Introduction

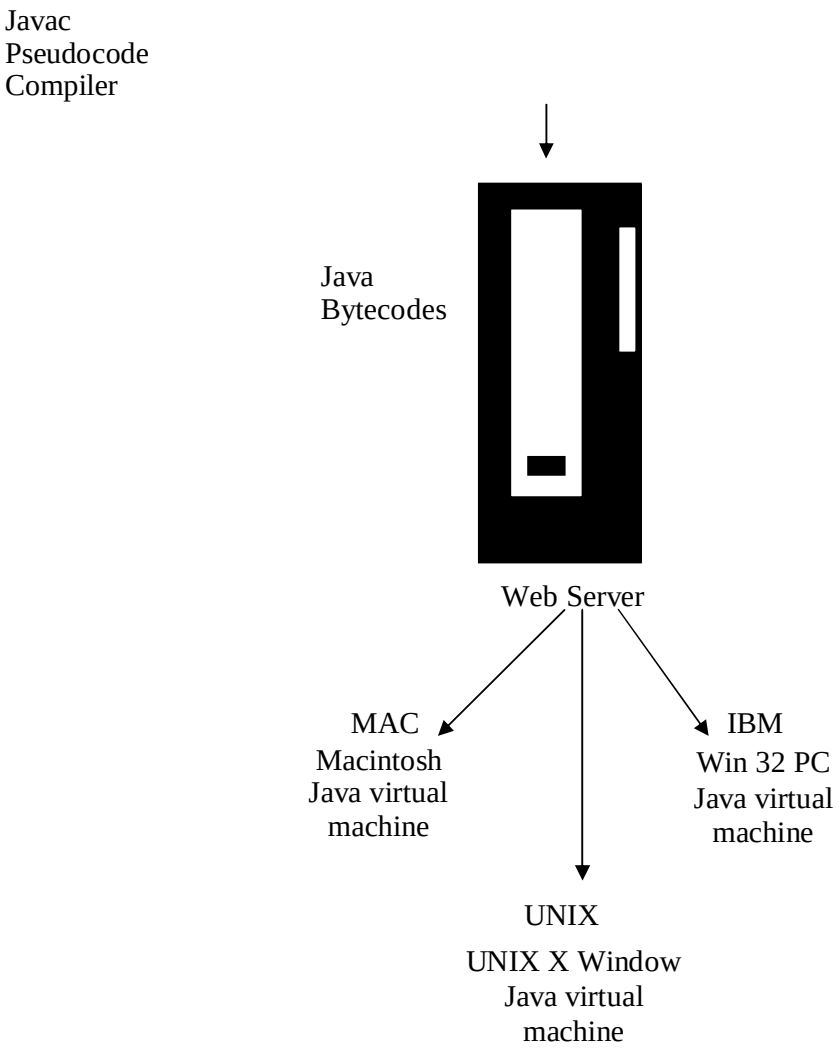
WHO NEEDS JAVA, ANYWAY?

"Marvelous!" you think as you look down at the software package in your hands. Just what you've been looking for—the program that will solve all your problems, dry all your tears, fulfill all your dreams. Yes, life is truly good. And then you notice the fine print: "Runs on Macintosh only"; but your computer runs Windows. If your world does not exactly turn black and your blood does not quite run cold, at the very least you end up frustrated and annoyed. "Why can't every computer run the same programs?" you ask. Java may well be the solution to this problem.

Java is a computer language designed for programming on the Internet. At the moment, Java has the attention of the media and the public because of its ability to add action to otherwise lifeless Web pages. Most people accessing the Web do so using a modem and a dial-up connection. While full-motion images of the sort we've all grown to expect take too long to deliver over a modem, Java allows motion by sending still images over the Internet and then animating them locally, on the same computer that hosts the browser. This effectively overcomes the bandwidth bottleneck. However, the ultimate potential significance of Java extends far beyond such entertaining tricks.

Java's power comes from a new sort of portability known as *cross-platform binary portability*. Programs written in C or Pascal on one type of computer can often be ported to run on a different type of computer. The cost of doing this can sometimes be much less than the cost of rewriting the program from scratch. Nevertheless, such porting is often time-consuming and difficult, and frequently the results are less than ideal. For batch-oriented programs such as processing payroll or utility bills, such porting is often employed, but in these days of graphical user interfaces, moving a program from Windows to the Macintosh or UNIX is often a Herculean task. And, even if the program is intentionally written to be portable, what is actually portable is the source code—the high-level language instructions—not the actual executable code. What if you could put that Windows *95* program disk in your X Window UNIX machine (and vice versa) and just have it work?

