A T M E
College of Engineering

Course Name	**Object Oriented Programming with Java**
Course Code	**BCS306A**
Contact Hours	**40**
CIE Marks	**50**
SEE Marks	**50**

Course Coordinator

Shrilakshmi Prasad
Assistant Professor
Dept of CSE,
ATMECE, Mysuru

To Understand

To Pass

To score

Simple English

Interest

Course Learning Objectives

- Learn fundamental features of object oriented language and JAVA
- Set up Java JDK environment to create, debug and run simple Java programs.
- Learn object oriented concepts using programming examples
- Study the concepts of importing of packages and exception handling mechanism
- Discuss the String Handling examples with Object Oriented concepts

Course Outcomes

- Explain the object-oriented concepts and JAVA
- Develop computer programs to solve real world problems in Java
- Develop simple GUI interfaces for a computer program to interact with users

ATME College of Engineering

Text Book

Herbert Schildt, Java The Complete Reference, 7th Edition, Tata McGraw Hill, 2007. (Chapters 2, 3, 4, 5, 6,7, 8, 9,10, 12,13,15)

Reference Books

Cay S Horstmann, "Core Java - Vol. 1 Fundamentals", Pearson Education, 10th Edition, 2016.

Raoul-Gabriel Urma, Mario Fusco, Alan Mycroft, "Java 8 in Action", Dreamtech Press/Manning Press, 1st Edition, 2014.

Question Paper Pattern

The question paper will have ten questions.

Each full Question consisting of 20 marks.

There will be 2 full questions (with a maximum of four sub questions) from each module.

Each full question will have sub questions covering all the topics under a module.

The students will have to answer 5 full questions, selecting one full question from each module.

Three Parameters Before Selecting a Language

Its a direct indication of how popular the programming language

Rank of the language

Usage of the language in IT Industry

How many new learners are trying to learn Java as there first programming language

How many online courses

Rank of the language

Go to Google

tiobe

https://https://www.tiobe.com/tiobe

C language used in Medical Science & Health Care

Programming Language	2020	2015	2010	2005	2000	1995	1990
Java	1	2	1	2	3	-	-
C	2	1	2	1	2	1	1

Usage of the Language

According to wikipedia

More than 10 Million Developers are using Java

3 Billion + devices make use of Java

Most Interested Programming Language to Learn

Go to Google

Pypl github

https://pypl.github.io/PYPL.html

Job Roles

Full Stack Developer

Desktop Developer

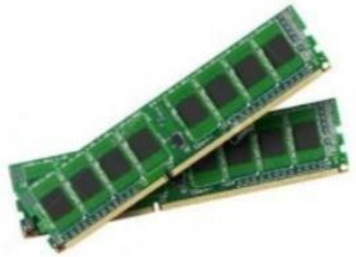
Introduction to Microprocessor

Before you learn any programming language, it is important for one to understand some of the basics about computer and what are the languages that a computer can understand.

A computer is a collection of hardware components.

Let us consider here few hardware components such as: Microprocessor RAM Hard disk Motherboard

Microprocessor


RAM

Hard disk

Motherboard

Introduction to Microprocessor

Microprocessor or CPU: A microprocessor is an electronic component that is used by a computer to do its work.

It is a central processing unit on a single integrated circuit chip containing millions of very small components including transistors, resistors, and diodes that work together.

They are created using a technology called as Semiconductor technology.

Semiconductor Technology?? Any device which is made up of transistors is referred to as working in Semiconductor Technology.

Introduction to Microprocessor

A transistor is a device that regulates current or voltage flow and acts as a switch or gate for electronic signals.

The transistors have three terminals emitter, base and collector.

There are two types of transistors: 1) NPN transistor. 2) PNP transistor.

Transistor NPN PNP Transistors can only store voltages.

There are two levels of voltages: Low Voltage referred to as → 0V
High Voltage referred to as→ 5V

Introduction to Microprocessor

If we see the same in Software engineer's view, he/she looks the two levels as:
Low level referred to as→ 0 High level referred to as→ 1

Therefore,in the perspective of a software engineer a Microprocessor or CPU can understand combinations of 0 and 1.

Programming Languages is the main medium for communicating between the computer systems.

A program is a collection of instructions that can be executed by a computer to perform a specific task.

There were several programming languages used to communicate with the Computer.

Case - 1:

The world’s first computer was invented in the year 1940’s.

During that time the task of a programmer was not simple.

For example, if they wanted microprocessor to perform any operation then they had to use combinations of 0’s and 1’s.

During this time all the programs where written in the language called as Machine Level Language.

It is one of the low-level programming languages.

Case - 1:

The language which machine's understand is what called as Machine Level Language.

Codes written in 1940's as To perform addition of two numbers: 0110110
To perform subtraction of two numbers: 1110111
To perform multiplication of two numbers: 1010101
To perform division of two numbers: 0100100

The machine level code was taken as input and given to the microprocessor as the machine understands the binary value code and it gives the output.

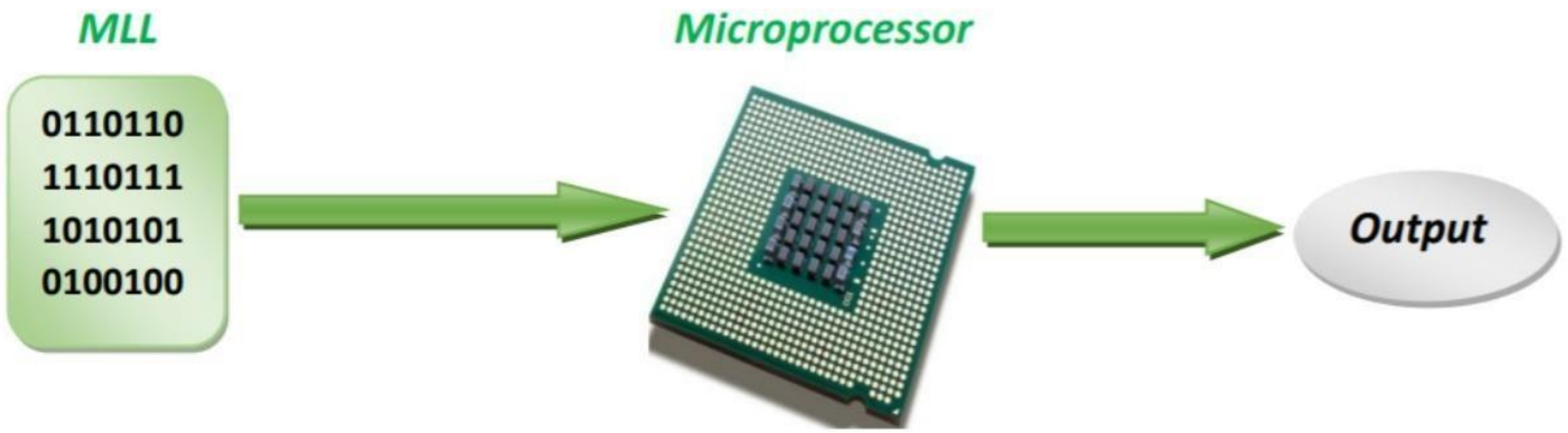
The main advantage of using Machine language is that there is no need of a translator to translate the code, as the Computer directly can understand.

