- **Comma (,):** It is used to separate two values, statements, and parameters.
- **Assignment Operator (=):** It is used to assign a variable and constant.
- **Semicolon (;):** It is the symbol that can be found at end of the statements. It separates the two statements.
- **Period (.):** It separates the package name form the sub-packages and class. It also separates a variable or method from a reference variable.

Java Character Set :Characters are the smallest units (elements) of Java language that are used to write Java tokens. These characters are defined by the Unicode character set.

A character set in Java is a set of alphabets, letters, and some special characters that are valid in java programming language.

The first character set used in the computer system was US-ASCII (American Standard Code for Information Interchange (ASCII pronounced as ass-kee)). It is limited to represent only American English.

Java language uses the character sets as the building block to form the basic elements such as identifiers, variables, array, etc in the program. These are as follows:

- Letters: Both lowercase (a, b, c, d, e, etc.) and uppercase (A, B, C, D, E, etc.) letters.
- Digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- Special symbols: _, (,), {, }, [,], +, -, *, /, %, !, &, |, ~, ^, <, =, >, $, #, ?, Comma (,), Dot (.), Colon (:), Semi-colon (;), Single quote (‘), Double quote (“), Back slash (\).
- White space: Space, Tab, New line.

Java Command Line Arguments

The java command-line argument is an argument i.e. passed at the time of running the java program.

The arguments passed from the console can be received in the java program and it can be used as an input.

So, it provides a convenient way to check the behavior of the program for the different values. You can pass **N** (1,2,3 and so on) numbers of arguments from the command prompt.

Simple example of command-line argument in java

In this example, we are receiving only one argument and printing it. To run this java program, you must pass at least one argument from the command prompt.

```
1. Class CommandLineExample{
2. public static void main(String args[]){
3. System.out.println("Your first argument is:"+args[0]);
4. }
5. }
```

compile by>javac CommandLineExample.java

```
1. run by >java CommandLineExample sonoo
```

Output: Your first argument is: sonoo

Example of command-line argument that prints all the values

In this example, we are printing all the arguments passed from the command-line. For this purpose, we have traversed the array using for loop.

```
Class A{
1. public static void main(String args[]){
2.
3. for(int i=0;i<args.length;i++)
4. System.out.println(args[i]);
5.
6. }
7. }
```

compile by>javac A.java

run b>java A sonoo jaiswal 1 3 abc

Output: sonoo
jaiswal
1
3
abc

Comments: Comments allow us to specify information about the program inside our Java code. Java compiler recognizes these comments as tokens but excludes it form further processing. The Java compiler treats comments as whitespaces.

Java provides the following two types of comments:

- **Line Oriented:** It begins with a pair of forwarding slashes (//).
- **Block-Oriented:** It begins with /* and continues until it founds */.

Java Type Casting

Type casting is when you assign a value of one primitive data type to another type.

In Java, there are two types of casting:

- **Widening Casting** (automatically) - converting a smaller type to a larger type size
  byte -> short -> char -> int -> long -> float -> double
- **Narrowing Casting** (manually) - converting a larger type to a smaller size type
  double -> float -> long -> int -> char -> short -> byte

Widening Casting

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

Example

```
public class Main {
  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

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 = 9.78d;
    int myInt = (int) myDouble; // Manual casting: double to int

    System.out.println(myDouble);   // Outputs 9.78
    System.out.println(myInt);      // Outputs 9
  }
}
```

Expression Evaluation

Evaluate an expression represented by a String. The expression can contain parentheses, you can assume parentheses are well-matched. For simplicity, you can assume only binary operations allowed are +, -, *, and /. Arithmetic Expressions can be written in one of three forms:

What is EvalEx Java?

EvalEx Java is an open-source library that allows you to evaluate mathematical expressions in Java with ease. It provides a simple and intuitive API for parsing and calculating complex expressions,

handling variables, and supporting a wide range of mathematical functions. With EvalEx Java, you can perform arithmetic, trigonometric, logarithmic, and other mathematical operations effortlessly.

What is operator precedence?

The **operator precedence** represents how two expressions are bind together. In an expression, it determines the grouping of operators with operands and decides how an expression will evaluate.

