
Chapter-1**Java Programming Fundamentals****History of JAVA:**

1. In 1990, James Gosling was given a task of creating projects to control consumer electronics. Gosling and his team Mike Sheridan, and Patrick Naughton at Sun Microsystems started designing their software using C++ because of its Object-oriented nature. Gosling, however, quickly found that C++ was not suitable for this project. They faced problems due to program bugs like memory leak, dangling pointer, multiple inheritance and platform dependent.
2. Gosling decided that he would develop his own, simplified computer language to avoid all the problems faced in C++.
3. Gosling kept the basic syntax and object oriented features of the C++ language to designing a new language.
4. He completed and named “OAK” in 1991 at Sun Microsystems.
5. Later Sun Microsystems was discovered that the name “OAK” was already claimed, they changed the name to “JAVA” in 1995.
6. The Java team realized that the language they had developed would be perfect for web programming because the world wide web had transformed the text-based internet into a graphic rich environment. Then team came up with the concept of web applets, small programs that could be included in web pages, and even went so far as to create a complete web browser that demonstrate the language’s power.
7. The new language was quickly embraced as a powerful tool for developing internet applications. Support for java was added in the Netscape (Web Browser on UNIX) and in the Internet Explorer.

Java’s Contribution to the Internet:

- Java in turn had a profound effect on the internet.
- Java innovated a new type of networked program called applet.
- Java also addresses some issue like portability, Security, JVM.

Java applets:

An applet is a special kind of java program that is designed to be transmitted over the Internet and automatically executed by java web browser.

Security

Java achieved this protection by confining an applet to the java execution environment and not to allowing it access to other parts of the computer.

Benefits of Using JAVA:

- Java programming language is very simple and object oriented. It is easy to learn and taught in many colleges and universities.
- Java applications run inside JVM, and now all major operating systems are able to run Java including Windows, Mac and UNIX.
- **Write once, run anywhere:** A Java application runs on all Java platforms.
- Java is very secure. Only Java applications that have permission can access the resources of the main computer. Thus, the main computer is protected from virus attackers and hackers.
- Java technologies have been improved by community involvement. It is suitable for most types of applications, especially complex systems that are used widely in network and distributed computing.

Features of JAVA:

1. **Simple:** Syntax is based on C and C++, No pointers, No goto, No operator overloading, No Preprocessor, No global variables.
2. **Object-Oriented:** Uses OOPs concepts (Inheritance, Polymorphism, Encapsulation, etc)
3. **Portable - Platform Independent – Architecture Neutral :**
 - **“Write once, Run anywhere”**
 - Runs on any platform that has JVM
4. **Secured:** No explicit pointer, Programs run inside JVM.
5. **Robust:**
 - Java uses the automatic garbage collection that prevents memory leaks.
 - Java is strictly typed language, hence Error free.
6. **Dynamic:**
 - Java loads in classes as they are needed.
 - JVM is capable of linking dynamic new classes, methods and objects.
7. **Compiled and Interpreted:**
 - Java code is compiled to byte code.
 - Byte code are interpreted on any platform by JVM.
 - Java programs can be shared over the internet.
8. **High Performance:**
 - Byte code are highly optimized.
 - JVM execute Byte code much faster.
9. **Distributed:** Java programs can be shared over the internet

10. Multithreaded:

- Multithreading means handling more than one job at a time.
- The main advantage of multi-threading is that it shares the same memory.

The Key Attributes of Object-Oriented Programming:

Object oriented programming took the ideas of structured programming and combined them with several new concepts. In the most general sense, a program can be organized in one of two ways:

- Around its code (What is happening)
- Around its data (What is being affected)

To support the principles of OOP, all OOP languages, including java, have 3 traits in common:

1. Encapsulation
2. Polymorphism
3. Inheritance

Encapsulation:

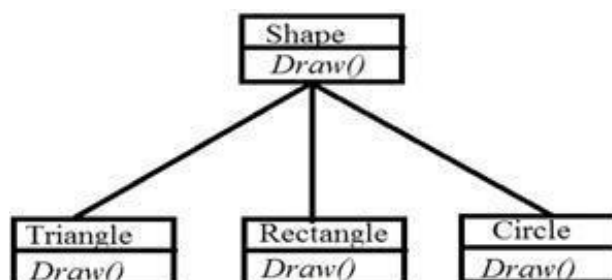
It is a mechanism that binds together code and the data it manipulates, and that keeps both safe from outside interference and misuse. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as **data hiding**.

Example:

```
class Employee
{
    int eid;
    string name;
    float salary
    void read();
    int getSalary();
}
```

Polymorphism:

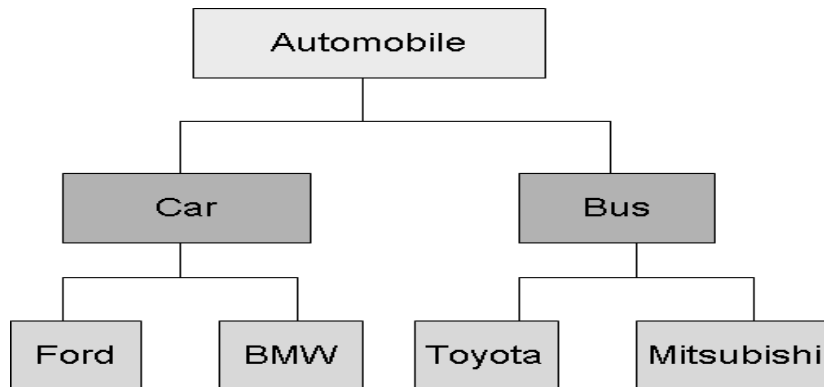
It is derived from 2 greek words: “**poly**” and “**morphs**”. The word “poly” means many and “morphs” means forms. So polymorphism means many forms. It is defined as that allows one interface to access a general class of actions or same name different operations.



Inheritance:

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. This is important because it supports the concepts of hierarchical classification. Without the use of hierarchies, each object would have to explicitly define all of its characteristics.

Using inheritance, an object need only define those qualities that make it unique within its class. It can inherit its general attributes from its parent. Thus, it is the inheritance mechanism that makes it possible for one object to be a specific instance of a more general case.