ATME College of Engineering

Case - 1:

The disadvantage was, it was difficult for a programmer to write the code or remember the code in this type of language.

The languages that the machines understand are what called as Machine Level Language. 0110110 1110111 1010101 0100100



Case - 2:

The problem with Machine level code approach was decided to be changed in the year 1950's.

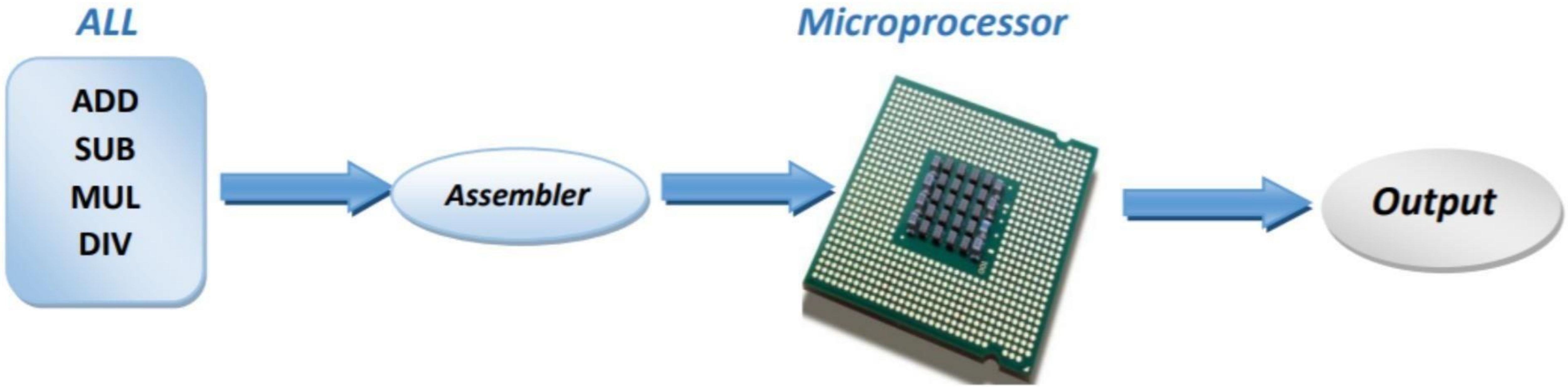
They thought that instead of writing a long sequence of 0's and 1's a single instruction can be given.
For example we use, Codes in 1950's written as To perform addition of two numbers: ADD
To perform subtraction of two numbers: SUB
To perform multiplication of two numbers: MUL
To perform division of two numbers: DIV

This approach of writing code is what called as Assembly Level Language.

Instead of using numbers like in Machine languages here we use words or names in English forms. ADD SUB MUL DIV

Case - 2:

An Assembler is software which takes Assembly Level Language (ALL) programs as input and converts it into Machine Level Language (MLL) program.



Case - 3:

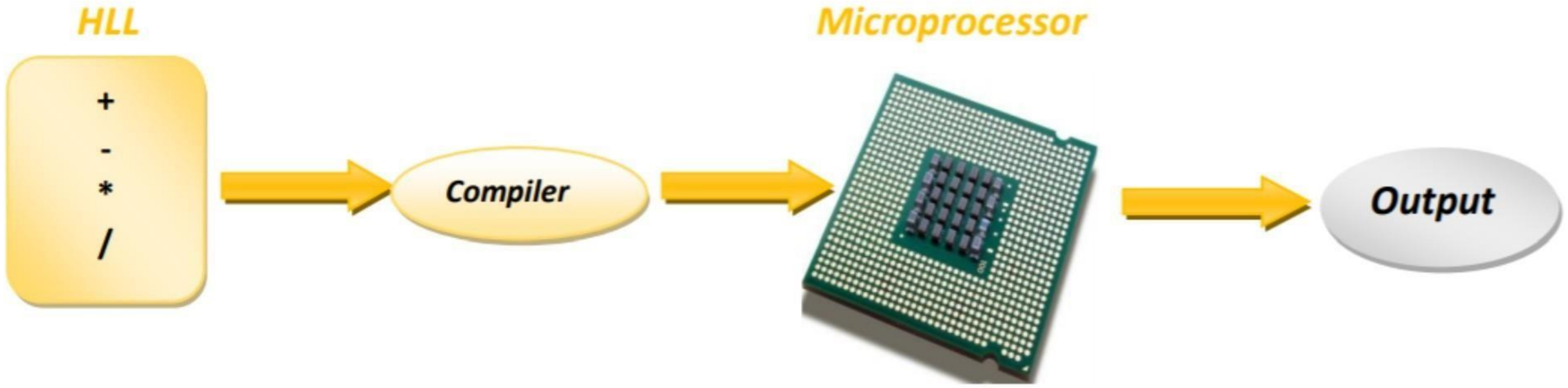
People always want the things to be simple and easier so, in 1960’s they came up with next type of language called High Level Programming Language.

High Level Languages are written in a form that is close to our human language,enabling the programmer to just focus on the problem being solved.

For example we use, Codes in 1960’s written as To perform addition of two numbers: +
To perform subtraction of two numbers: -
To perform multiplication of two numbers: *
To perform division of two numbers: /

Case - 3:

Compiler : A compiler is software which takes High Level Language (HLL) programs as input and converts it into Machine Level Language (MLL) program.



Platform dependency

One of the most important features of java is platform in dependency and to understand how java achieved platform in dependency, one must learn what is platform dependency.

