Declaring, Initializing, and Using Variables

Learning Objectives

After completing this topic, you should be able to

- identify the uses and the syntax of variables
- declare variables

1. Introducing variables

You use variables for storing and retrieving data for your program. Objects store their individual states in fields. Fields are also called *instance variables* because their values are unique to each individual instance of a class.

The code example shows a `Shirt` class that declares several non-static fields, such as `price`, `shirtID`, and `colorCode` in the class.

When an object is instantiated from a class, these variables contain data specific to a particular object instance of the class.

For example, one instance of the `Shirt` class might have the value of 7 assigned to the `quantityInStock` non-static field, while another instance of the `Shirt` class might have the value of 100 assigned to the `quantityInStock` non-static field.

Code

```
public class Shirt {
    public int shirtID = 0; // Default ID for the shirt

    public String description = "-description required-"; // default

    // The color codes are R=Red, B=Blue, G=Green, U=Unset
    public char colorCode = 'U';

    public double price = 0.0; // Default price for all shirts
    public int quantityInStock = 0; // Default quantity for all shirts

    // This method displays the values for an item
    public void displayInformation() {
    System.out.println("Shirt ID: " + shirtID);
```

Your programs can also have variables defined within methods. These variables are called local variables because they are available only locally within the method in which they are declared.

Code

```
public void displayDescription {
    String displayString = "";
    displayString = "Shirt description: " + description;
    System.out.println(displayString);
}
```

Note

The terms "variables" or "fields" may be used to refer to variables. If the situation requires, the term "local variable" will be used when it applies.

Variables are used extensively in the Java programming language for these types of tasks:

Code

```
public class Shirt {
    public int shirtID = 0; // Default ID for the shirt

    public String description = "-description required-"; // default

    // The color codes are R=Red, B=Blue, G=Green, U=Unset
    public char colorCode = 'U';

    public double price = 0.0; // Default price for all shirts
    public int quantityInStock = 0; // Default quantity for all shirts

    // This method displays the values for an item
    public void displayInformation() {
    System.out.println("Shirt ID: " + shirtID);
```

- holding unique attribute data for an object instance, for example with the price and ID variables in the example shown
- assigning the value of one variable to another
- representing values within a mathematical expression
- printing variable values to the screen, and
- holding references to other objects

Attribute variable declaration and initialization follow the same general syntax.

Syntax

[*modifiers*] *type identifier = value*;

> *The modifiers represent several special Java technology keywords, such as* `public` *and* `private`*, that modify the access that other code has to a field. Modifiers are optional, but*

you can't use modifiers with local variables – that is, variables declared within methods. For now, all of the fields you create should have a `public` *modifier.*
The type represents the type of information or data held by the variable. Some variables contain characters, some contain numbers, and some are Boolean and can contain only one of two values. All variables must be assigned a type to indicate the type of information that they contain.
The identifier is the name you assign to the variable that is of type `type`*.*
The value is the value you want to assign to the variable. The value is optional because you do not need to assign a value to a variable at the time that you declare the variable.

This example shows the declarations for the fields in the `Shirt` class.

Code

```
public int shirtID = 0;
public String description = "-description required-";
public char colorCode = 'U';
public double price = 0.0;
public int quantityInStock = 0;
```

Question

In which section of the syntax for declaring and initializing a variable would you include the keywords `public` or `private`?

Options:

1. `type`
2. `modifiers`
3. `identifier`
4. `value`

Answer

Option 1: Incorrect. The `type` represents the type of information or data held by the variable. Some variables contain characters, some contain numbers, and some are Boolean and can contain only one of two values.

Option 2: Correct. The `modifiers` represent several special Java technology keywords, such as `public` and `private`, which modify the access that other code has to a field.

Option 3: Incorrect. The `identifier` is the name you assigned to the variable that is of a certain type.

Option 4: Incorrect. The `value` is the value you want to assign to the variable. It is optional because you do not need to assign a value to the variable at the time that you declare the variable.

Correct answer(s):

2. `modifiers`

You want to create a class containing several fields. You decide to declare the fields and initialize them in the class and then test it by running the CustomerTest program.

You begin by opening NetBeans.

Graphic

The NetBeans interface consists of a menu bar and toolbar. The pane in the top left of the screen has three tabs – Projects, Files, and Services – that display the physical components of the project. The Projects tab displays a number of nodes in a hierarchy, starting with Project05 at the top. It has two child nodes called Source Packages, which holds a number of java files, and libraries.

Next you create a new project from existing Java source, using the following values when you complete the New Project wizard:

Graphic

The NetBeans application is open and the New Java Project with Existing Sources wizard is displayed. The wizard consists of four steps: Choose Project, Name and Location, Existing Sources which is the current step, and Includes and Excludes. The Existing Sources step allows you to specify the folders containing the source packages and JUnit test packages. Source Package Folders and Test Package Folders sections are available.