**The Java Development Toolkit:**

JDK (Java Development Kit) is provided by Sun Microsystem which is used to develop applications in Java. JDK contains set of libraries, APIs and Java Virtual Machine (JRE). A complete package to develop and run Java based applications and applets.

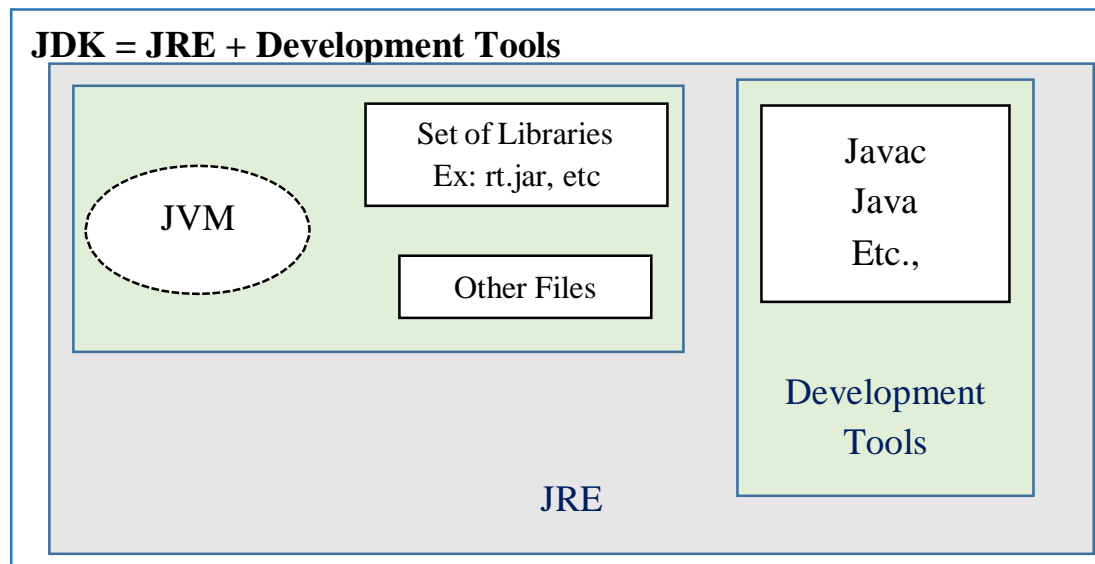


Figure: JDK Toolkit Contains

JDK is collection of tools like.

- **To compile - javac**
- **To execute - java**
- **To debug**
- **To document, etc.**

JVM

- JVM stands for Java Virtual Machine.
- JVM is a virtual machine that provides runtime environment to execute java byte code(*.class).
- It is the medium which compile Java code(*.java) to bytecode(*.class) which get interpret on different machine and hence it makes it Platform/Operating system independent.

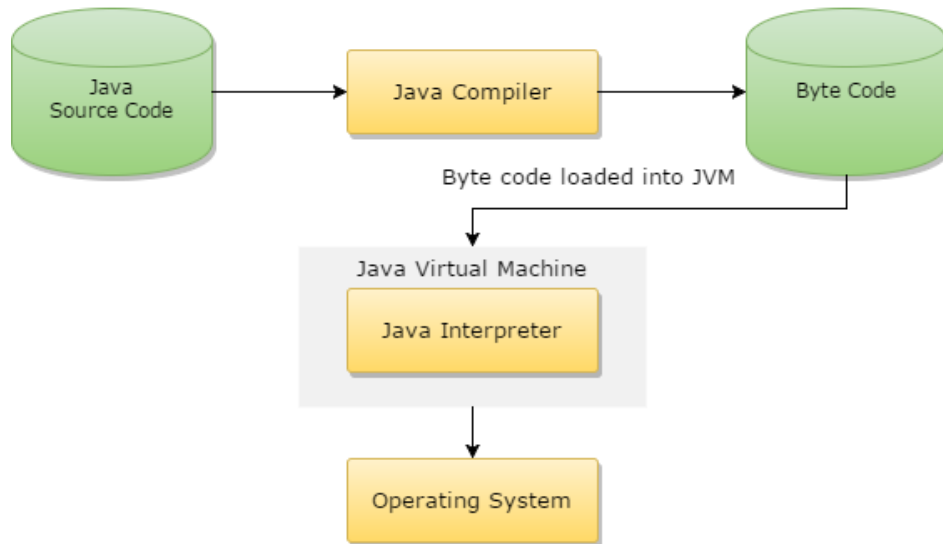


Figure: Diagram of JVM

JIT or Just-in-time compiler:

- JIT is a part of the JVM.
- It converts the selected portions of bytecode are compiled into the native machine code in order to speed up the execution time.
- In JVM, Java code is compiled to bytecode. This bytecode gets interpreted on different machines, hence java is both compiled as well as interpreted language.

What are Java Bytecode?

- Bytecode is a highly optimized set of instructions designed to be executed by the JVM.
- Bytecode is a machine language of the JVM
- Later bytecode translated into native code.