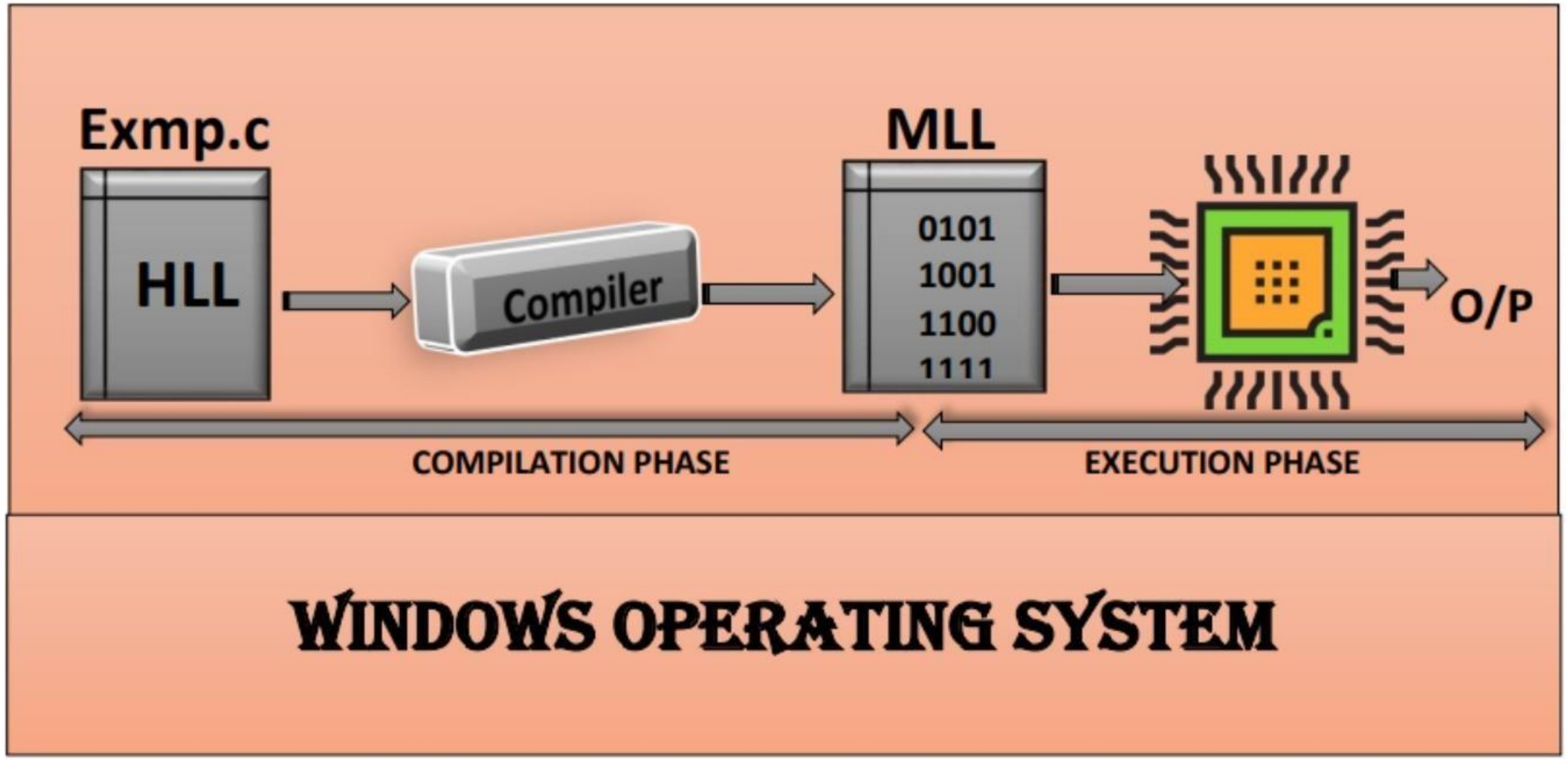
Platform is a combination of hardware and software. Hardware mainly refers to microprocessor and software refers to Operating system.

For example, platform in your computer could be the combination of i5 processor and windows OS or it could be i9 processor and mac OS as shown below. However, from Software engineer's perspective, platform only refers to OS.

Platform = Hardware + Software

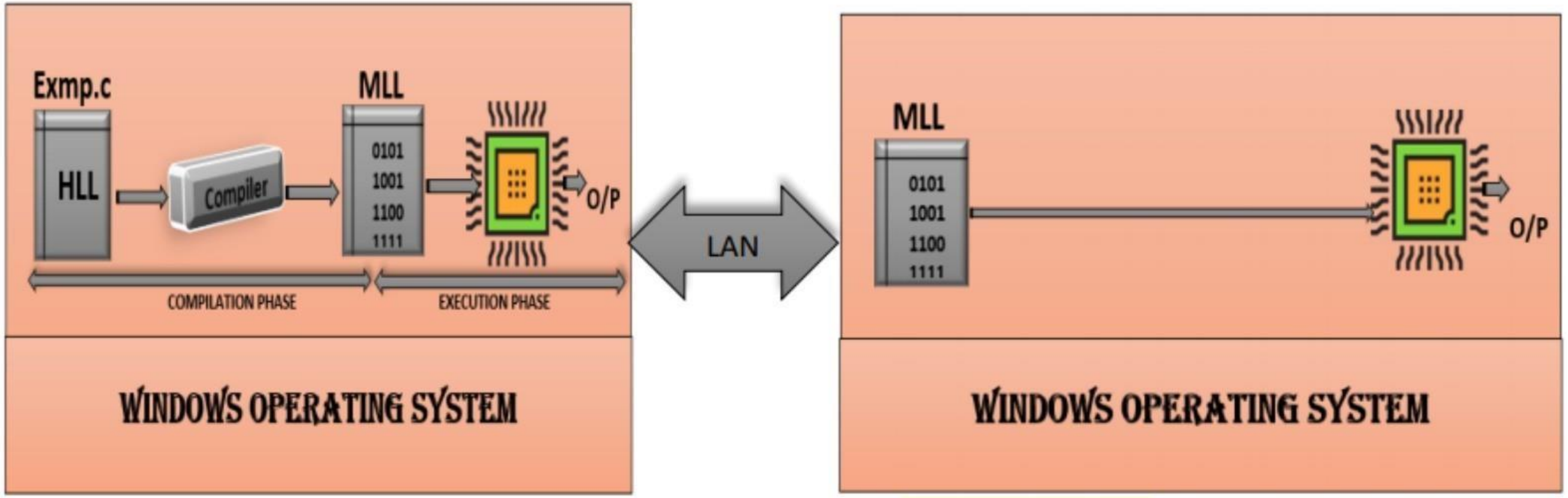
After getting to know what is platform let us now understand platform dependency by considering different cases of C language which is very much platform dependent.

Case - 1



In the above case, we considered a file of c programming language as the extension is exmp.c. Since C programming language is platform dependent programming language, its platform of execution and platform of compilation must be same. In the above case the platform of compilation and platform of execution is same which is windows operating system and hence we got the output.

