|  |  |
| --- | --- |
| **Topic** | **Variable Declaration and Data types**` |
| **Learning Outcomes** | Variables are the key to making Java programs general purpose. It change the output of your program you can also make computation more flexible and you can almost perform any kind of computations. In this lesson you will be able to:1. Properly declare variables using different data types
2. Know the different types of variable declarations
3. Differentiate variables from constants
4. Perform Type casting
 |
| **References** | Lemay, L., Perkins, C. L. (1996), Teach Yourself JAVA in 21 days,  Indianapolis, Indiana, Sams.net.Doug Lowe, Java® All-in-One For Dummies®, 4th Edition), New Jersey,  John Wiley & Sons, Inc.Barry Burd Ph.D., Java® For Dummies®, 7th Edition, New Jersey,  John Wiley & Sons, Inc.David J. Eck, Introduction to Programming Using Java Version 5.0, December 2006 Geneva, NY, Hobart and William Smith CollegesJester Lhee I. Pandio, Computer Programming 1, Global Port Taguig, STI College |

**Discussion**

1. **Declaring Variables**

In Java, you must explicitly declare all variables before using them. This rule is in contrast to some languages — most notably Basic and earlier versions of Visual Basic, which let you use variables that have not been automatically declared.

**Variables and Constants**

**Variable**

Is a name for a memory location that stores a specific value, such as numbers and letters. A variable is an identifier. It can hold only one (1) value at a time, but its value may change during program execution.

 The basic form of a variable declaration

 Type identifier or name;

 Example:

 int ***age***;

 String ***lastName***;

 double ***radius***;

 int ***height***, ***width***, ***length***;

 double ***areaOfRectangle***, ***areaOfSquare***;

 Note:

 Refer to the previous lessons for the detailed explanation of the data types

**Types of Variable declarations**

* Class Variables
* Instance Variables
* Local Variables
* Parameter Variables

**Class Variable**

A class variable is a variable that any method in a class can access, including static methods such as main.

When declaring a class variable, you have two basic rules to follow:

* + - You must place the declaration within the body of the class but not within any of the class methods.
		- You must include the word ***static*** in the declaration. The word static comes before the variable type.

Example

 import static java.lang.System.out;

public class OutputClass {

 static String name;

 public OutputClass(String yourName) { // Constructor with parameters

 out.println("Hello !, " + yourName);

 }

 public static void main(String args[]){

 name = "Jhun";

 new OutputClass(name);

 }

 public void setName(String lastname){ // Method declaration

 name = lastname;

 }

}

In the above example, the variable name is declared as class variable due to the presence of the reserve word static thus, the variable name is used in the main method and the setName method

Note:

1. Constructor, methods and parameter will be discussed in-depth in our succeeding discussions
2. You can declare a class variable at the end of the class.

**Instance Variables**

An instance variable is similar to a class variable but does not specify the word static in its declaration. As the name suggests, instance variables are associated with instances of classes. As a result, you can use them only when you create an instance of a class. Because static methods are not associated with an instance of the class, you cannot use an instance variable in a static method and that includes the main method.

The following example program won’t compile:

public class HelloApp{ String helloMessage; // error -- should use static keyword

public static void main(String[] args) {

helloMessage = "Hello, World!";

System.out.println(helloMessage); // will not compile

 }

}

If you attempt to compile this program, you get the following error messages:

C:\Users\STI\Desktop\HelloWorld.java:6: error: non-static variable helloMessage cannot be referenced from a static context

helloMessage = "Hello, World!";

 ^

C:\Users\STI\Desktop\HelloWorld.java:7: error: non-static variable helloMessage cannot be referenced from a static context

 System.out.println(helloMessage); // will not compile

 ^

**Example of usage of instance variables**

**public class InstanceVariables {**

 String studentName;

 String address;

 String program;

 **public void setName(String name){ //mutator method**

 **studentName = name;**

 **}**

 **public String getName(){ // accessor method**

 **return studentName;**

 **}**

**}**

**public class Instantiation {**

 **public static void main(String args[]){**

InstanceVariables var = new InstanceVariables(); //Instantiation

 var.studentName = "Justin";

 **}**

**}**

In the *InstanceVariables* class we declare three instance variables *studentName, address, program*.

In the Instantiation class we created an **instance** of *InstanceVariables* class and assign a value to the **instance variable** *studentName*. We are able to use the *studentName* outside its declaration using the instance of the class, hence its name instance variable.