This is the portability provided by Java. A Java executable written for one computer can be run, without modification, on another computer supporting Java. The other computer does not need the corresponding source code to accomplish this feat; porting it automatic and virtually instantaneous. This means that users owning entirely different types of computers can download a Java executable from a server and run that executable on their systems and expect an identical result. Figure 1 shows this process. The capability of downloading a program and executing it on a variety of computers is expected to lead to entirely new kinds of application programs. This is the true magic of Java.



THE HISTORY OF JAVA

Java began life in 1991 as a programming language for consumer electronics devices at Sun Microsystems, Inc. At the time, Sun was seeking to diversify its business beyond the very popular UNIX-based workstations for which the company is still noted. Sun engineers seeking to build an intelligent remote control for multihundred-channel cable TV needed a simple programming language capable of hosting highly reliable software, because nothing is guaranteed to generate complaints like customers having their cable TV go down in the midst of a movie.

The Sun folks looked at several alternatives. Building a compiler for an existing language such as C++ seemed to require too much effort, because the hardware components of the system changed so frequently. Use of assembly language was rejected because of the difficulty of producing reliable software. In the end, the engineers decided to create a new language called *Oak*, built specifically for their task. Oak was based on C++ but lacked certain C++ features known to be a common source of programming errors. It also had additional features not found in C++.

Unfortunately for Sun, but fortunately for the rest of us, the anticipated market for intelligent remote controls failed to appear. This looked like the end of Oak. But the engineers working on the project took a step back and asked themselves, "What can we possibly use this thing for?" Some of them looked at the Internet and saw a unique fit. In a short period of intense coding they crafted a Web browser called *HotJava*, using Oak, which they rechristened *Java*. HotJava introduced the idea of an applet, a small program that could be downloaded from a Web server and executed within an environment provided by a browser. This early demonstration of the capability for animation of Web pages captured the attention of Sun executives, who decided to sponsor the Java project.

A preliminary (alpha) version of Java and HotJava was made widely available over the Internet as part of a free Java Developer's Kit (JDK). Sun reasoned that the only way to achieve wide-scale acceptance of Java as a universal standard was to make it available to everyone. The experiences of users and developers who experimented with Java were helpful to Sun engineers, who released a second (beta) version of the JDK that incorporated suggested improvements and corrections. In 1996, Netscape Communications Corp. announced that support for Java applets would be included in version 2.0 of its Navigator browser. Later that year Microsoft followed suit by announcing Java support for its Internet Explorer version 3.0. The Java revolution was on!

THE FEATURES OF JAVA

The features of Java were described in an early paper written by Sun engineers. "The Java Language: A White Paper." This paper defined Java as "a simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language." Although this torrent of computer-speak jargon has often been labeled the "buzzword description" and was doubtless intended with tongue in cheek, it nevertheless accurately identifies many of the features of Java that make it so well-suited for programming Internet applications. Let's briefly examine each of these words, with a view to better understanding what makes Java tick.

Simple

Today, one of the most popular computer languages is C++ (pronounced "Sea-plus-plus"). This language was developed by AT&T's Bell Labs to bring object-oriented features to its popular C language. Despite its success, C++ has been widely criticized for being too complex.

Java, though adopting much of the look and feel of C++, is a much simpler language than C++. It has fewer and less-complex constructs, so it is easier to learn.

Object-Oriented

Like C++, Java is an object-oriented language. Object-oriented languages allow the programmer to organize a program so that it closely models the real world in structure and in the interactions among its components. This is particularly valuable in implementing applications using the graphical user interfaces popularized in the PC world by the various versions of Microsoft Windows. Writing an application using a graphical user interface tends to be much easier by use of an object-oriented language than otherwise.

Distributed

Java was built with the Internet and Web in mind. As do most other languages, Java includes prebuilt components or *libraries* that provide important additional capabilities beyond the language itself. However, Java's standard libraries specifically include network-aware units that greatly facilitate writing Internet applications. Additionally, the building blocks of a particular Java application do not have to reside locally, on your desktop machine, as they do with traditional programming languages.