Case - 2



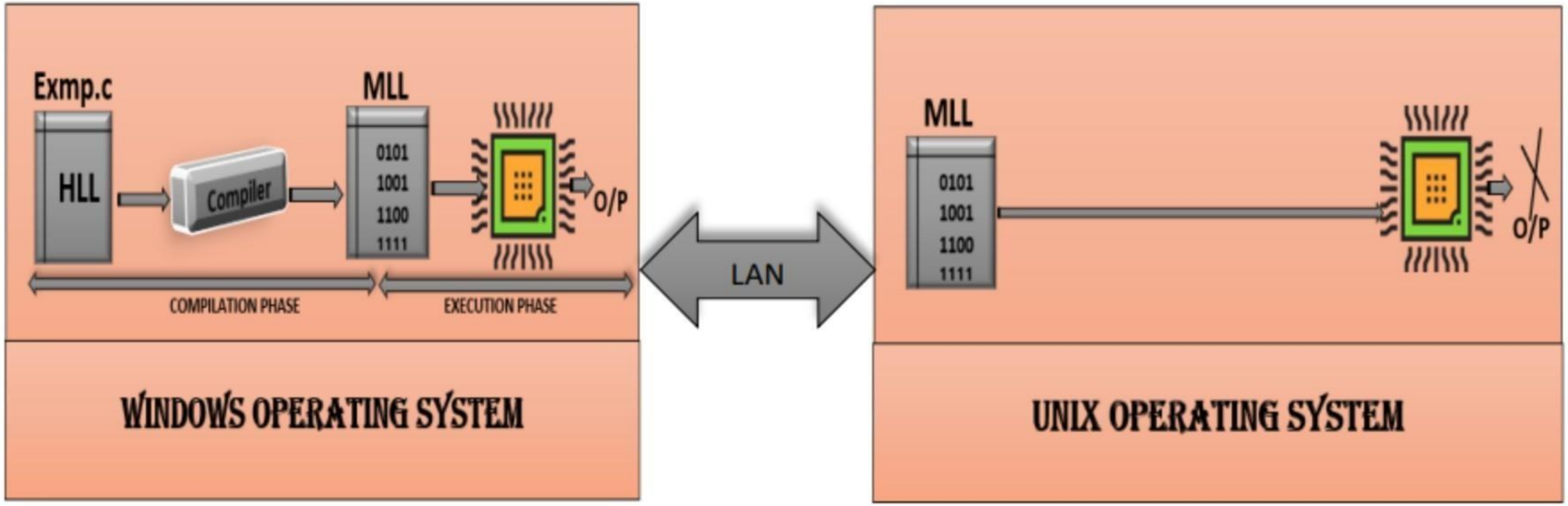
In the above case, we again considered a file of c programming language as the extension is exmp.c. In this case, we considered two computers having same operating system connected via LAN connection. If we now take a copy of MLL file from first computer and try to execute it on second computer, according to you will the file execute????

If incase you are still confused, just check two phases.

COMPILATION PHASE	EXECUTION PHASE
Windows	Windows

If you can refer the above table, compilation happened on windows and execution happened on windows. Since platform of compilation is same as that of platform of execution, the file gets executed which simply means you will get the output.

Case - 3


Exmp.c
HLL
Compiler
MLL
0101
1001
1100
1111
O/P
COMPILATION PHASE
EXECUTION PHASE
WINDOWS OPERATING SYSTEM
LAN
MLL
0101
1001
1100
1111
O/P
UNIX OPERATING SYSTEM

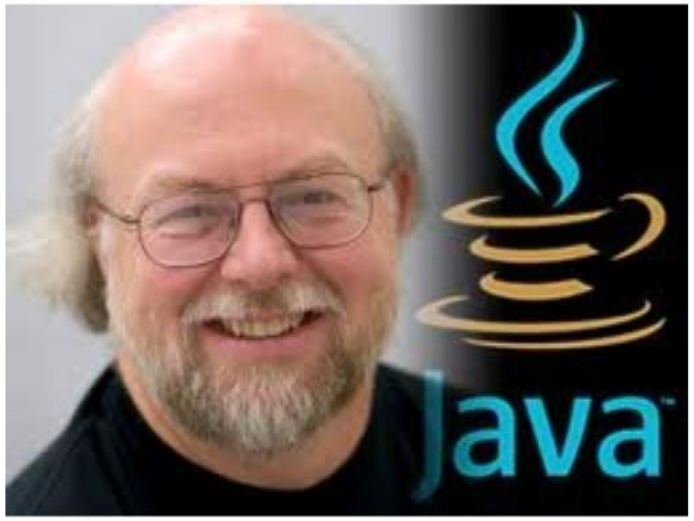
This case is same as that of case ii except the platform of second computer that is

COMPILATION PHASE	EXECUTION PHASE
WINDOWS	UNIX

If you can refer the above table, compilation happened on windows and execution happened on Unix. Since platform of compilation is not same as that of platform of execution, the file will not get executed which simply means you will not get the output.

The problem faced in C or C++ programming language is that their compiler directly converts High-level language code to machine level language code and if this code is copied on other type of operating system and tried to execute, it doesn't work because the code was written using platform dependent programming language.

To resolve this issue, JAMES GOSLING, the inventor of java introduced a programming language which was platform independent.



Let us now learn how java achieved platform in dependency

Let us assume you are writing code using java in your computer which has windows os. Since Machine understands machine level code not your high level code, conversion must happen.

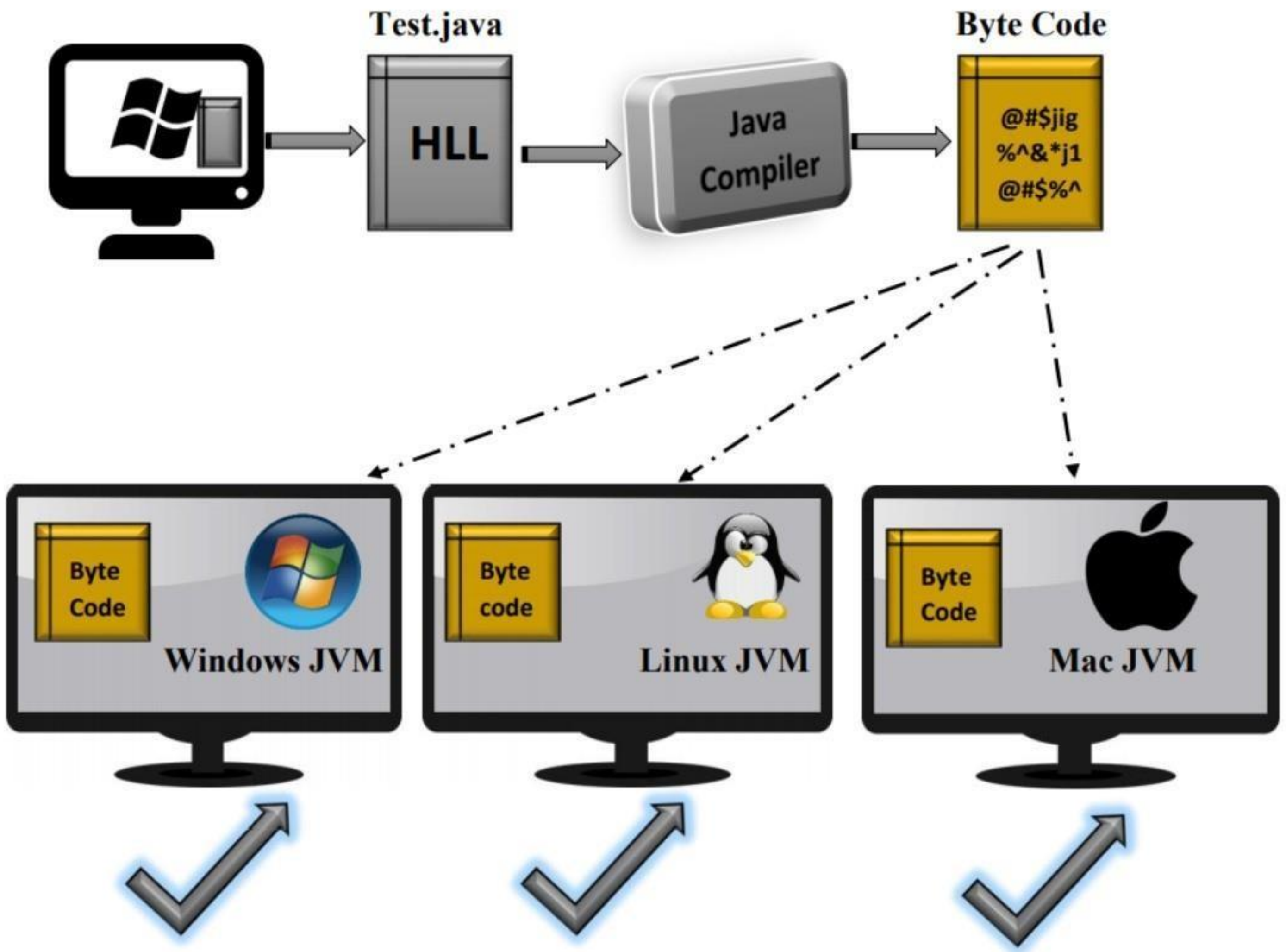
Let us see how exactly conversion happens in java.