While solving an expression two things must be kept in mind the first is a **precedence** and the second is **associativity**.

Precedence

Precedence is the priority for grouping different types of operators with their operands. It is meaningful only if an expression has more than one operator with higher or lower precedence. The operators having higher precedence are evaluated first. If we want to evaluate lower precedence operators first, we must group operands by using parentheses and then evaluate.

Associativity

We must follow associativity if an expression has more than two operators of the same precedence. In such a case, an expression can be solved either **left-to-right** or **right-to-left,** accordingly.

Java Operator Precedence Example

Let's understand the operator precedence through an example. Consider the following expression and guess the answer.

1. 1 + 5 * 3

You might be thinking that the answer would be **18** but not so. Because the multiplication (*) operator has higher precedence than the addition (+) operator. Hence, the expression first evaluates 5*3 and then evaluates the remaining expression i.e. 1+15. Therefore, the answer will be **16**.

Let's see another example. Consider the following expression.

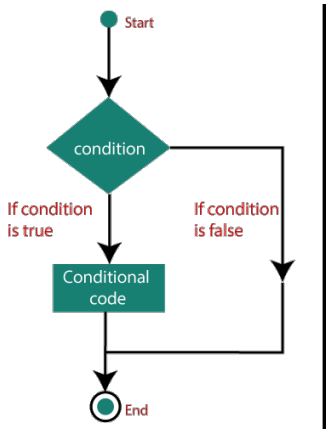
1. x + y * z / k

In the above expression, * and / operations are performed before + because of precedence. y is multiplied by z before it is divided by k because of associativity.

Decision-Making

conditional statements are used to perform different actions based on various conditions. The conditional statement evaluates a condition before the execution of instructions.

When you write the code, you require to perform different actions for different decisions. You can easily perform it by using conditional statements.



Types of Conditional Statements

if statement

- **if….else statement**
- **if….else if….statement**
- **nested if statement**
- **switch statement**

The if statement

It is one of the simplest decision-making statement which is used to decide whether a block of Java code will execute if a certain condition is true.

Syntax

if(condition){

1. //block of code will execute if the condition is true
2. }

If statements in Java is used to control the program flow based on some condition, it's used to execute some statement code block if the expression evaluated to true; otherwise, it will get skipped. This statement is the simplest way to modify the control flow of the program.

```
if(test_expression)
{
   statement 1;
   statement 2;
   ...
}
```

If else statements

If else statements in Java is also used to control the program flow based on some condition, only the difference is: it's used to execute some statement code block if the expression is evaluated to true, otherwise executes else statement code block.

Syntax:

```
if(test_expression)

{
  //execute your code
}
else
{
  //execute your code
}
```

else if statements

else if statements in Java is like another if condition, it's used in the program when **if statement** having multiple decisions.

Syntax:

```
if(test_expression)

{
  //execute your code
}
else if(test_expression n)
{
  //execute your code
}
else
{
  //execute your code
}
```

switch statement

Java switch statement is used when you have multiple possibilities for the if statement.

Syntax:

```
switch(variable)

{
case 1:
  //execute your code
break;

case n:
  //execute your code
break;

default:
  //execute your code
break;
```

}

After the end of each block it is necessary to insert a *break statement* because if the programmers do not use the break statement, all consecutive blocks of codes will get executed from each case onwards after matching the case block.

LOOPS IN JAVA:

Sometimes it is necessary for the program to execute the statement several times, and Java loops execute a block of commands a specified number of times until a condition is met. In this chapter, you will learn about all the looping statements of Java along with their use.

The process of repeatedly executing a collection of statement is called *looping*. The statements get executed many numbers of times based on the condition. But if the condition is given in such logic that the repetition continues any number of times with no fixed condition to stop looping those statements, then this type of looping is called ***infinite looping*.**

Java supports many looping features which enable programmers to develop concise Java programs with repetitive processes.

1.While loops

2.do while loops

3.for loops

while loops

while loops statement allows to repeatedly run the same block of code until a condition is met.

while loop is the most basic loop in Java. It has one control condition and executes as long the condition is true. The condition of the loop is tested before the body of the loop is executed; hence it is called an **entry-controlled** loop.