- in the Choose Project step, you set Java as the category, and Java Project with Existing Sources as the project
- in the Name and Location step, you name the project
- in the Existing Sources step, you add the folder C:\labs\les05, and
- in the Project Properties window you set the Source/Binary Format property to JDK 7

The Projects window should show four Java source files beneath the Source Packages node. Then you follow these steps to create a new Java class named Customer:

Graphic

The project, Practice05, is open and includes the following Source Packages: CustomerTest.java, OrderTest.java, PersonTest.java, and TemperatureTest.java.

1. click **File - New File**
   The File menu is open and includes options such as New Project, New File, Open Project, Open Recent Project, Close Project, Open File, Open Recent File, Project Group, Project Properties, and Import Project.

2. select **Java Class** as the file type in the Choose File Type step, click **Next**, and
   The New File wizard is open on the Choose File Type page. There are two panes: Categories and File Type.

3. in the Name and Location step, name the class `Customer`
   The New Java Class wizard is open on the second step, Name and Location. The Class Name text box displays Customer, the Project text box displays Project05, the Location drop-down list is set to Source Packages, the Package drop-down list box is blank, and the Created File text box contains C:\labs\les05\Customer.java.

With Customer.java open for editing in the Editor pane, you declare and initialize the fields using this code.

The syntax of a variable declaration and initialization remains the same. You can assume that all fields are `public` and include a comment at the end of each line describing the field.

Graphic

The variables declarations are:

public int customerID = 0; // Default ID for the customer
public char status = 'N'; // default
public double totalPurchases = 0.0; // default

Code

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 *
 * @author Administrator
 */
public class Customer {

    public int customerID = 0; // Default ID for the customer
    public char status = 'N'; // default
    public double totalPurchases = 0.0; // default

// This method displays the values for an item
    public void displayCustomerInfo() {

        System.out.println("Customer ID: " + customerID);
```

```
        System.out.println("Status: " + status);
        System.out.println("Purchases: " + totalPurchases);
    } // end of display method
}
```

You now add a method within the `Customer` class called `displayCustomerInfo`. This method uses the `System.out.println` method to print each field to the screen with a corresponding label, such as "Purchases: ".

Graphic

The code for the method is:

```
public void displayCustomerInfo() {
  System.out.println("Customer ID: " + customerID);
  System.out.println("Status: " + status);
  System.out.println("Purchases: " + totalPurchases);
  }
```

Code

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 *
 * @author Administrator
 */
public class Customer {

    public int customerID = 0; // Default ID for the customer
    public char status = 'N'; // default
    public double totalPurchases = 0.0; // default

// This method displays the values for an item
    public void displayCustomerInfo() {

        System.out.println("Customer ID: " + customerID);
        System.out.println("Status: " + status);
        System.out.println("Purchases: " + totalPurchases);
    } // end of display method
}
```

Lastly you save to compile the class. You will notice that the red error indicator next to the `CustomerTest` class in the Projects window disappears after saving the `Customer` class. The reason is that the `CustomerTest` class references the `displayCustomerInfo` method, which did not exist before you saved the file. NetBeans recognized a potential compilation error in the `CustomerTest` class, due to the missing method.

You now run `CustomerTest` class to test your code. If you are prompted with a warning indicating that there are compilation errors within the project, click **Run Anyway**.

2. Declaring variables

Many of the values in Java technology programs are stored as primitive data types. There are eight primitive types built in to the Java programming language.

Graphic

The eight primitive data types are listed: byte, short, int, long, float, double, boolean and char.

There are four integral primitive types in the Java programming language, that store numbers that do not have decimal points. They are identified by the keywords

`byte`
: The `byte` type has a length of 8 bits and a range of -2 to the power of 7 to 2 to the power of 7, minus 1 – or -128 to 127 – which is a range of 256 possible values. If you need to store people's ages, for example, a variable of type `byte` would work because `byte` types can accept values in that range.
Examples of literal values using the byte type are 2, -114, and 0b10.

`short`
: The `short` type has a length of 16 bits and a range of -2 to the power of 15 to 2 to the power of 15, minus 1.
Examples of literal values using the short type are 2 and -32699.

`int`, and
: The `int` type is the default type for integral literals. It has a length of 32 bits and a range of -2 to the power of 31 to 2 to the power of 31, minus 1. Integer literals are assumed by the compiler to be of type `int` unless you specify otherwise using an `L` to indicate long type.
Examples of literal values are 2, 147334778, and 123_456_678.