Initially your HLL code is given as input to compiler but java compiler will not give MLL code as ouput like c and c++ compiler rather it takes HLL as input and gives a special type of code as output called as byte code which is platform independent. Byte code is neither hll code nor mll code, hence it is also referred to as intermediate code.

If you can recollect machine understands only MLL code but java compiler gave you byte code. To resolve this, James Gosling provided a software called as JVM(Java vitual Machine) which was platform dependent that is different OS have different JVM. Since you are writing code on windows OS, you will have to download windows compatible JVM . JVM will now convert byte code to machine level code which machine can easily understand.

In this way, java achieved platform in dependency using a special type of code which is byte code.

Have a look at the figure shown below which justifies the explanation given above.



Key Points

1)Java is a platform independent programming language.
2)Java Compiler takes high-level language as input and converts it into byte Code.
3) Byte code is neither HLL code not MLL code, it is in between and hence it is also called as intermediate code.
4) Byte code is Platform independent where as machine level code is platform dependent.
5) JVM is a software which takes byte code as input and converts it into MLL code.
6) JVM is platform dependent.
7) Different OS have different JVM.
8) Java achieved platform In dependency by making use of byte code as it is a code which is platform Independent.
9) Byte code is stored in class file.
10) Platform in dependency is also referred as Portability.
11) Java is a portable programming language.
12) One of the disadvantage of java is, it is slow in execution when compared to c and c++. This is due to the extra step involved in the conversion of HLL code to MLL code which is converting HLL to byte code and then byte code to MLL code.

First Java Program

In Java all methods must be enclosed within a class.

JVM

Java Class Library

JRE

“ JDK ”

Development Tools

First Java Program

Two Approaches for making main method accessible to JVM

Using object of the enclosing class.

Using static keyword

First Java Program

Static methods can be accessed with out object creation

Public makes the main() visible to the JVM

Static makes the main() accessible without object creation

A T M E College of Engineering

If statement:

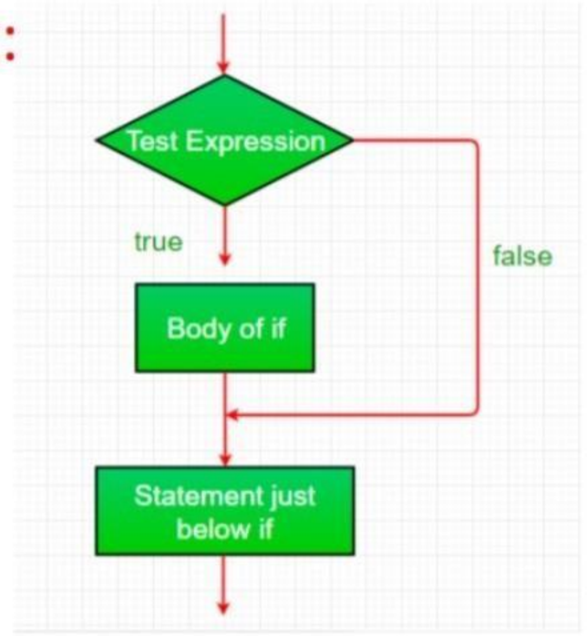
is the most simple decision making statement.

It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.

Syntax:

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

Flowchart:



Let us now learn Loops

“ You might be now wondering why loops? ”

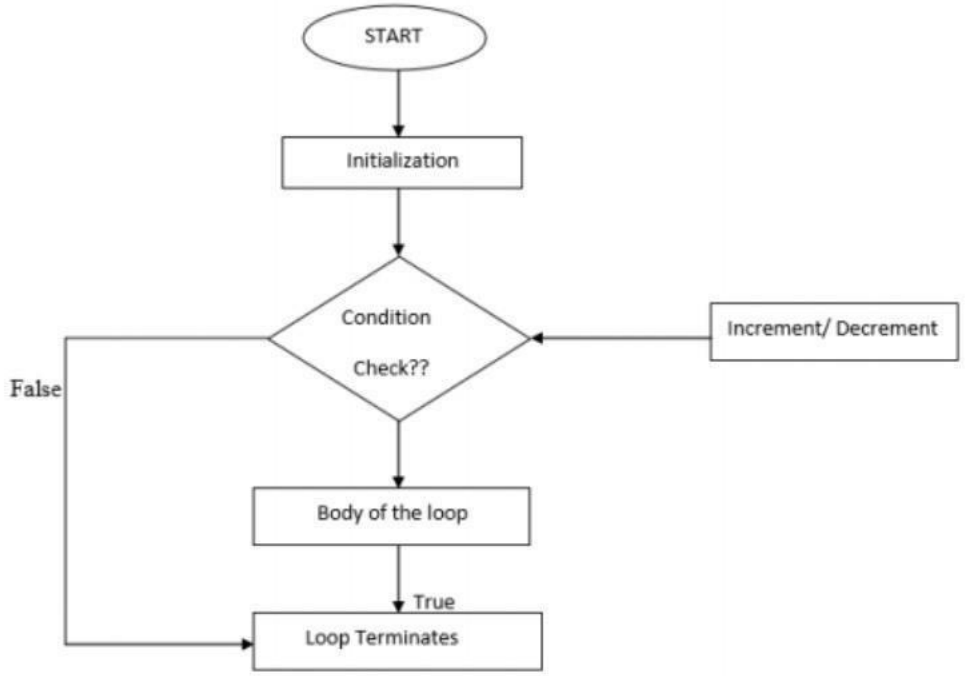
Looping in programming languages is a feature which facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to true.

Still confused?

