Variables in Java

A **variable** is a symbolic name for a location **in memory**. Memory is used to store **string** or **numeric** data. The value stored in memory (represented by a variable) can be modified and read during the execution of the Java program. To use a variable in Java, it must first be defined. Defining a variable in Java is done by stating the variable type and by giving it a variable name (identifier).

To define a variable in Java use the following format: **Type Identifier;**

Example

To create an **Integer** {type} variable in Java with the variable name {identifier} of **age**, write the following code:

int age;

This results in an **integer** memory location (named **age**) being set aside for use later in the Java program.

Alternatively, you can create many variables *of the same type* at once, as shown below:

int age, studNum, intNum, mark;

NOTE:

After a variable is created, it can be manipulated or used by the Java program. For example, the value stored in an integer variable named age can be changed with the following code:

age = 21;

This stores a value of **21** in the integer memory location referred to as **age**.

Rules for Variable Names

1. Variable names are made up from a series of characters that begins with a letter (no spaces or punctuation).
2. Some words, called "**keywords**" or "**reserved words**" are reserved by Java; they are part of the Java language and you may not use them as variable names. These are keywords used by the compiler to control your program. Keywords include:
   - system, println, if, case
   - while,for,main,
   - int, long, short, double, float
   - and many more

3. Variable names must be unique; do not declare a variable with the same name more than once. NOTE: A variable may have the same name as a variable whose declaration appears in a different scope. More on this topic will come later in the course.
4. Good variable names tell you what the variables are representing in memory. Using good names makes it easier to understand the flow of your program. As programs get larger it becomes increasingly difficult to track variables with non-meaningful names such as “x”. If you have a variable that is meant to track the age of a person, then call it “age”. This will save you a lot of time when debugging a large program.
5. JAVA is case-sensitive! This means that UPPERCASE and lowercase letters are considered to be different. A variable named **age** is different from **AGE**, which is different from **Age** and **AGe**.
6. Declare all local variables at the beginning of a method, and not as you need them. As programs get larger, it will become increasingly difficult to track your variables if they are not all grouped in a common place.

NOTE: It is important to initialize a variable (set it to some value) before you use it in a calculation or print it. If you do not initialize the variable then any random number could be stored in that variable and the results of a calculation using that variable may be unpredictable.

JAVA Variable Types

TYPE	**DESCRIPTION**	**# bytes**	**RANGE of Values**
boolean	Boolean value	1	only **TRUE** or **FALSE**
byte		1	0 to 255
char	characters	2	all displayable ASCII characters
short	integers (no decimals)	2	-32,767 to +32,767
int	integers (no decimals) (larger range)	4	-2,147,483,647 to +2,147,483,647
long	integers (no decimals) (larger range)	8	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
float	real numbers (decimal #)	4	virtually infinite
double	real numbers (decimal #) (larger range and precision)	8	virtually infinite

Scope of a Variable

When using variables in Java, it is important to know the **SCOPE** of the variables you are using.

The **SCOPE** of a variable refers to **WHERE** and **WHEN** a variable can be USED.

There are two types of SCOPES:

1. The **GLOBAL SCOPE** – these are GLOBAL VARIABLES and can be used in any part of the Java program.
2. The **LOCAL SCOPE –** these are LOCAL VARIABLES and can only be used in the part of the Java program where they are defined.

GLOBAL VARIABLES are variables that:

- are declared in a global context outside of functions or blocks of code.
- may be used anywhere in the program and can be used in all functions or blocks of code.

LOCAL VARIABLES are variables that:

- MUST be **declared** in the context (method) that they are used in.
- can be used *only* in the context (method) they are **declared in.**
- are lost when their context is completed.

Simple Arithmetic in Java

In Java, doing simple arithmetic is very much like doing arithmetic in other languages such as Basic and Pascal, using the +, -, * and / operators (plus, minus, times and divide).

Generally, you will store the result of the mathematical expression in a memory location represented by a variable.

An assignment statement is used to store a value or string in memory or to perform a calculation and store the result in memory by using the assignment operator (=), which causes the operand **(the variable)** on the left side of the assignment operator to have its value changed to the value on the right side of the assignment operator.

Example

The expressions...

age = 21;

... assigns the value 21 to the memory location age.

avg = (80 + 56 + 75 + 82)/4;

... assigns the average calculated to the memory location avg.