`long`
: The `long` type has a length of 64 bits and a range of -2 to the power of 63 to 2 to the power of 63, minus 1. When you specify a literal value for a `long` type, you put a upper case `L` to the right of the value to explicitly state that it is a `long` type.
Examples of literal values are 2, -2036854775808L, and 1L.

Supplement

Selecting the link title opens the resource in a new browser window.

Job aid

Access the job aid, primitive data types, to learn more about the four integral primitive types in the Java programming language.

A new SE 7 feature allows you to express any of the integral types as a binary, consisting of 0s and 1s. For instance, a binary expression of the number 2 is shown as an allowed value of the `byte` integral type. The binary value is `0b10`. This value starts with 0b – either a lowercase or uppercase letter B. This indicates to the compiler that a binary value follows.

Another new feature of SE 7 is the ability to include underscores in a lengthy `int` number that helps with readability of the code. For example, you might use this to make it easier to read a large integral number, substituting underscores for commas. The use of the underscore has no effect on the numerical value of the `int`, nor does it appear if the variable is printed to the screen.

The `Shirt` class contains two attributes of type `int` to hold the values for a `shirtID` and the quantity in stock, and literal values are used to supply a default starting value of zero for each.

Code

```
public int shirtID = 0; // Default ID for the shirt
public int quantityInStock = 0; // Default quantity for all shirts
```

Note

The only reason to use the `byte` and `short` types in programs is to save memory consumption. Because most modern desktop computers contain an abundance of memory, most desktop application programmers do not use `byte` and `short` types.

There are two types for floating point numbers, `float` and `double`. These types are used to store numbers with values to the right of the decimal point, such as 12.24 or 3.14159.

The `float` type has a length of `32` bits. When you specify a literal value for a `float` type, put a upper case `F` to the right of the value to explicitly state that it is a `float` type, not a `double` type.

The `double` type has a length of 64 bits and is the default type for floating point literals. You use the `double` type when a greater range or higher accuracy is needed.

Supplement

Selecting the link title opens the resource in a new browser window.

Job aid

Access the job aid, floating point number types, to learn more about the two types for floating point numbers.

Literal values for floating point types are assumed to be of type `double` unless you specify otherwise, using the upper case `F` indicating `float` type. The `Shirt` class shows the use of one `double` literal value to specify the default value for the price.

Code

```
public double price = 0.0; // Default price for all shirts
```

Another data type you use for storing and manipulating data is single-character information. The primitive type used for storing a single character, such as a `y`, is `char` which is 16 bits in size. The `Shirt` class shows the use of one textual literal value to specify the default value for a `colorCode`.

When you assign a literal value to a `char` variable, such as `t`, you must use single quotation marks around the character. Using single quotation marks around the character clarifies for the compiler that the t is just the literal value t, rather than a variable `t` that represents another value.

Code

```
public char colorCode = 'U';
```

The `char` type does not store the actual character you type. The `char` representation is reduced to a series of bits that corresponds to a character. The number character mappings are set up in the character set that the programming language uses.

Many computer languages use American Standard Code for Information Interchange, or ASCII, which is an 8-bit character set that has an entry for every English character, punctuation mark, number, and so on.

The Java programming language uses a 16-bit character set called Unicode that can store all the necessary displayable characters from the vast majority of languages used in the modern world. Therefore, your programs can be written so that they work correctly and display the correct language for most countries. Unicode contains a subset of ASCII, that is the first 128 characters.

Question

Which primitive data type can accept a 16 bit value with a range of -2 to the power of 15 to 2 to the power of 15, minus 1?

Options:

1. `long`
2. `int`
3. `short`
4. `byte`

Answer

Option 1: Incorrect. The `long` data type has a length of 64 bits and a range of -2 to the power of 63 to 2 to the power of 63, minus 1.

Option 2: Incorrect. The `int` data type has a length of 32 bits and a range of range of -2 to the power of 31 to 2 to the power of 31, minus 1.

Option 3: Correct. The `short` data type has a length of 16 bits and a range of - range of -2 to the power of 15 to 2 to the power of 15 -1. Short is one of the four integral primitive types in the Java programming language, which also includes `byte`, `short`, `int`, and `long`.

Option 4: Incorrect. The `byte` data type has a length of 8 bits and a range of -128 to 127, or 256 possible values.

Correct answer(s):

3. `short`

Computer programs must often make decisions. The result of a decision, whether the statement in the program is true or false, can be saved in Boolean variables. Variables of type `boolean` can store only the following:

- the Java programming language literals `true` or `false`, and
- the results of an expression that only evaluates to `true` or `false` – for example, if the variable answer is equal to 42, this expression evaluates to a `false` result

  Code
  if answer < 42

Some rules you must adhere to when naming a variable are that

- variable identifiers must start with either an uppercase or lowercase letter, an underscore, or a dollar sign
- variable identifiers cannot contain punctuation, spaces, or dashes, and
- Java technology keywords cannot be used

As with a class or method, you must assign an identifier or a name to each variable in your program. Remember, the purpose of the variable is to act as a mechanism for storing and retrieving values. Therefore, you should make variable identifiers simple but descriptive.