Let us now learn Loops

In simple words if you are doing the same operation multiple times, you can make use of loops.

Let us look at its flow chart:



Let us now learn Loops

Consider the example given below without using for loop

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

As you can see in the above code you are printing JAVA 5 times that is doing the same operation multiple times

First Java Program

Let us now make use of for-loop and write the code:

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

You can now see the way we made use of for-loop and reduced the size of code.

Both the codes gave us the same output but the code was written effectively using for-loop.

Isn't that great how loops reduce the size of code

Using Blocks of Code

Java allows two or more statements to be grouped into blocks of code, also called code blocks.

This is done by enclosing the statements between opening and closing curly braces.

Once a block of code has been created, it becomes a logical unit that can be used any place that a single statement can.

For example, a block can be a target for Java's if and for statements. Consider this if statement:

```
if(x < y)
{ // block begins
x = y;
y = 0;
} // block ends here
```

The main reason for the existence of blocks of code is to create logically inseparable units of code.

Lexical Issues

Java programs are a collection of whitespace, identifiers, literals, comments, operators, separators, and keywords.

Whitespace

In Java, whitespace is a space, tab or newline. Usually, a space is used to separate tokens; tab and newline are used for indentation.

Identifiers

Identifiers are used for class names, method names, and variable names.

An identifier may be any sequence of uppercase and lowercase letters, numbers, or the underscore and dollar-sign characters.

They must not begin with a number. As Java is case-sensitive,
Avg is a different identifier than avg.

Examples of valid identifiers: Avg, sum1, $x, sum_sq etc.

Examples of invalid identifiers: 2sum, sum-sq, x/y etc.

Literals

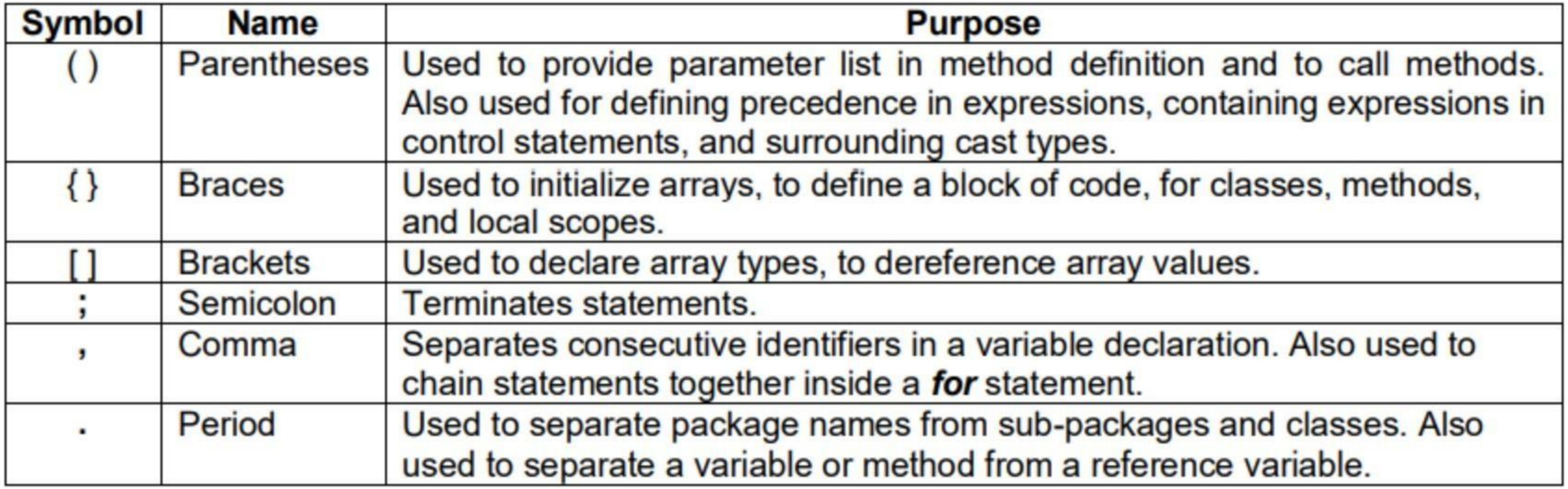
A constant value in Java is created by using a literal representation of it. For example, 25 (an integer literal), 4.5 (a floating point value), ‘p’ (a character constant, “Hello World” (a string value).

Comments

There are three types of comments defined by Java. Two of these are well-know viz.
Single-line comment (starting with //),
Multiline comment (enclosed within /* and */).
The third type of comment viz. documentation comment is used to produce an HTML file that documents your program.
The documentation comment begins with a /** and ends with a */.

Separators

In Java, there are a few characters that are used as separators.
The most commonly used separator in Java is the semicolon which is used to terminate statements.



Symbol	Name	Purpose
( )	Parentheses	Used to provide parameter list in method definition and to call methods. Also used for defining precedence in expressions, containing expressions in control statements, and surrounding cast types.
{ }	Braces	Used to initialize arrays, to define a block of code, for classes, methods, and local scopes.
[ ]	Brackets	Used to declare array types, to dereference array values.
;	Semicolon	Terminates statements.
,	Comma	Separates consecutive identifiers in a variable declaration. Also used to chain statements together inside a ***for*** statement.
.	Period	Used to separate package names from sub-packages and classes. Also used to separate a variable or method from a reference variable.

Keywords

There are 50 keywords currently defined in the Java language as shown in the following Table.

These keywords, combined with the syntax of the operators and separators, form the foundation of the Java language.

These keywords cannot be used as names for a variable, class, or method.

Abstract	assert	boolean	break	byte	case	catch	char	class	const
Continue	default	goto	do	double	else	enum	extends	final	finally
Float	for	if	implements	import	instanceof	int	interface	long	native
New	package	private	protected	public	return	short	static	strictfp	super
Switch	synchronized	this	throw	throws	transient	try	void	while	

The keywords const and goto are reserved but are rarely used. In addition to the keywords, Java reserves the following: true, false, and null.
These are values defined by Java. You may not use these words for the names of variables, classes and so on.

Arrays

Array is a collection of related items of same data type.

Many items of an array share common name and are accessed using index.

Array can be one dimensional or multi-dimensional.

One Dimensional Arrays

It is a list of related items. To create 1-d array, it should be declared as –

```
type arr_name[];
```

type determines the data type of elements of arr_name.

In Java, the above declaration will not allocate any memory. That is, there is no physical existence for the array now.

To allocate memory, we should use new operator as follows:

```
arr_name=new type[size];
```

size indicates number of elements in an array. The new keyword is used because, in Java array requires dynamic memory allocation.

```
type arr_name[]=new type[size];
```

Multidimensional Arrays

Multidimensional arrays are arrays of arrays. Here, we will discuss two dimensional arrays in Java.

The declaration of 2-d array is as follows –

```
type arr_name[][]=new type[row_size][col_size];
```

row_size and col_size indicates number of rows and columns of 2-d arrays. In other words, row size indicates number of 1-d arrays and col_size indicates size of each of such 1-d array.