Interpreted

Remember the second generation of programming languages? One of the first advances, after the invention of the program itself, was the invention of the program interpreter. Like the early Short-Code and Speedcode systems, Java is an interpreted language.

This means that Java's executable files are composed of so-called *bytecodes* that are instructions and data relating to a hypothetical computer called the *Java virtual machine.* Each machine that runs a Java program uses a small program, known as the *Java run-time system,* to execute the Java bytecodes in your program. This design is what makes it possible to run the same program on a Macintosh, a Sun, and a PC.

Robust

A robust program is one that does not fail (at least in a catastrophic way). If you've used a computer for any length of time, you're obviously aware that much modern software lacks this property. One cause of this is the complexity of modern software systems.

Java contains features that make the task of writing robust software easier. The designers did this by first omitting features (found in other languages) that are known to cause errors. They then added features (such as strong typing) that allow the developer to discover errors early, rather than after the product is in the customer's hands.

Other features in this same vein include automatic memory allocation, garbage collection, and exception handling. In Java programs, exceptions can be detected and handled according to instructions written by the programmer, often allowing software to keep working in the face of unexpected problems.

Secure

One of the potential terrors of the Internet is the possibility of security breaches—viruses that infect your computer, or hackers who take advantage of a software glitch to invade your personal cyberspace and make off with confidential information. Java has a multitude of ways for dealing with evildoers who would try to compromise your system using a Java program.

Applets, which are Java programs automatically downloaded when a Web page is displayed, are subject to a number of limitations that are designed to reduce the chance that simply viewing someone's page might result in harm to your system or data. No such system is absolutely reliable and none will ever be; but Java represents the state-of-the-art in reducing the chances of a disaster.

Architecture Neutral

This means, of course, that your Java program will work identically whether you run it in a Quonset hut or in a Georgetown townhouse. No, wait! That's not what it means at all.

The word *architecture* in this phrase does not refer to the building in which you live, but to the home in which your computer program lives—in other words, the computer system. Java's bytecodes are designed to be read and interpreted—in exactly the same manner—on any computer hardware or operating system that supports a Java run-time. No translation or conversion is necessary.

Portable

An early form of portability involved carrying media, for example, floppy disks, from one system to another. Portability became a much larger problem once different sorts of computers were interconnected to form the Internet.

Java programs contain no implementation-dependent aspects, so the result of executing a series of Java bytecodes should always be the same no matter on what system they are executed. Moreover, the Java run-time system itself, though it is written in C, is written in a way that simplifies porting the Java run-time to a new computer system.

High Performance

A typical problem with interpreted languages is that they are somewhat less efficient than compiled languages. A program written by use of an interpreted language may run 20 to 100 times slower than the same program written by use of a compiled language.

Java aims at overcoming this problem through the use of a technique known as *just-in-time compilation*. A just-in-time compiler is an interpreter that remembers the machine code sequences it executes corresponding to the input bytecodes. Having *figured* out the proper machine code sequence once, it doesn't have to figure it out again if the same code is reexecuted. Instead, it retrieves the memorized sequences and executes them straight away. Studies have suggested that just-in-time compilation may make interpretation of Java bytecodes almost as efficient as native execution of machine-language code.

Multithreaded

Most of us can walk and chew gum at the same time. All of us do many other things simultaneously. Computers are no different. If they are able to perform activities in parallel, the performance of the entire system can be improved. This technology is known as *multithreading*. A multithreaded system can, for example, format a floppy disk while a user surfs the Web using a browser. Multithreaded applications allow you to complete more tasks in a given time and to use a system's resources more efficiently.

However, developing multithreaded applications in C or C++ can be agony, because these languages lack standard support for operations necessary to create and control threads. Java includes support for multithreaded applications as part of its basic library.

Dynamic

Java's program units, *classes*, are loaded dynamically (when needed) by the Java runtime system. Loaded classes are then dynamically linked with existing classes to form an integrated unit. The lengthy link-and-load step required by third-generation programming languages is eliminated.

Thus, when parts of an application you use are updated, you don't have to buy the latest copy. The dynamic nature of Java allows you as a developer to always have the most up-to-date version of your software available to your users.