Syntax:

While (condition)

```
{
  statement(s);
  Incrementation;
}
```

do-while loops

Java do-while loops are very similar to the while loops, but it always executes the code block at least once and furthermore as long as the condition remains true. This loop is an **exit-controlled loop**.

Syntax

```
do

{
  statement(s);

}while( condition );
```

Java for loops

Java for loops is very similar to Java while loops in that it continues to process a block of code until a statement becomes false, and everything is defined in a single line.

Syntax:

```
for ( init; condition; increment )

{
  statement(s);
}
```

Labeled Loop

In programming, a **loop** is a sequence of instructions that is continually repeated until a certain condition is met. In this section, we will discuss the **labeled loop in Java** with examples.

A **label** is a valid variable name that denotes the name of the loop to where the control of execution should jump. To label a loop, place the label before the loop with a colon at the end. Therefore, a loop with the label is called a **labeled loop**.

Arrays:

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type with **square brackets**:

```
String[] cars;
```

We have now declared a variable that holds an array of strings. To insert values to it, you can place the values in a comma-separated list, inside curly braces:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

To create an array of integers, you could write:

```
int[] myNum = {10, 20, 30, 40};
```

Access the Elements of an Array

You can access an array element by referring to the index number.

This statement accesses the value of the first element in cars:

Example

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars[0]);
// Outputs Volvo
```

Note: Array indexes start with 0: [0] is the first element. [1] is the second element, etc.

Change an Array Element

To change the value of a specific element, refer to the index number:

Example

```
cars[0] = "Opel";
```

Example

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

cars[0] = "Opel";
System.out.println(cars[0]);
// Now outputs Opel instead of Volvo
```

Array Length:

To find out how many elements an array has, use the length property:

Example

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

System.out.println(cars.length);
// Outputs 4
```

Loop Through an Array

You can loop through the array elements with the for loop, and use the length property to specify how many times the loop should run.

The following example outputs all elements in the **cars** array:

Example

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (int i = 0; i < cars.length; i++) {
```

```
  System.out.println(cars[i]);
}
```

Loop Through an Array with For-Each

There is also a "**for-each**" loop, which is used exclusively to loop through elements in arrays:

Syntax

```
for (type variable : arrayname) {

  ...
}
```

The following example outputs all elements in the **cars** array, using a "**for-each**" loop:

Example

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (String i : cars) {
  System.out.println(i);
}
```

The example above can be read like this: **for each** String element (called **i** - as in **i**ndex) in **cars**, print out the value of **i**.

If you compare the for loop and **for-each** loop, you will see that the **for-each** method is easier to write, it does not require a counter (using the length property), and it is more readable.

Multidimensional Arrays:

A multidimensional array is an array of arrays.

Multidimensional arrays are useful when you want to store data as a tabular form, like a table with rows and columns.

To create a two-dimensional array, add each array within its own set of **curly braces**:

Example

```
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
```

myNumbers is now an array with two arrays as its elements.

Access Elements

To access the elements of the **myNumbers** array, specify two indexes: one for the array, and one for the element inside that array. This example accesses the third element (2) in the second array (1) of myNumbers:

Example

```
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
```

```
System.out.println(myNumbers[1][2]); // Outputs 7
```

Remember that: Array indexes start with 0: [0] is the first element. [1] is the second element, etc.

Change Element Values

You can also change the value of an element:

Example

```
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
myNumbers[1][2] = 9;
System.out.println(myNumbers[1][2]); // Outputs 9 instead of 7
```

Loop Through a Multi-Dimensional Array

We can also use a for loop inside another for loop to get the elements of a two-dimensional array (we still have to point to the two indexes):

Example

