



# Compiling and Running Java Programs

Download Java compiler form <http://java.sun.com>

Also download English Java 2 SDK documentation from Sun Java page

To run Java program

Save file with .java extension using text editor

Use dos prompt, go to the directory where file is saved

Enter javac filename.java

.class version of the file is created

Errors could be as a result of:

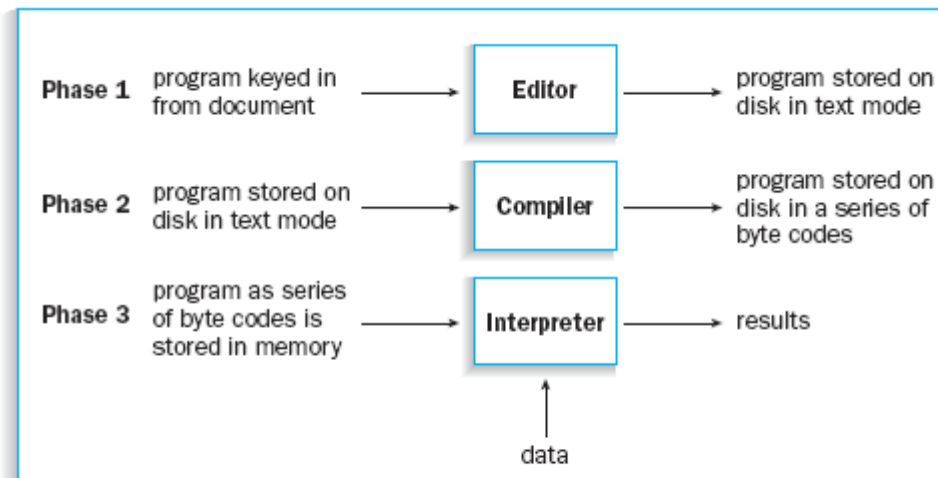
You have not modified the path entry of your autoexec.bat file correctly and the computer cannot execute the command javac to compile your program.

You have made a mistake when typing the program, and the syntax of at least one statement might be incorrect.

To execute the java object code, issue the following command in the same window where you compiled the program:

java filename

- If a program fails during the execution phase, it must be stopped from any further execution. If modifications to the program are required, it is necessary to perform the amendments at phase 1, and repeat phases 2 and 3.



**Figure 0.9** Three phases of program implementation



# SDK Tools

- **javac**—the **Java Language Compiler that you use to compile programs** written in the Java programming language into bytecodes.
- **java**—the **Java Interpreter that you use to run programs written in the Java** programming language.
- SDK tool documentation gives a full explanation of the function of each tool.



# Data

- *Data is the name given to characters and quantities operated upon by a computer.*
- *A computer program consists of a series of instructions for the computer to execute and provides a method for processing data. The data from bank transactions can be processed by a computer into information for bank statements.*
- *we can identify four data types: integer (a positive or negative whole number), real (a positive or negative number with a decimal fraction), character (a single character), and a string (a group of characters).*
- *A variable is a named memory location that can hold a particular type of data. For example, the following statement in a Java program will reserve a memory location called count that can hold data of type integer.*
  - `int count;`
- *This statement is called a type declaration since it declares that the variable count must hold data of type integer. We say that the data type of count is int.*



## Data types: character

- The type declaration for a character is declared in Java as char.
- E.g `char myChar = 'A';`
- declares a variable called myChar, of type char which can hold a character as a value, and is initialized to the character 'A'

# Integer numbers

- **An integer is stored as a binary number. In Java there are**
- several integer data types. The one we will be using most frequently is declared as `int` and uses four bytes of computer memory. Therefore, an `int` can represent any of  $2^{32} = 4,294,967,296$  different integers. The range of `int` values is `-2,147,483,648` to `+2,147,483,647`.
- If you want to store an integer number that lies outside of the range for `int` types, then use the Java type `long`. These numbers are represented with eight bytes (64 bits) and have a range of `-9,223,372,036,854,775,808` to `+9,223,372,036,854,775,807`.
- The use of a plus sign (+) is optional for positive integer literals. All decimal integer literals must begin with a digit in the range from 1 to 9 after the sign (if a sign is present). Integer literals must not begin with 0 (zero). A long integer literal, either decimal or hexadecimal, has the character `l` or `L` appended immediately after the number.