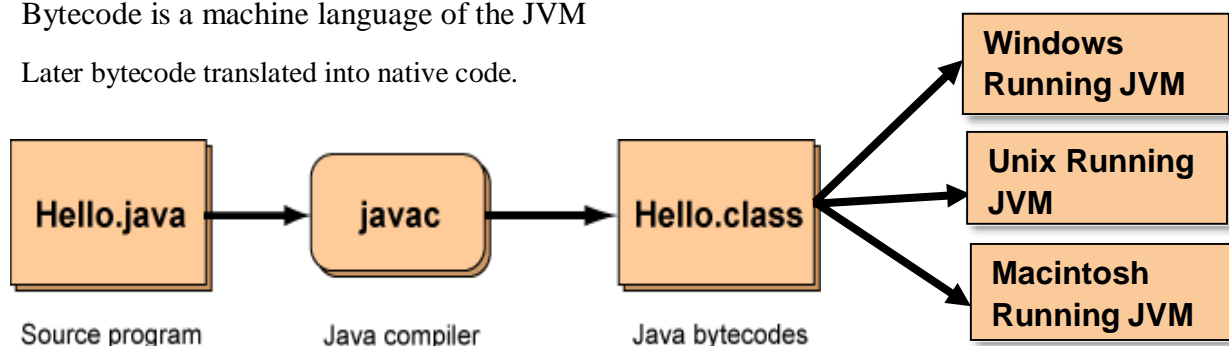


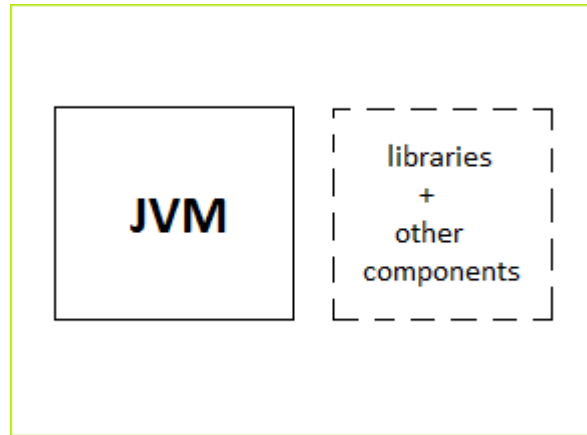
Figure: How Java compiler converts Java source into Java bytecodes

The Java compiler reads Java language source (*.java) files, translates the sources into Java bytecode and places the bytecodes into class (*.class) files. The compiler generates one class file per class in the source.

JRE

The **Java Runtime Environment (JRE)**, also known as Java Runtime, is part of the Java Development Kit (JDK), a set of programming tools for developing Java applications.

The Java Runtime Environment (JRE) provides the libraries, the Java Virtual Machine, and other components to run applets and applications written in the Java programming language. JRE does not contain tools and utilities.



A First Simple Program:

Let us look at a simple java program.

```
class Example{
    public static void main(String[] args)
    {
        System.out.println("Welcome to JAVA Class!!!");
    }
}
```

Where,

- ❖ **class:** class keyword is used to declare classes in Java
- ❖ **public:** It is an access specifier. Public means this function is visible to all.
- ❖ **static:** static is again a keyword used to make a method static. To execute a static function you do not have to create an Object of the class. The **main()** method here is called by JVM, without creating any object for class.
- ❖ **void:** It is the return type, meaning this method will not return anything.
- ❖ **main:** main() method is the most important method in a Java program. This is the method which is executed, hence all the logic must be inside the main() method. If a java class is not having a main() method, it causes compilation error. Multiple main() methods in the same class not allowed .
- ❖ **String[] args:** This represents an array whose type is String and name is args.
- ❖ **System.out.println:** This is used to print anything on the console like *printf* in C language.

You will follow these 3 steps to execute the java program:

- i. Enter the program
- ii. Compile the program
- iii. Run the program

i. Enter the program:

Open a text editor and write the code as above. Then save the file as **Example.java**

ii. Compile the program

Open command prompt and go to the directory where you saved your java program. Then, you must compile it using **javac** as shown here:

```
javac Example.java
```

This command will call the Java Compiler asking it to compile the specified file. If there are no errors in the code the command prompt will take you to the next line.

iii. Run the program

To run the program, you must use **java**. Now type as shown here:

```
java Example
```

on command prompt to run your program. Then, you will be able to see **Welcome to Java Class!!!** printed on your command prompt.

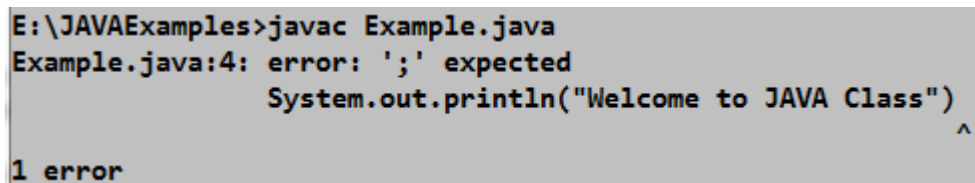
Handling Syntax Errors:

If you enter something incorrectly into your program, compiler will report a Syntax error messages.

Example:

```
class Example{
    public static void main(String[] args)
    {
        System.out.println("Welcome to JAVA Class!!!")
    }
}
```

When you compiled, it will report a syntax error message that



```
E:\JAVAEamples>javac Example.java
Example.java:4: error: ';' expected
        System.out.println("Welcome to JAVA Class")
                                   ^
1 error
```

The Java Keywords:

In the **Java** programming language, **50 keyword** or **reserved words** are currently defined that have a predefined meaning in the language; because of this, programmers cannot use **keywords** as names for variables, methods, classes, or as any other identifier.

- The keywords **const** and **goto** are reserved but not used.
- In addition to the keywords, java reserves the following: true, false and null. These are values defined by java.
- assert added in 1.4 and enum added in 1.5

List of Java Keywords

abstract	default	if	package	synchronized
assert	do	implements	private	this
boolean	double	import	protected	throw
break	else	instanceof	public	throws
byte	extends	int	return	transient
case	false	interface	short	true
catch	final	long	static	try
char	finally	native	strictfp	void
class	float	new	super	volatile
const	for	null	switch	while
continue	goto			

Identifiers in Java:

An identifier is a name given to a method, variable, or any other user-defined item that you want to identify in program.