The Primitive Types

Java defines eight primitive (or simple) data types.

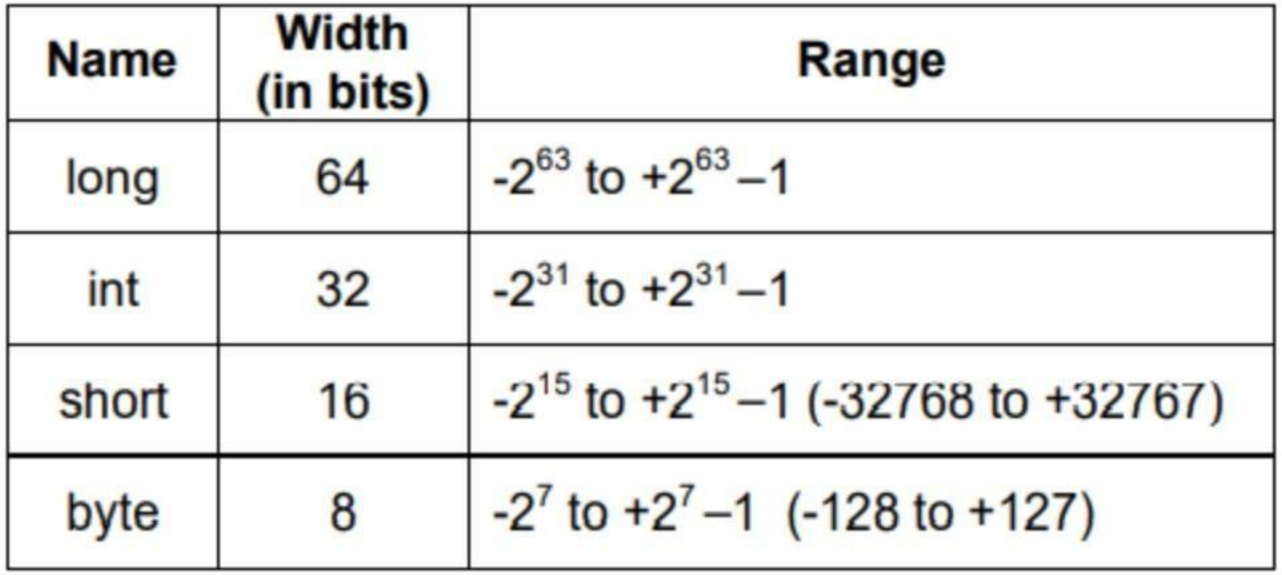
byte, short, int, long :	belonging to Integers group involving whole-valued signed numbers
char :	belonging to Character group representing symbols in character set like alphabets, digits, special characters etc
float, double :	belonging to Floating-point group involving numbers with fractional part
boolean :	belonging to Boolean group, a special way to represent true/false values.

Integers

Java defines four integer types viz. byte, short, int and long.

All these are signed numbers and Java does not support unsigned numbers.

The width of an integer type should not be thought of as the amount of storage it consumes, but rather as the behavior it defines for variables and expressions of that type.



Name	Width (in bits)	Range
long	64	-2^{63} to $+2^{63}-1$
int	32	-2^{31} to $+2^{31}-1$
short	16	-2^{15} to $+2^{15}-1$ (-32768 to +32767)
byte	8	-2^{7} to $+2^{7}-1$ (-128 to +127)

byte

This is the smallest integer type.

Variables of type byte are especially useful when you are working with a stream of data from a network or file.

They are also useful when you are working with raw binary data that may not be directly compatible with Java's other built-in types.

Byte variables are declared by use of the byte keyword.

```
byte b, c;
```

short

It is probably the least-used Java type.

Here are some examples of short variable declarations:

```
short s;
short t;
```

int

The most commonly used integer type is int.

In addition to other uses, variables of type int are commonly employed to control loops and to index arrays.

Although you might think that using a byte or short would be more efficient than using an int in situations in which the larger range of an int is not needed, this may not be the case. The reason is that when byte and short values are used in an expression they are promoted to int when the expression is evaluated.

Therefore, int is often the best choice when an integer is needed.

Floating –Point Types

Floating-point (or real) numbers are used when evaluating expressions that require fractional precision.

Java implements the standard (IEEE–754) set of floating-point types and operators.

There are two kinds of floating-point types, float and double, which represent single- and double-precision numbers, respectively.

Name	Width (in bits)	Range
double	64	4.9e–324 to 1.8e+308
float	32	1.4e–045 to 3.4e+038

float

The type float specifies a single-precision value that uses 32 bits of storage.

Single precision is faster on some processors and takes half as much space as double precision, but will become imprecise when the values are either very large or very small.

Variables of type float are useful when you need a fractional component, but don’t require a large degree of precision.

For example, float can be useful when representing currencies, temperature etc.

double

Double precision is actually faster than single precision on some modern processors that have been optimized for high-speed mathematical calculations.

All transcendental math functions, such as sin( ), cos( ), and sqrt( ), return double values.

When you need to maintain accuracy over many iterative calculations, or are manipulating large-valued numbers, double is the best choice.

Characters

In Java, char is the data type used to store characters.

In C or C++, char is of 8 bits, whereas in Java it requires 16 bits.

Java uses Unicode to represent characters. Unicode is a computing industry standard for the consistent encoding, representation and handling of text expressed in many languages of the world. Unicode has a collection of more than 109,000 characters covering 93 different languages like Latin, Greek, Arabic, Hebrew etc. That is why, it requires 16 bits. The range of a char is 0 to 65,536.

The standard set of characters known as ASCII still ranges from 0 to 127 as always, and the extended 8-bit character set, ISO-Latin-1, ranges from 0 to 255.

Since Java is designed to allow programs to be written for worldwide use, it makes sense that it would use Unicode to represent characters.

Though it seems to be wastage of memory as the languages like English, German etc. can accommodate their character set in 8 bits, for a global usage point of view, 16-bits are necessary.

Boolean

For storing logical values (true and false), Java provides this primitive data type.

Boolean is the output of any expression involving relational operators.

For control structures (like if, for, while etc.) we need to give boolean type.

In C or C++, false and true values are indicated by zero and a non-zero numbers respectively. And the output of relational operators will be 0 or 1. But, in Java, this is not the case.

Java is a Strongly Typed Language

A strongly-typed programming language is one in which each type of data (such as integer, character, hexadecimal, packed decimal, and so forth) is predefined as part of the programming language and all constants or variables defined for a given program must be described with one of the data types.

Certain operations may be allowable only with certain data types. In other words, every variable has a type, every expression has a type, and every type is strictly defined. And, all assignments, whether explicit or via parameter passing in method calls, are checked for type compatibility.

There are no automatic conversions or conversions of conflicting types as in some languages.
The Java compiler checks all expressions and parameters to ensure that the types are compatible. Any type mismatches are errors that must be corrected before the compiler will finish compiling the class.
These features of Java make it a strongly typed language.