Java Program Types

- Java applets – programs embedded in a Web page
- Java applications – stand-alone programs

Your First Java Program

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

class name

Class Object Method

Class Name Rules

- A class name ***Must*** begin with a letter of alphabet (which includes any non-English letter such as α , β , π) , an underscore , or a dollar sign.
- A class name ***can*** contain only letters, digits, underscores, or dollar signs.
- A Class name can not be a Java Programming language reserved keywords such as ***public*** or ***class***.
- A class name can not be one of the following values : true, false , or null.

Reserve keywords in Java

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

Keywords for data types are:

```
boolean byte  char  int  long  short  float  double
```

Keywords for access control are:

```
private  protected  public
```

Keywords for modifiers are:

```
abstract  final  native  private  protected  public
static   transient  synchronized  volatile  strictfp
```

Keywords for catch-exception are:

```
try  catch  finally  throw
```


Keywords for loops or decision-makers are:

```
break  case  continue  default  do  while  for
switch  if  else
```

Keywords for class functions are:

```
class  extends  implements  import  instanceof
new  package  return  interface
this  throws  void    super
```

Keywords for assigned values are:

```
true  false  null
```

Outdated keywords are:

```
const  goto
```

Some Valid Class Names in the Java Programming Language

Class Name	Description
Employee	Begin with an uppercase letter.
UnderGraduateStudent	Begins with uppercase letter, contains no spaces, each new word with an initial uppercase letter.
InventoryItem	Begins with uppercase letter, contains no spaces, second word with an initial uppercase letter.
Budget2004	Begin with an uppercase letter, no spaces.

Some unconventional Class Names in the Java Programming Language

Class Name	Description
employee	Begin with an lowercase letter.
undergraduatestudent	New words are not indicated with initial uppercase letters.
Inventory_Item	The underscore is not commonly used to indicate a new word.
BUDGET2004	All are in uppercase letter.

Some illegal Class Names in the Java Programming Language

Class Name	Description
an employee	Space between an and employee
Class	Class is Reserved Word
2004BUDGET	Starts with a number
Phone#	The # symbol is not allowed

Program Comments

Program Comments- Nonexecuting statements that you add to a program for the purpose of documentation

- Line comments- //
- Block comments - /* */
- javadoc comments - /** */

Line Comment Example :

```
int i = 0;   // initialize the loop variable
```

Block Comment Example

Error!

```
/*
 * Step 4: Print static methods, both public and
protected,
 *         but don't list deprecated ones.
 */
```

javadoc Example

```
/**
 * Display a list of classes, many to a line.
 *
 * @param classes The classes to display
 * @return <tt>true</tt> on success,
 * <tt>false</tt> on failure.
 * @author David Flanagan
 */
```


Primitive Types of Data

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

The eight primitive data types are called **primitive types** because they are simple and uncomplicated

Integers

- Variables of type int store **integers** or whole numbers
- Integer can be any whole number from
  –2,147,483,648 to 2,147,483,647

To declare variables of the ***same type*** with an initial value :-

```
int numberOne = 15;
int numberTwo = 25;
```

Another way

```
int numberOne = 15 , numberTwo = 25;
```

To define variables of the ***different types without initial values :-***

```
int numberOne , numberTwo ;
double hisSalary, herSalary ;
```


limits on integer values by type :-

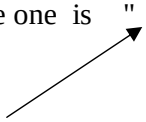
Type	Min Value	Max Value	Size in Byte
Byte	-128	128	1
Short	-32,768	32,768	2
Int	-2,147,438,648	2,147,438,648	4
long	-9,223,372,036,854,775,808	9,223,372,036,854,775,808	8

```
short   oneShort   =  25;
long    oneLong    =  12345678765431L;

System.out.print("The Short one  is  ");
System.out.println(oneShort);
System.out.print("The Lone one  is  ");
System.out.println(oneLong);
```

Or

```
System.out.println("The Short one  is    "  + oneShort);
System.out.println("The Lone one  is    "  + oneLong);
```



Concatenates the sentence with the number stored in the variable

More examples :-

```
1234        // An int value
1234L       // A long value
0xffL       // Another long value
```