Rules:

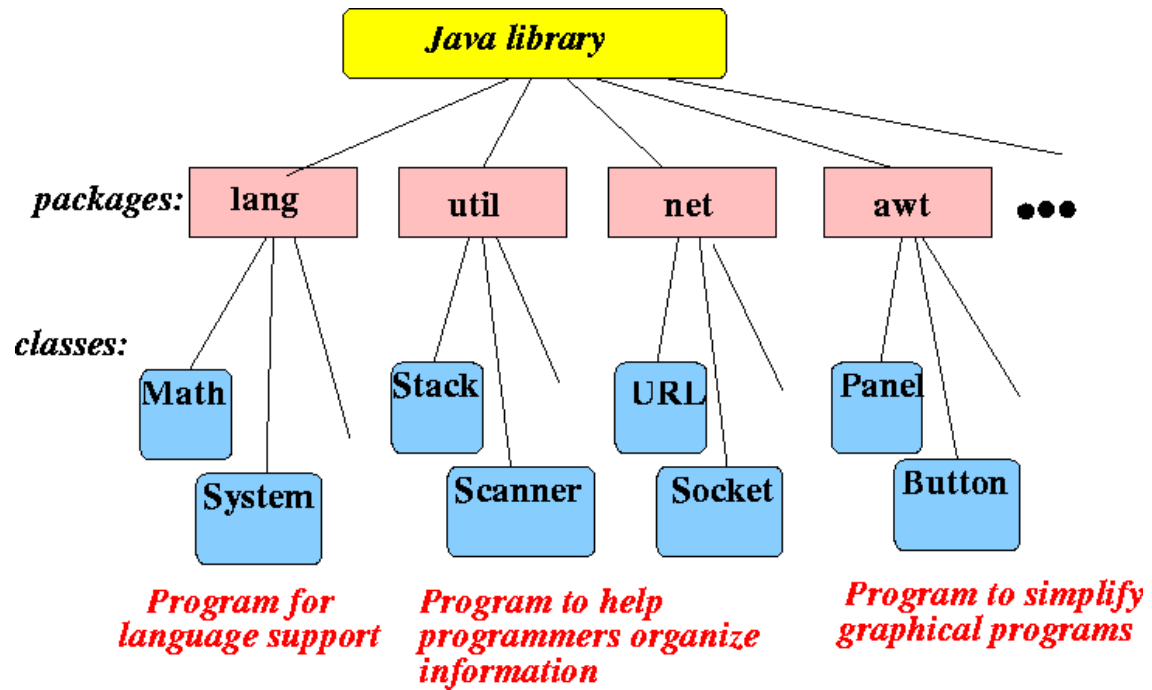
- Can have one to several characters
- Variable names should begin with a letter(A-Z, a-z), an underscore(_), a dollar sign(\$)
- A keyword cannot be used as an identifier.
- **Cannot** start with a digit but digits can be used after first character.
- Most importantly identifiers are case sensitive

Valid Example: age, \$salary, _value, __1_value, my_var, sample123, maxLoad

Invalid Example: 123abc, -salary, super, this

The Java Class Libraries:

The Java Class Library (JCL) is a set of dynamically loadable libraries that Java applications 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.



Chapter-2

Introducing Data Types and Operators

Java's Primitive Types:

Java contains two general categories of built-in data types as shown here:

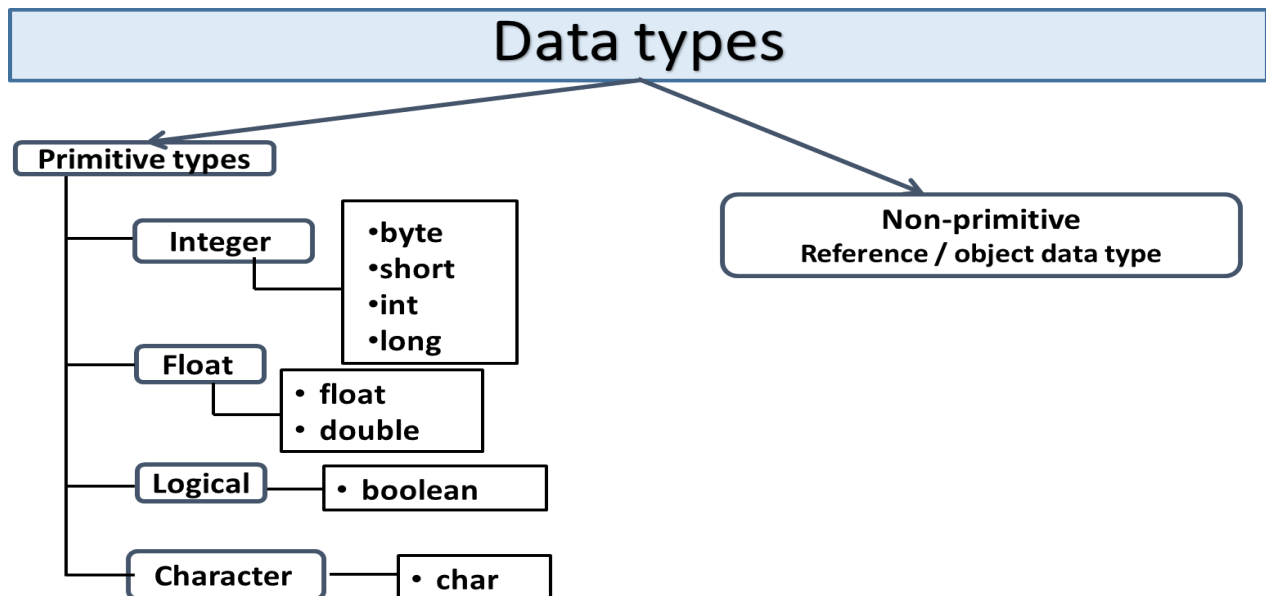


Table: Java's built-in data types

Type	Contains	Default	Size	Range
boolean	True or false	False	1 bit	NA
byte	Signed integer	0	8bits	-128 to 127
char	Unicode Character	'\u0000'	16 bits	\u0000 to \uFFFF
double	IEEE 754 floating point	0.0d	64 bits	$\pm 4.9E-324$ to $\pm 1.7976931348623157E+308$
float	IEEE 754 floating point	0.0f	32 bits	$\pm 1.4E-45$ to $\pm 3.4028235E+38$
int	Signed Integer	0	32 bits	-2147483648 to 2147483647
long	Signed Integer	0L	64 bits	-9223372036854775808 to 9223372036854775807
short	Signed Integer	0	16 bits	-32768 to 32767

Note: Java defines four integer types: **byte**, **short**, **int**, and **long**

Literals:

Literals in Java are a sequence of characters (digits, letters, and other characters) that represent constant values to be stored in variables. Types of literals in java with example as shown here.