# Data types: Real numbers

- A real number is stored in the computer memory in two parts, a mantissa (the fractional part) and an exponent (the power to which the base of the number must be raised in order to give the correct value of the number when multiplied by the mantissa). For example, 437.875 can be rewritten as  $0.437875 \times 10^3$ , where 0.437875 is the mantissa and 3 is the exponent. A four-byte representation of a real number will give a maximum value of  $\pm 3.40282347 \times 10^{38}$  and the smallest value as  $\pm 1.40239846 \times 10^{45}$ . The majority of decimal fractions
- do not convert exactly into binary fractions; therefore, the representation of a real number is not always accurate.
- In Java, the type real is declared as float.
- If the float range is too restrictive for the real numbers being stored, Java can store much larger real numbers using the type double. The number of bytes used to store a double-precision number is increased to eight. This increase in storage space will give a maximum value of  $\pm 1.79769313486231570 \times 10^{308}$  and the smallest value as  $\pm 4.94065645841246544 \times 10^{324}$ .
- A real-number literal can be written in one of two ways. For example, the literal 123.456 can be written as depicted or using a scientific notation 1.23456E+2. The character E represents the base 10, so the number can be interpreted as  $1.23456 \times 10^2$ , which, of course, evaluates to 123.456 when you adjust the decimal point.
- The data types char, int, long, float, and double are known as *primitive* data types. Only a selection of the primitive data types that you are likely to use in this course have been presented



# Identifiers

- To distinguish data in different areas of memory, we could give the data a name or use the numeric memory address of the first byte of the address in which the data is stored.
- An identifier may contain combinations of the letters of the alphabet (both uppercase A-Z and lowercase a-z), an underscore character `_`, a dollar sign `$`, and decimal digits 0-9. The identifier may start with any of these characters with the exception of a decimal digit.
- Java is a *case-sensitive language*, meaning that uppercase letters and lowercase
- letters of the alphabet are treated as different letters. Identifiers can normally be of any practicable length.



# Identifiers (contd.)

- An identifier must not be the same as the Java keywords listed below
- A programmer should always compose identifiers so they convey meaning of the data that they represent
- When an identifier is constructed from more than one word, each successive word should begin with an uppercase letter; an identifier should be easy to read, and its meaning should be clear.

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

we will use keywords in program statements, but not as identifiers. The words in Figure 1.7 that appear in black are reserved by Java, but are currently unused.

Examples of legal identifiers are subTotal, salesTax, unitCost, and rateOfPay.

# Variables and Constants

- If the values of the data in the storage cells can be changed by the instructions in a computer program, the values of the data will vary, and the data identifiers are known as *variables*. *The syntax for making a variable declaration is:*
  - *data-type identifier;*
  - *data-type identifier-list;*
- Examples:
  - `int weightLimit;`
  - `int parkingTime;`
  - `float balance;`
  - `float costOfGas;`
  - `float valueOfShares;`
- When variables are of the same type, you may declare the type followed by a list of identifiers separated by commas, for example:
  - `int weightLimit, parkingTime;`
  - `float balance, costOfGas, valueOfShares;`

# Variables and Constants (contd.)

- Variables can be initialized (set to an initial value) by the programmer at their point of declaration, using the following syntax:
  - *data-type identifier = literal;*
- Example:
  - `char parkingSymbol= 'P';`
  - `int weightLimit = 10;`
  - `float balance = 1225.11f;`
- Note that the `f` signifies that the numeric literal is stored as a single-precision value since numeric literals are stored in double-precision form by default.
- The syntax for a constant declaration follows:
  - *Final data-type identifier = literal;*
- Such constants can be declared in Java as follows.
  - `final float SALES_TAX = 0.05f;`
  - `final double PI = 3.14159;`
  - `final float G = 9.80665f;`

# The Format of a Simple Program

*heading giving details of the name and purpose of the program*

```
import list
```

```
class name
```

```
{
```

```
    main method
```

```
    {
```

```
        declarations of constants
```

```
        declarations of variables
```

```
        program statements
```

```
    }
```

```
}
```

# Arithmetic Operations

- The following symbols are used to perform arithmetic on data stored in memory
- + unary plus
- - unary minus
- Unary operators have one operand and are used to represent positive or negative numbers.
- **Binary Multiplicative Operators**
- \* multiplication
- / division
- % remainder
- Note that the % operator will compute the remainder after the division of two numeric values; for example,  $33\%16$  computes the remainder 1 after 33 is divided by 16;  $16\%33$  computes the remainder 16 after 16 is divided by 33. (The result of the division is 0, remainder 16.)
- **Binary Additive Operators**
- + addition
- - subtraction
- Both multiplicative and additive operators have two operands.