Integer arithmetic in Java is modular, which means that it never produces an overflow or an underflow when you exceed the range of a given integer type. Instead, numbers just wrap around. For example:

```
byte b1 = 127, b2 = 1;   // Largest byte is 127
byte sum = b1 + b2;      // Sum wraps to -128, which is
the smallest byte
```


Integer Arithmetic Operators

Operator	Description	Example
+	Addition	45 + 2 >>> 47
-	Subtraction	45 - 2 >>> 43
*	Multiplication	45 * 2 >>> 90
/	Division	45 / 2 >>> 22 (not 22.5)
%	Modulus (Remainder)	45 % 2 >>> 1

```
int   value1 = 43, value2 = 10, sum , difference, product, quotient, modulus ;

sum      =          value1   +   value 2;
difference =   value1   –   value 2;
product =        value1  *  value2;
quotient =   value1  /  value2;
modulus =   value1  %   value2;

System.out.println("The sum is      "  + sum);
System.out.println("The difference is      "  +  difference);
System.out.println("The  product is      "  +  product);
System.out.println("The quotient      "  +  quotient);
System.out.println("The modulus is      "  +  modulus);
```

or

```
System.out.println("The sum is      "  + ( value1 + value2) );
System.out.println("The difference is      "  +  (value1 – value2 ) );
System.out.println("The  product is      "  +  (value1 * value2) );
System.out.println("The quotient      "  +   (value1 / value2));
System.out.println("The modulus is      "  +  (value1 % value2));
```

Pre & post increment

More example, the following code sets both `i` and `j` to 2:

```
i = 1;
j = ++i;
```

But these lines set `i` to 2 and `j` to 1:

```
i = 1;
j = i++;
```


What is the value of the following

a = 5;
b = 2 + (a++);

Pre & post decrement

More example, the following code sets both `i` and `j` to 0:

```
i = 1;
j = --i;
```

But these lines set `i` to 0 and `j` to 1:

```
i = 1;
j = i--;
```

What is the value of the following

a = 5;
b = 2 + (--a);

Boolean Variables

- Can hold only one of two values : ***True*** or ***false***

Comparison Operators

Operator	Description	True Example	False Example
<	Less Than	3 < 8	3 > 8
>	Greater Than	8 > 3	8 > 3
==	Equal To	3 == 3	3 == 4
<=	Less Than or Equal to	3 <= 3	7 <= 5
>=	Greater Than or Equal to	8 >= 2	2 >= 8
!=	Not Equal to	4 != 2	54 == 54

```
boolean    isItSunny     =   false;
boolean    isItRaining   =   true;
boolean      isEightBigger   =  ( 8 > 4 );
                                    (  true is stored in the variable )
boolean      isEightBigger   =  ( 8 >9 );
                                    (  false is stored in the variable )

if (o != null) ...;          // The not equals operator
while(i < a.length) ...;     // The less than

if (x < 10 && y > 3) ... // If both comparisons are true

f (data != null && i < data.length && data[i] != -1) ...

if (x < 10 || y > 3) ... // If or comparisons

if (!(x > y && y > z))
```

Floating-Point Data Types

- **Floating-point** numbers contain decimal positions
- A **float** data type can hold values up to 6 or 7 significant digits of accuracy
- A **double** data type can hold 14 or 15 significant digits of accuracy

Table 2-4 Limits on floating-point values

Type	Minimum	Maximum	Size in Bytes
Float	$-3.4 * 10^{38}$	$3.4 * 10^{38}$	4
Double	$-1.7 * 10^{308}$	$1.7 * 10^{308}$	8

Examples :

```
123.45
0.0
.01
1.2345E02       // 1.2345 * 10^2, or 123.45
1e-6            // 1 * 10^-6, or 0.000001
6.02e23         // Avagadro's Number: 6.02 * 10^23
```

Floating-point literals are `double` values by default. To include a `float` value literally in a program, follow the number by the character `f` or `F`:

```
double d = 6.02E23;
float f = 6.02e23f;
```

Unifying Type

- When performing arithmetic operation with operands of the unlike types, the Java programming language chooses a ***unifying*** type for the result. The Java programming language implicitly (***automatically***) converts nonconforming operand to unifying type

Order for Implicitly Establishing Unifying Type

1. double
2. float
3. long
4. int
5. short
6. byte

```
int  hoursWorked = 20;
double payRate  =  10;
double  totalPay =  hoursWorked * payRate;
```

Converts int to double because double supersedes int.