Important points:

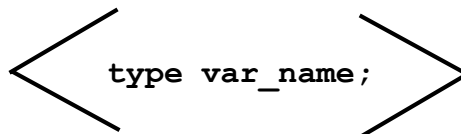
- also commonly called **constants**.
- Java literals can be any of the primitive data types.
- **Character constants** are enclosed in **single quotes**.

Table: Types of literals

Type	Sub Type	Representation	Example
Integer	Decimal	Default	123
	Octal	Prefix: '0' (Zero)	0123
	Hexadecimal	Prefix: '0x' or '0X'	0x123
	Binary	Prefix: '0b' or '0B'	0b1100
	long	Prefix: 'l' or 'L'	123L
Floating	float	Postfix: 'f' or 'F'	40.33f
	double	Default - Postfix: 'd' or 'D'	
Character		Single quote: ' '	'a', ':', '\n', '\b', 'r', '\f', '\t', '\ddd', '\uxxx'
String		Double quote: " "	"HELLO"
Boolean			True, false

A Closer Look at Variables:

Variables are declared using the form.



`type var_name;`

Where, **type** – data type of the variable
var_name – name of the variable

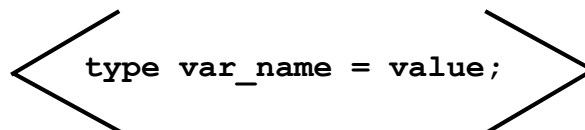
Note:

- ❖ an **int** variable cannot turn into a **char** variable
- ❖ All the variables in java must be declared prior to their use
- ❖ The type of the variable cannot be change during its lifetime

Initializing a Variable:

A variable must be declared before to their use. This is necessary because the compiler must know what type of data or value that can contains it use.

At the time of declaration, we can initialize the value to the variable by using assignment operator as shown here:



`type var_name = value;`

Where, **value** - is the value that is to var when var is created. Value is known at compile time

Example:

```
int count = 10;
char ch = 'x';
float f = 1.2F;
int a,b = 8;
```

Dynamic Initialization:

Java allows variables to be initialized dynamically, using any expression valid at the time the variable is declared. Value is not known at compile time but known at runtime.

Example: `int port = getSocket();`
 `double volume = 3.14 * radius * radius * height;`
 `int It = getTax(basic, Da, PF);`

Here, volume, port, basic, Da, PF, and It is dynamically initialized at runtime.

The Scope and Lifetime of Variables:

Java allows variables to be declared within any block. A block is begin with an opening brace and ended by a closed brace.

Scope of variable is the part of the program where the variable is accessible and determined at the compile time.

Example:

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

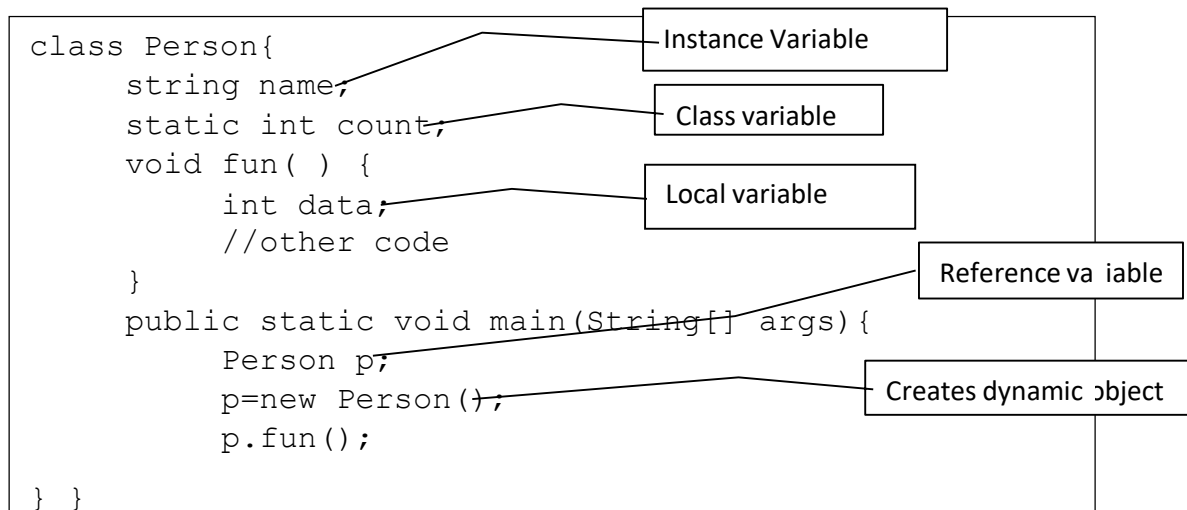
        if (true) {    // creates new scope
            int y = 20;

            System.out.println ( y );    // OK
        }
        System.out.println ( y ); //ERROR, not in scope

        System.out.println ( x );    // OK
    }
}
```

Types of variables:

- Local variables – inside method
- Instance variables - common to class methods.
- Class variables – are static variables
- Reference variable – hold some reference



Operators:

An operators is a symbol that tells the compiler to perform a specific mathematical, logical or other manipulations.

Table: List of operators and example

Category	Operators	Meaning	Example
Arithmetic	+	Addition	<pre> class ArtDemo{ public static void main(String[] args){ int iresult, irem; double dresult, drem; iresult=10/3; irem=10%3; dresult=10.0/3.0; drem=10.0%3.0; System.out.println("Result&remainder:" +iresult + "\t" +irem); System.out.println("Result&remainder:" +dresult + "\t" +drem); } } </pre>
	-	Subtraction	
	*	Multiplication	
	/	Division	
	%	Modulus	
	++	Increment	
	--	decrement	
Relational	==	Equal to	<pre> class RelDemo{ public static void main(String[] args){ int i=10, j=11; if(i==j) System.out.println("Equal"); else System.out.println("NotEqual"); } } </pre>
	!=	Not equal to	
	>	Greater than	
	<	Less than	
	>=	Greater than or equal to	
	<=	Less than or equal to	
Logical	&	AND	<pre> class RelDemo{ public static void main(String[] args){ int i=10, j=11; if(i==j) System.out.println("Equal"); else System.out.println("NotEqual"); } } </pre>
		OR	
	^	XOR(exclusive OR)	
		Short-circuit OR	
	&&	Short-circuit AND	
	!	NOT	

Short Circuit Operator:

Provides Short-Circuit for AND & OR Logical Operators

- In **AND (&&)** – if the first operand is *false*, the outcome is false no matter what value the second operand has.
- In **OR (||)** – if the first operand is *true*, the outcome is true no matter what value the second operand has.

Example:

```
class ShrtCirDemo{
    public static void main(String[] args){
        int n=10, d=2;
        if(d!=0 && (n%d) == 0)
            System.out.println("Modulus is performed ");

        d=0;
        if(d!=0 && (n%d) == 0)
            System.out.println("Modulus not performed ");
        if(d!=0 & (n%d) == 0)
            System.out.println("Modulus is performed ");
    }
}
```

Both expressions are evaluated

Short circuit operator prevents a divisions by zero

Both expressions are evaluated

- A short circuit operator is one that doesn't necessarily evaluate all of its operands.
- *expr1* && *expr2* represents a logical **AND** operation that employs short-circuiting behaviour. That is, *expr2* is not evaluated if *expr1* is logical 0 (*false*).

Note: The difference between the normal (&, ||) and short-circuit (&&, ||) versions

- ✓ The normal operands will always evaluated each operand
- ✓ The short-circuit version will evaluate the second operand only when necessary.

The Assignment Operator:

It is single equal sign (=) that can used to assign a value to the variable.

Syntax:

var = expression / value;

Example:

```
int x, y, z;
x = y = z = 100;
```

Shorthand Assignments:

Shorthand assignment operators that simplify the coding of certain assignment statements.

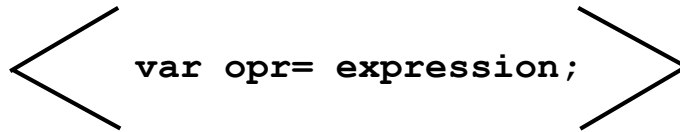
Example:

`x = x + y;` Equivalent to `x += y;`

The operator pair += tells the compiler to assign x the value of x plus 10.

The shorthand will work for all the binary operators in Java.

Syntax:



var opr= expression;

Thus, the arithmetic and logical shorthand operators are the following:

+=	-=	*=	/=
%=	&=	 =	^=

Bitwise Operator:

- Bitwise operators are used to test, set, or shift the individual bits that make up a value.
- Bitwise operator can be applied to values of type long, int, short, char, or byte.
- Bitwise operations cannot be used on Boolean, float or double or class type.

Table: List of Operators and example (continued)

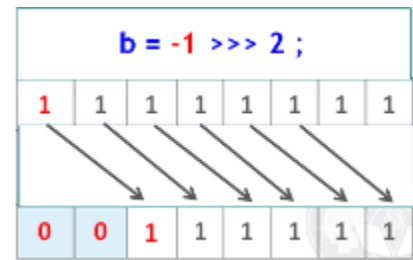
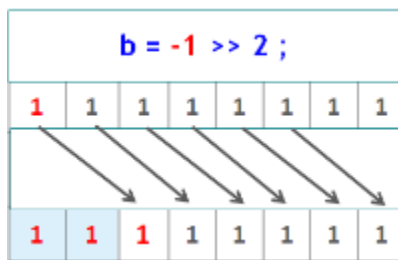
Category	Operators & Meaning	Description	Example
Bitwise	& Bitwise AND	Bit by bit evaluated	<pre>class UpLwDemo{ public static void main(String[]args) { char ch; for(int i=0; i<10; i++) { ch=(char) ('A' + i); System.out.print(ch); ch=(char) ((int) ch 32); System.out.print(ch + " "); } } //Uppercase to Lowercase Output: Aa Bb Cc Dd Ee Ff Gg Hh Ii Ji</pre>
	 Bitwise OR	Bit by bit evaluated	
	^ Exclusive OR	Bit by bit evaluated	
	>> Signed Shift right	sign bit is shifted in high-order positions.	
	>>> Unsigned Shift right	Zero's are shifted in high-order positions.	
	<< Left shift	Zero's are shifted in Low-order positions.	
	~ One's complement	One's complement	
Miscellaneous	?:	Ternary or conditional	<code>b = (a == 1) ? 20 : 30;</code>
	instanceof	The operator checks whether the object is of a particular type	<pre>String name = "James"; boolean result=name instanceof String; System.out.println(result);</pre>

Shift operator:

Java defines the 3 shift-operators shown here:

<< - Left Shift	Bits are moved left by the number of bits specified by the right operand and 0 bits are filled on the right.
>> - Right Shift	Bits are moved right by the number of bits specified by the right operand and preserves the signed bit that means shifted values filled up with 1 bits on the right.
>>> Unsigned right shift	Bits are moved right by the number of bits specified by the right operand and shifted values are filled up with zeros. It is also called as <i>Zero –fill right shift</i> .

Examples:



Type conversion in Assignments:

It states that converting from one type to another type. Booleans cannot be converted to other types. There are two types:

- Implicit Type
- Explicit Type

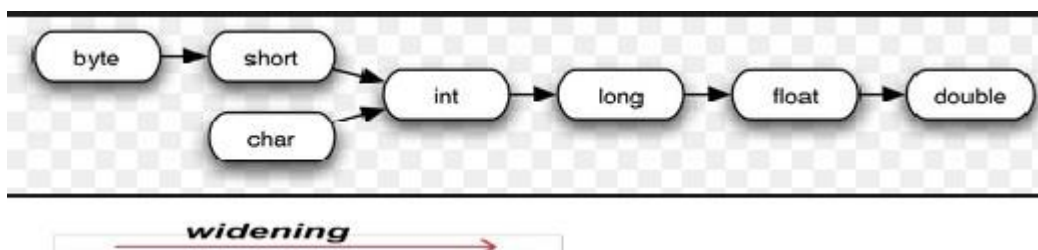
Implicit Type:

An implicit conversion means that a value of one type is changed to a value of another type automatically.

Automatic type casting take place when,

- The two types are compatible
- The target type is larger than the source type.

The lower size is widened to higher size. This is also named as **automatic type conversion** or **widening**.



Example:

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


Explicit Type:

Explicit conversions are done via **casting**. When you are assigning a larger type value to a variable of smaller type, then you need to perform explicit type casting. It is called as **explicit conversion** or **narrowing conversion** and there may be data loss in this process because the conversion is forceful.

Syntax:

`(target-type) expression;`

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

double → float → long → int → short → byte

Narrowing

Example: `double x = 10.5;`
`int y = (int)x; //explicit casting`

The double x is explicitly converted to int y.

Operator Precedence:

An operator's precedence determines at what point it is evaluated in an expression. An operator with a higher precedence will be evaluated before an operator with a lower precedence.

Table: Shows the order of precedence for all Java operators, from highest to lowest.

Highest	Type	Categories	Precedence						
	Unary	Postfix	expr++	expr- -					
		Prefix	++expr	--expr	~	!	+	-	(type-cast)
	Arithmetic		*	/	%				
			+	-					
	Shift		>>	>>>	<				
					<				
	Relational	Comparison	>	>=	<	<=	instanceof		
		Equality	==	!=					
	Bitwise	AND	&						
		Exclusive OR	^						
		Inclusive OR							
	Logical	AND	&&						
		OR							
	Ternary		?:						
	Assignment		=	Op=					
Lowest									

Expressions

Operators, variables, and literals are components of expressions.

Type Conversion in expressions:

- Possible to mix 2 or more compatible different data types.
- they are all converted to the same type.
- *Java's type promotion* rules can be used for the conversion in an expression.

Example: `2 + (5.0*2.0) -3;`
 `char ch= (char) (ch1 + ch2);`