- Example

- $total = subTotal + tax$

Before

5	16	12
total	subTotal	tax

After

28	16	12
total	subTotal	tax

Result of computation  
 $total = subTotal + tax;$

- adds the contents of subTotal to the contents of tax and stores the result in total, destroying or overwriting the previous contents of total.

- More examples:

- $total = score - penalty;$
  - $tax = price * taxRate;$
  - $time = distance / speed;$
  - $result = sum \% divisor;$
  - $counter = counter + 1;$
  - Try your hands on exercises from <http://adapt2.sis.pitt.edu/kt/ensemble/java.html>

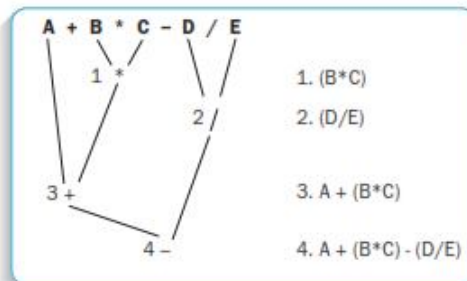
Before	100	100	4	
	total	score	penalty	
After	96	100	4	Result of computation $total = score - penalty;$
	total	score	penalty	
Before	1.5	20.0	0.05	
	tax	price	taxRate	
After	1.0	20.0	0.05	Result of computation $tax = price * taxRate;$
	tax	price	taxRate	
Before	50.0	120.0	60.0	
	time	distance	speed	
After	2.0	120.0	60.0	Result of computation $time = distance / speed;$
	time	distance	speed	
Before	19	37	15	
	result	sum	divisor	
After	7	37	15	Result of computation $result = sum \% divisor;$
	result	sum	divisor	
Before	3			
	counter			
After	4			Result of computation $counter = counter + 1;$
	counter			

# Operator Precedence

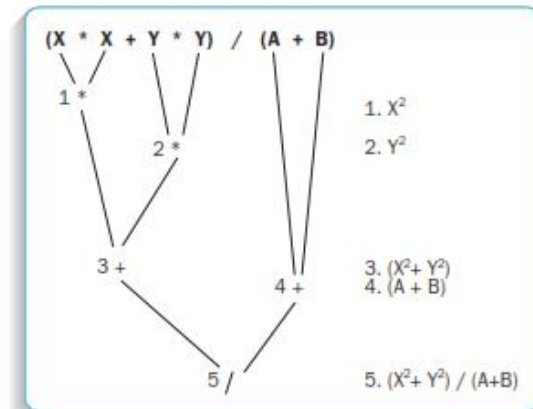
- Expressions are evaluated by using operators with a higher priority before those of a lower priority. Generally, operators of the same priority, are evaluated from left to right. Expressions in parenthesis will be evaluated before non-parenthesized expressions.
- Example: the algebraic expression  $\frac{UV}{WX}$  can be written in Java as `U*V/W/X`;
- however, it is easier to understand  $(U*V)/(W*X)$ .

Priority level	Operator	Operand type(s)	Associativity	Operation performed
1	+ - (type)	arithmetic any	R	unary plus, unary minus cast
2	* / %	arithmetic	L	multiplication, division, remainder
3	+ -	arithmetic	L	addition, subtraction
13	=	variable any	R	assignment