NOTE: Be careful not to confuse the assignment operator (=) with the test for equivalence(==) discussed later in the course.

The Modulus Operator

The division of two integers like 27 divided by 5 will result in an answer of 5. The modulus operator tells you the remainder. To find the remainder of the division of two integers like 27 divided by 5, you would use the modulus division operator **%**.

Example

remainder = 27 % 5; // divides 27 by 5 and determines the remainder.

This would result in the 2 being placed in a variable called remainder.

Auto incrementing and auto decrementing variables

The most common mathematical formula used in programming languages is to take a variable and add (or subtract) the value 1 to it, and then store the result in the same variable. Adding 1 is called incrementing. Decreasing by 1 is called decrementing.

- The increment operator (++) increases the value of the variable by 1.
- The decrement operator (--) decreases the value of the variable by 1.

Example

If you have a variable **x** and you want to increment it, you would use this statement:

x++; // increases the value stored in x by one. This is the same as the statement x=x+1;

If you have a variable x and you want to decrement it, you would use this statement:

x--; // decreases the value stored in x by one. This is the same as the statement x=x-1;

Strings in Java

A **String** is a data type that can hold one or more ASCII characters (letters, numbers or special characters such as !).

It is possible to store no characters at all in a string - this is called the null string "".

To define a **string variable** use the same process as defining a numeric (int, double, etc.) variable as follows: **Type Identifier;**

Example

To define a **String** {type} variable in Java with the variable name {identifier} of **lastName**:

String lastName;

This results in a **String** memory location (named **lastName**) being set aside for use later in the Java program.

Alternatively, you can create many String variables at once, for example:

String lastName, firstName, address, city;

String Literals vs. String Variables

In Java, there are ***String Variables*** and ***String Literals***.

String Literals are Strings that are constant (they do not change) and are not stored in a variable. They are used to pass data into a method or to set the value of a variable. String literals are used all the time in methods:

For example ...
System.out.println("**hello**");
... where the word "hello" is a string literal passed into the println method.

String Variables are variables that can hold one or more ASCII characters (or none). String variables are used to store data that is non-numeric—you cannot do math operations on that data. Like every other variable in Java, you must create the String variable before you are able to use it.

Example

String teamName;
// creates a String variable called teamName ** Note: String has a capital S

teamName = "Toronto Maple Leafs";
// sets the value of the String variable

System.out.println("A great team is the " + teamName);
// displays "*A great team is the Toronto Maple Leafs*"

To store data in a String, that data must come from:

- a string literal (e.g., **name1 = "Toronto";**)
- another string (e.g., **name2 = name1;**)
- a joining of other strings (e.g., **name = name1 + name2;**)

Example

```
name1 = "Toronto";
name2 = "Maple Leafs";
name = name1 + name2;
System.out.println(name);    //Displays Toronto Maple Leafs
```

Java Comment Statements

Internal Documentation is an important part of programming. Leaving messages to others looking at the *source code* of your program about who created the program and what the program does is important. Comment lines are considered **non-executable statements** and as such are ignored by the Java Compiler and Interpreter when the JAVA program is executed.

There are two main ways to show internal documentation or *comments* in Java:

1. Multiple line comments are comments that are more than one line in length. Multi-line comments start with /* and end with */. Everything between the opening and closing of the comment is ignored by the Java Compiler and Interpreter.

Example

```
/*
* This is an example of a multiple line comment
* that shows information over many lines. It begins with a
* slash and an asterisk and ends with an asterisk and a slash
*/
```

2. Single line comments use two forward slashes,// to specify that a comment is following on this programming line. All information on the line following the comment opening is ignored by the Java Compiler and Interpreter.

Example

```
// This comment type is used for short one line comments
```

Generally, multiple line comments are used for large comments such as to give information at the top of a program. Single line comments are used for specific comments about what is going on in a section of a program.

Include a comment similar to the following at the top of every program:

```
/*
* This page was created by John Doe
* on November 1
* to "give a detailed program description here"
*/
```

You may also choose to use single line comments to document important aspects of your program. A couple of key areas where you should do this are: at or near the beginning of each method; and each loop to describe what those blocks of code do.

```
// Calculate Payroll Section
```