```
class ExprDemo{
    public static void main(String[] args)
    {
        byte b;
        int i;

        b=10;
        i = b * b; //No cast needed

        b=10;
        b=(byte) (b * b); // cast is needed
        System.out.println("i= " + i + "b= " +b);
    }
}
```

Note: Java's type Promotion Rules

- All char, byte and short values are promoted to int.
- Any one operand is a long, the whole expression is promoted to long
- Any one operand is a float, the whole expression is promoted to float
- Any one operand is a double, the result is double.

Chapter-3

Program Control Statements

Handling User Input / Output:

- Java I/O (Input and Output) is used *to process the input and produce the output*.
- uses the concept of stream to make I/O operation fast.
- The java.io package contains all the classes required for input and output operations

Stream

A stream is a sequence of data. In Java a stream is composed of bytes.

Two types of Streams:

- Byte Streams: handling the input and output of bytes.
- Character Streams: handling the input and output of characters.

Predefined Streams:

There are 3 streams variables:

1. **System.in**: standard output stream
 2. **System.out**: standard input stream
 3. **System.err**: standard error stream
- } automatically import the **java.lang** package

By default, all these streams are the *console*.

Input characters from the Keyword:

- To read a character from the keyboard we will use **System.in.read()**.
- System.in is the complement to **System.out**
- It is the input object attached to the keyboard.
- The **read()** method waits until the user presses a key and then returns the result.
- The character is returned as an **integer**, so it must be **cast** into a **char** to assign it to a **char** variable.

Example:

```
class RdChar{
    public static void main(String[] args)throws IOException{
        char ch;
        System.out.println("enter a key");
        ch=(char)System.in.read(); //Read a character
        System.out.println("your key is= " +ch);
    }
}
```

Output:

```
E:\JAVAEamples>java RdChar
enter a key
a
your key is= a
```

Reading string:

String can be read using following classes

1. *InputStreamReader*
 2. *BufferedReader*
 3. *Scanner* → *java.util package*
- } automatically import the **java.io** package

1. *InputStreamReader* class:

- Can be used to read data from keyboard.
- Converts Byte stream to character stream.

Syntax:

```
InputStreamReader ir=new InputStreamReader(System.in);
```

2. *BufferedReader* class:

can be used to read data line by line

Syntax:

```
BufferedReader input = new BufferedReader(ir);
```

Methods used for reading string in *BufferedReader*

Sl No.	Method and description
1	<i>int read ()</i> :- This method reads a single character.
2	<i>int read (char[] bufArr, int off, int len)</i> - This method reads characters into a portion of an array.
3	<i>String readLine ()</i> :- This method reads a line of text.

Example:

```
class RdBr {
    public static void main(String[] args)throws java.io.IOException{
        String str;
        InputStreamReader ir=new InputStreamReader (System.in);
        BufferedReader input = new BufferedReader(ir);
        // declare in a single line.
        //BufferedReader input = new BufferedReader(new
                                InputStreamReader(System.in));
        System.out.println ("Enter a Name");
        str = input.readLine();
        System.out.println ("String entered is = " + str );
    }
}
```

Output:

```
E:\JAVASamples>java RdBr
Enter a Name
Paul
String entered is = Paul
```

3. Scanner class:

- The **Java Scanner class breaks the input into tokens using a delimiter that is whitespace by default.**
- It provides many methods to read and parse various primitive values.

Syntax:

```
Scanner sc_name= new Scanner(System.in);
sc_name.method();
```

Methods in Scanner class

Method	Returns	Exception on failure
int nextInt()	Returns the next token as an int.	InputMismatchException
long nextLong()	Returns the next token as a long.	
float nextFloat()	Returns the next token as a float.	
double nextDouble()	Returns the next token as a long.	
String next()	Finds and returns the next complete token as a string; a token is usually ended by whitespace such as a blank or line break.	NoSuchElementException
String nextLine()	Returns the rest of the current line, excluding any line separator at the end.	
void close()	Closes the scanner.	

Example:

```
class ScannerTest {
    public static void main(String args[]){

        Scanner input = new Scanner(System.in);
        System.out.println("Enter your rollno");
        int rollno = input.nextInt();

        System.out.println("Enter your name");
        String name = input.next();

        System.out.println("Enter your fee");
        double fee = input.nextDouble();
        System.out.println ("Rollno:" + rollno + " name:" + name
                             + " fee:" + fee);

        input.close();
    }
}
```

Output:

```
E:\JAVAEexamples>java ScannerTest
Enter your rollno
10
Enter your name
John
Enter your fee
30000.00
Rollno:10 name:John fee:30000.0
```

Control Statements:

Program control statements can be organized into the following categories:

1. Selection or Conditional or Decision Statements
2. Iteration or Looping Statements
3. Jump Statements

Selection or Conditional or Decision Statements:

Decision making structures have one or more conditions to be evaluated or tested, if it is true the statements get executed and optionally, other statements to be executed if the condition is determined to be false.

Table: Types of Conditional statements

Type	Syntax and Explanations	Example
1. Simple if	<pre>if(condition) { //Statements }</pre> <p>The Java if statement tests the condition. It executes <i>if block</i> if condition is true.</p>	<pre>int age=20; if(age>18){ System.out.print("Age is greater than 18"); }</pre>
2. if-else	<pre>if(condition) { //Statements }else{ //Statements }</pre> <p>It executes the <i>if block</i> if condition is true otherwise <i>else block</i> is executed.</p>	<pre>int number=13; if(number%2==0) { System.out.println("even number"); }else{ System.out.println("odd number"); }</pre>
3. if-else-if ladder	<pre>if(condition1){ Statements; }else if(condition2){ Statements; } ... else{ Statements; }</pre> <p>The if-else-if ladder statement executes one condition from multiple statements.</p>	<pre>int age=25,basic=4000,tax = 200; if (age>18) { System.out.println("greater"); } else if (basic > 5000) sal = basic - tax; else sal = basic;</pre>

Type	Syntax and Explanations	Example
4. nested if	<pre>if (condition){ if (condition){ Statements; } }</pre> <p>You can use one if or else if statement inside another if or else if statement(s).</p>	<pre>int a=15, b=20, c=18, big; if (a>b){ if (a>c) { big = a; } }</pre>
5. switch or multiway	<pre>switch(expression) { case value1:Statements; break; case value2:Statements; break; default: //code to be executed if all cases are not matched }</pre> <p>The Java <i>switch statement</i> executes one statement from multiple conditions.</p>	<pre>int number=20; switch(number) { case 10: System.out.println("10"); break; case 20: System.out.println("20"); break; case 30: System.out.println("30"); break; default: System.out.println("Not in 10, 20 or 30"); }</pre>

Iteration Statements:

A loop statement allows us to execute a statement or group of statements multiple times.

Table: Types of Iteration Statements

Type	Syntax and Explanations	Example
1. while	<pre>while(condition) { Statements; }</pre> <p>Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.</p>	<pre>int i=1; while(i<=10) { System.out.println(i); i++; }</pre> <p>Output: 1 2 3 4 5 6 7 8 9 10</p>
2. do-while	<pre>do{ Statements; }while(condition);</pre> <p>Like a while statement, except that it tests the condition at the end of the loop body.</p>	<pre>int i=1; do{ System.out.println(i); i++; }while(i<=10);</pre> <p>Output: 1 2 3 4 5 6 7 8 9 10</p>

Type	Syntax and Explanations	Example
3. for loop	<pre>for(initialization;condition;incr/decr) { Statements; }</pre> <p>Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.</p>	<pre>for(int i=1;i<=10;i++){ System.out.println(i); }</pre> <p>Output: 1 2 3 4 5 6 7 8 9 10</p>
4. Enhanced for loop	<pre>for(type var : array){ Statements; }</pre> <ul style="list-style-type: none"> • for-each loop is used to traverse array or collection in java. • It is easier to use than simple for loop because we don't need to increment value and use subscript notation. • It works on elements basis not index. It returns element one by one in the defined variable. 	<pre>int arr[]={12,23,44,56,78}; for(int i:arr){ System.out.println(i); }</pre> <p>Output: 12 23 44 56 78</p>

Jump Statements:

Loop control statements change execution from its normal sequence.

Type	Syntax and Explanations	Example
break	<pre>jump-statement; break;</pre> <ul style="list-style-type: none"> • 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. 	<pre>for(int i=1;i<=10;i++){ if(i==5){ break; } System.out.println(i); }</pre> <p>Output: 1 2 3 4</p>
continue	<pre>jump-statement; continue;</pre> <ul style="list-style-type: none"> • Used to continue loop. • It continues the current flow of the program and skips the remaining code at specified condition. • In case of inner loop, it continues only inner loop 	<pre>for(int i=1;i<=10;i++){ if(i==5){ continue; } System.out.println(i); }</pre> <p>Output: 1 2 3 4 6 7 8 9 10</p>

Type	Syntax and Explanations	Example
break label or break as a form of goto	<p>jump-statement; break LABEL;</p> <ul style="list-style-type: none"> •The break statement can be followed by a label. •The presence of a label will transfer control to the start of the code identified by the label. 	<pre>for(i=1 ; i<=3 ; i++){ one: { two: { three: { System.out.println("i="+ i); if(i==1) break one; if(i==2) break two; if(i==3) break three; }System.out.println("End Three"); } System.out.println("End Two"); } System.out.println("End One"); } // end for</pre> <p>Output:</p> <pre>End One End Two End One End Three End Two End One</pre>
continue label or continue as a form of goto	<p>jump-statement; continue LABEL;</p> <ul style="list-style-type: none"> •skips the current iteration of an outer loop marked with the given label. 	<pre>int a[][]={{1,1,1},{2,2,2},{3,3,3}}; Outer: for(int i = 0; i < 3; i++) { for(int j = 0 ; j < 3 ; j++) { if(i == j) continue Outer; System.out.println("Element is:" + a[i][j]); } }</pre> <p>Output:</p> <pre>Element is: 2 Element is: 3 Element is: 3</pre>
return	<p>return value;</p> <ul style="list-style-type: none"> •The return statement exits from the current method, and control flow returns to where the method was invoked. •The return statement has two forms: one that returns a value, and one that doesn't. 	<pre>boolean isOdd(int num) { if (num/2 == 0) return false; return true; }</pre>

Nested Loops:

- one loop can be nested inside of another loop.
- used to solve a wide variety of programming problems and are an essential part of programming.

Example:

```
for(int i=1;i<=5;i++){  
    for(int j=1;j<=i;j++){  
        System.out.print(j+" ");  
    }  
}
```