Note:

1. You can use **instance variable** within a non-static method declared within the class where the instance variable is declared.
2. Mutator and Accessor method will be discussed later in our succeeding topics

**Declaring local variables**

A local variable is a variable declared within the body of a method. Then you can use the variable only within that method. Other methods in the class are not even aware that the variable exists.

 Example

**public class LocalVariables {**

 **public static void main(String [] args){**

 String studentName = "Matthew"; // Declaration with initialization

 String address = "San Fernando";

 byte age = 18;

 System.out.println("My name is : " + studentName);

 System.out.println("My Address : " + address);

 System.out.println("My age is : " + age);

 **}**

**}**

As you can see in the program listing above the three variables can only be used inside the method where it is declared.

**Final variables (Constant)**

A final variable, also called a constant, is a variable whose value you can’t change after it’s been initialized.

It is a memory location whose value cannot be changed during program execution. When a created variable is stored in the memory, Java uses a named constant to instruct a program to mark the memory location as constant throughout program execution. To allocate memory, use Java’s declaration statements.

The syntax to declare a named constant is:

**final data\_type variable\_name = value;**

 Example

 final double PI = 3.1416;

 final int numberOfDays = 7;

The final is a Java reserved word. This keyword is used to specify that the value stored in the variable\_name is fixed and cannot be changed.

During the execution of the program, the value of both PI and numberOfDays cannot be changed.

Although you can create final local variables, most final variables are class or instance variables. To create a final class variable (sometimes called a class constant), add static final (not final static) to the declaration, as follows:

static final WEEKDAYS = 5;

**Type casting**

This refers to converting a value from a specific type to a variable of another type

(note:booleans cannot be converted to numeric types).

 **There are two (2) types of conversion**

***Widening conversion* (implicit casting)**

The conversion of the lower precision data type to a value of a higher precision data type. This causes no loss of information, and the Java Virtual Machine (JVM) will perform the casting implicitly or automatically.

 byte 🡪 short 🡪 int 🡪 long 🡪 float 🡪 double

 Example

 byte x = 4;

 double a = x;

In the above example variable x is of type byte when variable x is assigned to variable a of type double the value of 4 becomes a double 4.0

 Sample program:

 public class TypeConversion {

public static void main(String[] args){

 byte x = 4;

 double a;

System.out.println("Value of x " + x);

a = x; // widening conversion

System.out.println(" Value of a " + a);

 }

}

**Narrowing conversion (explicit casting)**

The conversion of a higher precision data type into a value of a lower precision data type. This will typically involve loss of information.

The following is an example of Java’s strict type checking. The code will not compile, and an error will be generated:

float x = 6.82f;

int a = x; //cannot be converted or assignment is illegal

The casting is not done by JVM and should be made explicit by the programmer through a cast operator. The cast operator takes the following form:

 (data type) expression;

First, the expression is evaluated. Its value is then treated as a value of the type specified by data type.

The following is the example of explicit casting:

double x = 6.82f;

byte a = (int) x; //the value of variable a is 6

The conversion is done from higher-order data type to lower-order data type as follows

 Double 🡪 float 🡪 long 🡪 int 🡪 short 🡪 byte

**Wrapper Class**

Every primitive type has a corresponding class defined in the Java API class library. This class is sometimes called a wrapper class because it wraps a primitive value with the object-oriented equivalent of pretty wrapping paper and a bow to make the primitive type look and behave like an object.

|  |  |
| --- | --- |
| **Primitive Type** | **Wrapper Class** |
| int | Integer |
| short | Short |
| long | Long |
| byte | Byte |
| float | Float |
| double | Double |
| char | Character |
| boolean | Boolean |

 Instead of declaring a variable using the primitive data types, you can declare a variable using a wrapper class.

In declaring a variable using a wrapper, class the variable now can be used as an object.

 Example:

 Integer convert = 13;

 String strConvert;

 strConvert = convert.toString(convert); // convert here is used as an object.

Sample Program

public class WrapperClass {

 Integer convert = 13;

 String strConvert;

 public WrapperClass() {

 strConvert = ***convert****.*toString(***convert***); // convert is used as an object and a variable

 // the value is converted into a String.

 System.out.println("Converted Integer : " + strConvert);

 }

 public static void main(String[] args){

 new WrapperClass();

 }

}

Here is another example

 int a = 8;

 Integer a\_wrapper = new Integer (a);