The result of **adding** a ***short*** and a ***int*** is an ***int***

Type Casting

- The unifying type can be overridden by explicitly stating a type cast
- Place the desired type result in parentheses followed by the variable or constant to be cast
- Example:

```
double  bankbalance  =  189.66;

float weeklybudget = (float) bankbalance /4;       >>>>>  47.415

int dollars =  (int) weeklybudget;        >>>>>>>>>>     47
```

More Examples

```
int i = 13;
byte b = (byte) i;   // Force the int to be  a byte
i = (int) 13.456;     //Force this double literal to int 13
```

Constants

Variable whose value remain unchanged during program execution is called **constant.** For example, the value of π which is equivalent to 3.14. in order to define a constant variable, follow the following definition :-

```
final double Pi = 3.14;        //  the value of pi will  is fixed and can not be
                               //  changed  during  program execution.

final  int  Meter = 100 ;  //  1 meter is 100 centimeter
```

Working with the char Data Type

- char data type is used to hold a single character
- Uses ***single*** quotation marks

```
char   firstInitial     =  'h';
char   charnumber   =  '9';
```


String Data Structures

- A string can contain a string of characters
- Uses ***double*** quotation marks

```
string    lastName   =  "Gholoom";
```

One can store any characters in the char field even nonprinting one suh as tabs, backspace .. etc. to store these characters, you must use a an

escape sequence

Table 2-5 Common escape sequences

Escape Sequence	Description
\b	Backspace
\t	Tab
\n	Newline or linefeed
\f	Form feed
\r	Carriage return
\"	Double quotation mark
\'	Single quotation mark
\\	Backslash

```
System.out.println("\nthis is new line \nthis is another line");
System.out.println("This to demo \thow\tthis\twill\twork");
```

ASCII and Unicode

- Both are character sets
- Java programming language uses Unicode
- Unicode is a 16 bit coding scheme
- ASCII is an 8 bit coding scheme
- ASCII is most widely used coding scheme

Key Terms

- **associatively** – A property of arithmetic operators, such that they are evaluated left to right.
- **arithmetic expression** – Constructed by using arithmetic operators and operands.
- **arithmetic operators** – Operators such as + (addition) and – (subtraction).
- **assignment operator** – The equal sign in Java.
- **binary operator** – An operator that has two operands.
- **boolean** – A data type that deals with logical values and has the values `true` and `false`.
- **cast operator** – An operator that allows for explicit type conversion. Also called type conversion or type casting.
- **character string** – A string such as "Hello"; also called a string literal or a string constant.
- **class** – The basic unit of a Java program.
- **collating sequence** – A nonnegative integer value in the Unicode character set.
- **computer program** – A sequence of statements to accomplish a task.
- **constant** – A specific value of a given data type, which does not change during program execution.
- **data type** – A set of values together with a set of operations on those values.
- **decimal expression** – All operands in an expression are decimal numbers.
- **decrement operator** – The operator `--`, which decreases the value of a variable by one.
- **empty string** – A string with no characters.
- **floating-point** – A data type that deals with decimal numbers.
- **floating-point expression** – All operands in an expression are floating-point numbers.
- **floating-point notation** – A form of scientific notation used by Java.
- **identifier** – Name of a thing such as a variable, constant, method, or class.
- **implicit type coercion** – When a value of one data type is automatically treated as another data type.
- **increment operator** – The operator `++`, which increases the value of a variable by one.
- **initialize** – To place a value in a variable the first time.
- **integral** – A data type that deals with integers and characters.
- **integral expression** – All operands in an expression are integers.
- **keywords** – Reserved words, which cannot be used for anything other than their intended use.
- **length** – The length of a string is the number of characters in it.
- **literals** – A specific value of a given data type.
- **logical expression** – An expression that evaluates to `true` or `false`. Also called a Boolean expression.
- **mixed expression** – An expression that has operands of different data types.
- **modulus operator** – Also called mod and represented by the symbol %, which computes the remainder of a division operation.
- **named constant** – A memory location whose content is not allowed to change during program execution.