For example, if you store the value of an item ID, you might name the variable `myID`, `itemID`, `itemNumber`, or anything else that clarifies the use of the variable to yourself and to others reading your program.

Many programmers follow the convention of using the first letter of the type as the identifier, for example `int i` and `float f`. This convention is acceptable for small programs that are easy to decipher, but generally you should use more descriptive identifiers.

Some guidelines in naming a variable include

Graphic

In this example, my is written in lowercase, and in the word Variable, V in uppercase and the rest of the word is written in lowercase.

- beginning each variable with a lowercase letter, and capitalizing subsequent words , for example, `myVariable` with an upppercae V, and
- choosing names that are mnemonic and that indicate to the casual observer the intent of the variable

You can assign a value to a variable when the variable is declared, or you can assign the value later. To assign a value to a variable during declaration, add an equal sign after the declaration, followed by the value to be assigned. For example, the price field in the `Shirt` class could be assigned the value of `12.99` as the price for a `Shirt` object.

Code

```
double price = 12.99;
```

The = operator assigns the value on the right side to the item on the left side. The = operator should be read as "is assigned to." In this example, you could say, "`false` is assigned to `isOpen`."

Code

```
boolean isOpen = false;
```

Note

Fields are automatically initialized. Integral types are set to `0`, floating point types are set to `0.0`, the `char` type is set to `\u0000`, and the `boolean` type is set to `false`. You should explicitly initialize your fields so that other people can read your code. Local variables – those declared within a method – must be explicitly initialized before being used.

You can declare one or more variables on the same line of code, but only if they are all of the same type. You use this syntax for declaring several variables in one line of code.

Therefore, if there are separate retail and wholesale prices in the `Shirt` class, you can declare the variables like this.

Code

```
double price = 0.0, wholesalePrice = 0.0;
```

Syntax

type identifier = *value* [, *identifier* = *value*];

You can assign values to variables using several different approaches:

- assigning literal values directly to variables, and

  Code
  int ID = 0;
  float pi = 3.14F;
  char myChar = 'G';
  boolean isOpen = false;

- assigning the value of one variable to another variable

  Code
  int ID = 0;
  int saleID = ID;

The first line of code creates an integer called `ID` and uses it to store the number 0. The second line of code creates another integer called `saleID` and uses it to store the same value as `ID`, which is 0. If the contents of `ID` are changed later, the contents of `saleID` do not automatically change. Even though the two integers currently have the same value, they can be independently changed later in a program.

Code

```
int ID = 0;
int saleID = ID;
```

You can also assign the result of an expression to integral, floating point, or Boolean type variables. In these examples, the result of everything on the right side of the `=` operator is assigned to the variable on the left side of the `=` operator.

You can also assign the return value of a method call to a variable.

Graphic

The relevant code is

float casePrice = 19.99F;
and
int hour = 12;

Code

```
float numberOrdered = 908.5F;
float casePrice = 19.99F;
float price = (casePrice * numberOrdered);
int hour = 12;
boolean isOpen = (hour > 8);
```

In addition to using variables that store values that you can change, you can also use constants to represent values that cannot change.

Assume that you are writing part of a scheduling application, and you need to refer to the number of months in a year. You make the variable a constant by using the `final` keyword to inform the compiler that you do not want the value of the variable to be changed after it has been initialized. Also, by convention, name the constant identifier using all capital letters, with underscores separating words, so that it is easy to determine that it is a constant.

Code

```
final int NUMBER_OF_MONTHS = 12;
```

Any values that tend to change rarely, if ever, are good candidates for a constant variable, such as `MAX_COUNT` or `PI`. If someone attempts to change the value of a constant after it has already been assigned a value, the compiler gives an error message. If you modify your code to provide a different value for the constant, you need to recompile your program.

You should name constants so that they can be easily identified. Generally, constants should be capitalized, with words separated by an underscore.

When you use a literal value or create a variable or constant and assign it a value, the value is stored in the memory of the computer.

Local variables are stored separately on the stack, whereas fields are stored on the heap. Objects and their fields and methods are usually stored in heap memory. Heap memory consists of dynamically allocated memory chunks containing information used to hold objects, including their fields and methods, while they are needed by your program. Other variables are usually stored in stack memory. Stack memory stores items that are used for only a brief period of time, such as variables declared inside of a method.

Summary

In this topic you've learned to identify the use and syntax of variables, and you've also learned how to declare variables.

Table of Contents