Priority levels of operators



Evaluation of  $A + B * C - D / E$

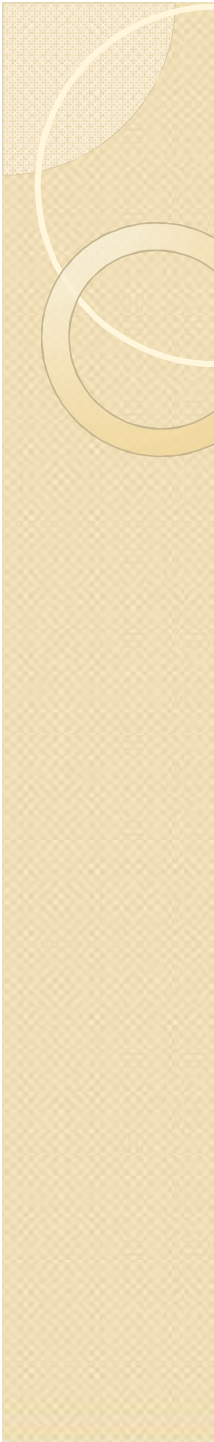


Evaluation of  $(X * X + Y * Y) / (A + B)$

Similarly,  $x^2 + y^2 + \frac{4}{z^2}(x + y)$  can be written in Java as

$$(X*X) + (Y*Y) + 4 * (X+Y) / (Z*Z).$$

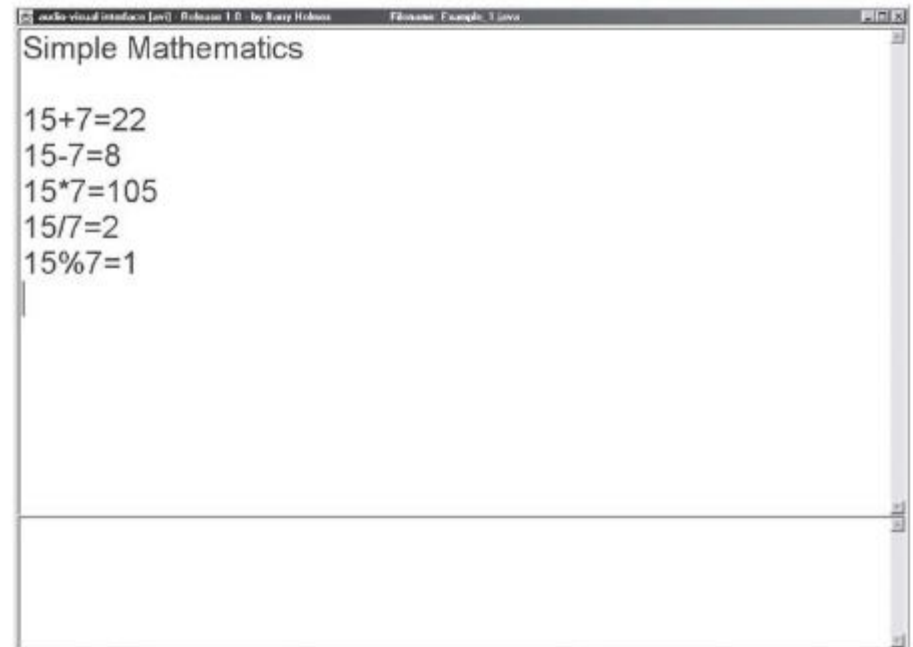
*With the exception of the % remainder operator, which must have integer operands, all other operators can have integer or real operands or a mixture of both types. In a division, if both the operands are integer, then the result will also be an integer, i.e., the fractional remainder in the result will be truncated.*

- 
- Casting: When operands are of different types, one or more of the operands must be converted to a type that can safely accommodate all values before any arithmetic can be performed
  - SYNTAX:
    - Cast operation: (data-type)expression;
  - For example,
    - float money = 158.05;
    - int looseChange = 275;
    - money = (float) looseChange;
  - The cast expression (float)looseChange is used to convert the value of the Variable looseChange to a number of type float. This does not imply that looseChange has altered its type from int to float; only the value has been converted to type float for the purpose of the assignment



# Example program

```
// program to calculate the sum, difference, product, quotient,  
// and remainder of two numbers  
  
import avi.*;  
  
public class Example_3  
{  
    public static void main(String[] args)  
    {  
        // declare variables  
        int first;  
        int second;  
        int sum;  
        int difference;  
        int product;  
        int quotient;  
        int remainder;  
  
        // assign values to variables  
        first = 15;  
        second = 7;  
  
        // perform computations  
        sum = first+second;  
        difference = first - second;  
        product = first*second;  
        quotient = first/second;  
        remainder = first%second;  
  
        // output results of calculations  
        Window screen = new Window("Example_1.java","plain","blue",36);  
        screen.showWindow();  
  
        screen.write("Simple Mathematics\n\n");  
        screen.write(first+" "+second+"="+sum+"\n");  
        screen.write(first+" - "+second+"="+difference+"\n");  
        screen.write(first+" * "+second+"="+product+"\n");  
        screen.write(first+" / "+second+"="+quotient+"\n");  
        screen.write(first+" % "+second+"="+remainder+"\n");  
    }  
}
```



The screenshot shows a Java Swing window titled "Simple Mathematics". The window contains the following text:

```
15+7=22  
15-7=8  
15*7=105  
15/7=2  
15%7=1
```