- **null string** – A string that contains no characters.
- **operands** – Constants and variables in an arithmetic expression.
- **precision** – The maximum number of significant digits.
- **predefined method** – A method that has been provided as part of the Java Library.
- **programming** – The process of planning and creating a program.
- **programming language** – A set of rules, symbols, and special words used to construct programs.
- **semantics** – The set of rules that give language a meaning.
- **single-line comments** – Comments that begin with `//` are placed anywhere in the line.
- **statement terminator** – In Java, the semicolon.
- **token** – The smallest individual unit of a program written in any programming language.
- **unary operator** – An operator that has only one operand.

Additional Resources

1. An article about type casting:
   www.roseindia.net/java/beginners/TypeCasting.shtml

2. Articles about data types in Java:
   http://java.sun.com/docs/books/tutorial/java/nutsandbolts/datatypes.html

3. Java language basics summary:
   www.javaprepare.com/notes/funda.html

Comprehensive Example

```
public class ArithmeticAndLogicalOperator {
   // This Program is written By Husain Gholoom
   // The Function of this program is to demonstrate few functions of
   // arithmetic and logical operators, using math library , type casting and
   // few of escape sequences

  public static void main(String[] args) {
      int value1 = 43, value2 = 10, sum, difference, product, quotient,modulus;
      boolean isItPayday = false;
      boolean areYouBroke = true;
      boolean sixIsBigger  = ( 6 > 5 );
      boolean sevenIsSmaller  = ( 7 >  9 );
      sum = value1 + value2;
      product = value1 * value2;
      difference = value1 - value2;
      quotient = value1 / value2;
      modulus = value1 % value2;

      System.out.println("The sum is   " + sum);
      System.out.println("The product  is  " + product);
      System.out.println("The difference  is  " + difference);
      System.out.println("The quotient  is   "  + quotient);
      System.out.println("The modulus   is   "  + modulus);

      System.out.println("The sum is   " + (value1 + value2));
      System.out.println("The product  is  " + (value1 * value2));
      System.out.println("The difference  is  " +(value1 - value2));
      System.out.println("The quotient  is   "  + (value1 / value2));
      System.out.println("The modulus   is   "  + (value1 % value2));

      System.out.println(value1 + value2 + "   Is the sum of Value1 + value2");
      System.out.println(((value1 + value2) * value2) +"  IS example of nested
brackets");

      System.out.println("Is It Pay Day   =   "  + isItPayday);
      System.out.println("Are You Broke   =   "  + areYouBroke );
      System.out.println("isSixBigger    "  + sixIsBigger);
      System.out.println("sevenIsSmaller   "  + sevenIsSmaller);

     double bankBalance = 189.66;
     float weeklyBudget = (float) bankBalance / 4;
     System.out.println("Float weeklyBudget   =   "  + weeklyBudget);
      int dollars = (int) weeklyBudget ;
      System.out.println("int  dollars   =   "  + dollars);

      System.out.println("\nThis is one line\nThis is another line ");
```


```
System.out.println("This shows\thow\ttabs\tworks");

System.out.println("Thee square root of val is:"+Math.sqrt(30));
System.out.println(" the sin of val is:"+Math.sin(100));
System.out.println(" the cosine of val is:"+Math.cos(100));
System.out.println("The floor value is:"+Math.floor(44.7));
System.out.println("The ceiling value is:"+Math.ceil(44.7));
System.out.println("The round value is:"+Math.round(44.7));
System.out.println(" the max valueis:"+Math.max(75,70));
System.out.println("the min value"+Math.min(75,70));



System.out.println(Math.round(1.49));
System.out.println(Math.round(1.50));
System.out.println(Math.round(-1.49));
System.out.println(Math.round(-1.50));

System.out.println(Math.floor(1.49));
System.out.println(Math.floor(1.50));
System.out.println(Math.floor(-1.49));
System.out.println(Math.floor(-1.50));


System.out.println(Math.ceil(1.49));
System.out.println(Math.ceil(1.50));
System.out.println(Math.ceil(-1.49));
System.out.println(Math.ceil(-1.50));


  }
}
```

Lab Assignment

1) Write a program that produces the following banner

```
**********************************
********* Hello ****************
*** My Name is Husain Gholoom**
 ============================
*** Date : 17 - 10 - 2009**********
** This is My First Java Program **
**********************************
============================
**********************************
```