The Java Class Libraries

The sample programs discussed make use of two of Java’s built-in methods:println() and print( ).

As mentioned, these methods are members of the System class, which is a class predefined by Java that is automatically included in your programs. I

In the larger view, the Java environment relies on several built-in class libraries that contain many built-in methods that provide support for such things as I/O, string handling, networking, and graphics. The standard classes also provide support for windowed output.

Thus, Java is a combination of the Java language itself, plus its standard classes. The class libraries provide much of the functionality that comes with Java.

Variables

The variable is the basic unit of storage. A variable is defined by the combination of an identifier, a type, and an optional **initializer.**

Declaring a Variable

In Java, 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] ...] ;

The type is any of primitive data type or class or interface. The identifier is the name of the variable. We can initialize the variable at the time of variable declaration. To declare more than one variable of the specified type, use a comma-separated list.

```
int a, b=5, c;
byte z = 22;
double pi = 3.1416;
char x = '$'
```

Dynamic Initialization

The preceding examples have used only constants as initializers, Java allows variables to be initialized dynamically, using any expression valid at the time the variable is declared.

```
int a=5, b=4;
int c=a*2+b;
```

The Scope and Lifetime of Variables

A variable in Java can be declared within a block. A block is begun with an opening curly brace and ended by a closing curly brace. A block defines a scope which determines the accessibility of variables and/or objects defined within it. It also determines the lifetime of those objects.

A Closer Look at Literals

A literal is the source code representation of a fixed value. In other words, by literal we mean any number, text, or other information that represents a value. Literals are represented directly in our code without requiring computation.

Integer Literals

Integers are the most commonly used type in the typical program. Any whole number value is an integer Literal.
For example, 1, 25, 33 etc. These are all decimal values, having a base 10.

With integer literals we can use octal (base 8) and hexadecimal (base 16) also.

Octal values are denoted in Java by a leading zero. Normal decimal numbers cannot have a leading zero. Thus, a value 09 will produce an error from the compiler, since 9 is outside of octal's 0 to 7 range.

Hexadecimal constants denoted with a leading zero-x, (0x or 0X). The range of a hexadecimal digit is 0 to 15, so A through F (or a through f) are substituted for 10 through 15.

Floating-point numbers represent decimal values with a fractional component. They can be expressed in either standard or scientific notation. Standard notation consists of a whole number component followed by a decimal point followed by a fractional component. For example, 2.0, 3.14159, and 0.6667 represent valid standard-notation floating-point numbers. Scientific notation uses a standard-notation, floating-point number plus a suffix that specifies a power of 10 by which the number is to be multiplied.

The exponent is indicated by an E or e followed by a decimal number, which can be positive or negative. Examples include 6.022E23, 314159E–05, and 2e+100

Floating-point literals in Java default to double precision. To specify a float literal, you must append an F or f to the constant. You can also explicitly specify a double literal by appending a D or d. Doing so is, of course, redundant. The default double type consumes 64 bits of storage, while the less-accurate float type requires only 32 bits.

Boolean Literals

Boolean literals are simple. There are only two logical values that a boolean value can have, true and false. The values of true and false do not convert into any numerical representation. The true literal in Java does not equal 1, nor does the false literal equal 0. In Java, they can only be assigned to variables declared as boolean, or used in expressions with Boolean operators.

Type Conversion and Casting

It is quite common in a program to assign value of one type to a variable of another type.

If two types are compatible, Java performs implicit type conversion. For example, int to long is always possible. But,
whenever the types at two sides of an assignment operator are not compatible, then Java will not do the conversion implicitly. For that, we need to go for explicit type conversion or type casting.

Java's Automatic Conversions

When one type of data is assigned to another type of variable, an automatic type conversion will take place if the following two conditions are met:

The two types are compatible.

The destination type is larger than the source type.

Casting Incompatible Types

Although the automatic type conversions are helpful, they will not fulfill all needs.

For example, what if you want to assign an int value to a byte variable? This conversion will not be performed automatically, because a byte is smaller than an int. This kind of conversion is sometimes called a narrowing conversion, since you are explicitly making the value narrower so that it will fit into the target type.

To create a conversion between two incompatible types, we must use a cast. A cast is simply an explicit type conversion. It has this general form:

```
(target-type) value
```

Here, target-type specifies the desired type to convert the specified value to.

When a floating-point value is assigned to an integer type, the fractional component is lost. And such conversion is called as truncation (narrowing).

Type Promotion Rules

Java defines several type promotion rules that apply to expressions. They are as follows:

All byte, short, and char values are promoted to int.

If one operand is a long, the whole expression is promoted to long.

If one operand is a float, the entire expression is promoted to float.

If any of the operands is double, the result is double.

Java - Basic Operators

Java provides a rich set of operators to manipulate variables.

We can divide all the Java operators into the following groups −

Arithmetic Operators

Relational Operators

Bitwise Operators

Logical Operators

Assignment Operators

The Arithmetic Operators

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra.

+ (Addition)	Adds values on either side of the operator.	A + B will give 30
- (Subtraction)	Subtracts right-hand operand from left-hand operand.	A - B will give -10
* (Multiplication)	Multiplies values on either side of the operator.	A * B will give 200

/ (Division)	**Divides left-hand operand by right-hand operand.**	**B / A will give 2**
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.	B % A will give 0
++ (Increment)	Increases the value of operand by 1.	B++ gives 21
-- (Decrement)	Decreases the value of operand by 1.	B-- gives 19

The Relational Operators

== (equal to)	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!= (not equal to)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.

> (greater than)	**Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.**	**(A > B) is not true.**
< (less than)	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.

>= (greater than or equal to)	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<= (less than or equal to)	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

The Bitwise Operators

Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte.

Bitwise operator works on bits and performs bit-by-bit operation.

& (bitwise and)	**Binary AND Operator copies a bit to the result if it exists in both operands.**	**(A & B) will give 12 which is 0000 1100**
| (bitwise or)	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^ (bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001

~ (bitwise compliment)	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<< (left shift)	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000

>> (right shift)	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111

>>> (zero fill right shift)	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

The Logical Operators

&& (logical and)	**Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.**	**(A && B) is false**
|| (logical or)	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A || B) is true

! (logical not)	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true

The Assignment Operators

Following are the assignment operators supported by Java language

=	**Simple assignment operator. Assigns values from right side operands to left side operand.**	**C = A + B will assign value of A + B into C**
+=	Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C + A

-=	**Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand.**	**C -= A is equivalent to C = C – A**
*=	Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A

/=	**Divide AND assignment operator. It divides left operand with the right operand and assign the result to left operand.**	**C /= A is equivalent to C = C / A**
%=	Modulus AND assignment operator. It takes modulus using two operands and assign the result to left operand.	C %= A is equivalent to C = C % A

There are few other operators supported by Java Language.

Conditional Operator (? :)

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

The operator is written as

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

A T M E
College of Engineering