```
public class Main {
  public static void main(String[] args) {
    int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
    for (int i = 0; i < myNumbers.length; ++i) {
      for(int j = 0; j < myNumbers[i].length; ++j) {
        System.out.println(myNumbers[i][j]);
      }
    }
  }
}
```

Java Vector:

Vector is like the *dynamic array* which can grow or shrink its size. Unlike array, we can store n-number of elements in it as there is no size limit.

It is recommended to use the Vector class in the thread-safe implementation only. If you don't need to use the thread-safe implementation, you should use the ArrayList, the ArrayList will perform better in such case.

It is similar to the ArrayList, but with two differences

- Vector is synchronized.
- Java Vector contains many legacy methods that are not the part of a collections framework.

java Vector Methods

The following are the list of Vector class methods:

SN	Method	Description
1)	add()	It is used to append the specified element in the given vector.
2)	addAll()	It is used to append all of the elements in the specified collection to the end of

		this Vector.
3)	addElement()	It is used to append the specified component to the end of this vector. It increases the vector size by one.
4)	capacity()	It is used to get the current capacity of this vector.
5)	clear()	It is used to delete all of the elements from this vector.
6)	elementAt()	It is used to get the component at the specified index.
7)	elements()	It returns an enumeration of the components of a vector.
8)	equals()	It is used to compare the specified object with the vector for equality.
9)	firstElement()	It is used to get the first component of the vector.
10)	forEach()	It is used to perform the given action for each element of the Iterable until all elements have been processed or the action throws an exception.
11)	get()	It is used to get an element at the specified position in the vector.
12)	hashCode()	It is used to get the hash code value of a vector.
13)	indexOf()	It is used to get the index of the first occurrence of the specified element in the vector. It returns -1 if the vector does not contain the element.
14)	insertElementAt()	It is used to insert the specified object as a component in the given vector at the specified index.
15)	isEmpty()	It is used to check if this vector has no components.
16)	remove()	It is used to remove the specified element from the vector. If the vector does not contain the element, it is unchanged.
17	removeAll()	It is used to delete all the elements from the vector that are present in the specified collection.
18)	size()	It is used to get the number of components in the given vector.
19)	sort()	It is used to sort the list according to the order induced by the specified Comparator.

Java Vector Example:

Import java.util.*;

```
1. public class VectorExample{
2. public static voit565d main(String args[]){
3. //Create a vector
4. Vector<String>vec=new Vector<String>();
5. //Adding elements using add() method of List
6. vec.add("Tiger");
7. vec.add("Lion");
8. vec.add("Dog");
9. vec.add("Elephant");
10.//Adding elements using addElement() method of Vector
11.vec.addElement("Rat");
12.vec.addElement("Cat");
13.vec.addElement("Deer");
14.System.out.println("Elements are:"+vec);
15.}
16.}
```

Wrapper classes

The **wrapper class in Java** provides the mechanism *to convert primitive into object and object into primitive*.

Since J2SE 5.0, **autoboxing** and **unboxing** feature convert primitives into objects and objects into primitives automatically. The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.

Use of Wrapper classes in Java

Java is an object-oriented programming language, so we need to deal with objects many times like in Collections, Serialization, Synchronization, etc. Let us see the different scenarios, where we need to use the wrapper classes.

Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Autoboxing

The automatic conversion of primitive data type into its corresponding wrapper class is known as autoboxing, for example, byte to Byte, char to Character, int to Integer, long to Long, float to Float, boolean to Boolean, double to Double, and short to Short.

Wrapper class Example: Primitive to Wrapper

//Java program to convert primitive into objects

```
1. //Autoboxing example of int to Integer
2. public class WrapperExample1
3. public static void main(String args[]){
4. //Converting int into Integer
5. int a=20;
6. Integer i=Integer.valueOf(a);//converting int into nteger explicitly
7. Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally
8. System.out.println(a+" "+i+" "+j);
9. }}
```

Output:

```
20 20 20
```

Unboxing

The automatic conversion of wrapper type into its corresponding primitive type is known as unboxing. It is the reverse process of autoboxing.

Wrapper class Example: Wrapper to Primitive

//Java program to convert object into primitives

```
1. //Unboxing example of Integer to int
2. public class WrapperExample2{
3. public static void main(String args[]){
4. //Converting Integer to int
5. Integer a=new Integer(3);
6. int i=a.intValue();//converting Integer to int explicitly
7. int j=a;//unboxing, now compiler will write a.intValue() internally
8. System.out.println(a+" "+i+" "+j);
9. }}
```