| [Print](javascript:window.print()) | [Close](javascript:parent.window.close()) |

**Course Transcript**

Java SE7 Fundamentals: Variables, Operators, and Objects

**Variables, Operators, and Type Casting in Java**

| [1. Declaring, Initializing, and Using Variables](http://xlibrary.skillport.com/courseware/Content/cca/jl_jsef_a02_it_enus/output/html/course_transcript.html#t19) |

| [2. Using Operators, Promotion, and Type Casting](http://xlibrary.skillport.com/courseware/Content/cca/jl_jsef_a02_it_enus/output/html/course_transcript.html#t3) |

| [3. Using Variables, Operators, and Type Casting](http://xlibrary.skillport.com/courseware/Content/cca/jl_jsef_a02_it_enus/output/html/course_transcript.html#t16) |

**Working with Objects in Java**

| [1. Working with Objects](http://xlibrary.skillport.com/courseware/Content/cca/jl_jsef_a02_it_enus/output/html/course_transcript.html#t6) |

| [2. Using the String and StringBuilder Classes](http://xlibrary.skillport.com/courseware/Content/cca/jl_jsef_a02_it_enus/output/html/course_transcript.html#t8) |

| [3. Using the Java API Documentation](http://xlibrary.skillport.com/courseware/Content/cca/jl_jsef_a02_it_enus/output/html/course_transcript.html#t10) |

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.*

* 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.*
* 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.*
* 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,](javascript:doWindow('./ja_jl_jsef_a02_it_enusT1901_frame.html')) 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](javascript:doWindow('./ja_jl_jsef_a02_it_enusT1902_frame.html')), 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.

**3. Summary**

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

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/jl_jsef_a02_it_enus/output/html/course_transcript.html#top)

Using Operators, Promotion, and Type Casting

Learning Objectives

*After completing this topic, you should be able to*

* *describe how to modify variables using operators*
* *recognize how to use promotion and type casting*

**1. Modifying variables using operators**

Programs do a lot of mathematical calculating, from the simple to the complex. Arithmetic  
operators let you specify how the numerical values within variables should be evaluated or  
combined. The standard mathematical operators, often called binary operators, used in the Java programming language are shown in the tables. These are the most common arithmetic operators.

**+**

The + operator is used for addition. For example, if num1 is 10 and num2 is 2, the sum is 12.

**-**

The - operator is used for subtraction. For example, if num1 is 10 and num2 is 2, the result is 8.

**\***

The \* operator is used for multiplication. For example, if num1 is 10 and num2 is 2, the product is 20.

**/**

The / operator is used for division. For example, if num1 is 10 and num2 is 2, then the result is 5. A comment states that division returns an integer value with no remainder

**%**

The % operator is used for finding the remainder. The % symbol is known as a modulus. For example, if num1 is 31 and num2 is 6, the result of 31/6 is 5 with a remainder of 1. A comment states that Remainder finds the remainder of the first number divided by the second number, and remainder always gives an answer with the same sign as the first operand.

Supplement

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

**Job aid**

Access the job aid,[arithmetic operators](javascript:doWindow('./ja_jl_jsef_a02_it_enusT301_frame.html')), to learn more about the most common arithmetic operators in Java.

A common requirement in programs is to add or subtract 1 from the value of a variable. You can  
do this using the + or - operators.

Code

age = age + 1;  
  
count = count – 1;

Incrementing or decrementing by 1 is such a common action that there are specific unary operators for it. These are the increment and decrement operators. These operators can come before a variable, called pre-increment and pre-decrement, or after a variable, called post-increment and post-decrement.

**Pre-increment**

The pre-increment operator increments the value before assigning it. For example, i is first incremented before its value is assigned to j. So i = 7 and j = 7 in this case.

**Post-increment**

The post-increment operator increments the value after assigning it. For example, int i is assigned to j and then incremented. So i = 7 and j = 6 in this case.

**Pre-decrement**

The pre-decrement operator decrements the value before assigning it. For example, i is first decremented before its value is assigned to j. So i = 5 and j = 5 in this case.

**Post-decrement.**

The post-decrement operator decrements the value after assigning it. For example, int i is assigned to j and then decremented. So i = 5 and j = 6 in this case.

The code, in which the age was incremented by 1, can also be written like this.  
  
You should use these operators within an expression cautiously. With the prefix form, the operation – increment or decrement – is applied before any subsequent calculations or assignments. With the postfix form, the operation is applied after the subsequent calculations or operations, so that the original value – and not the updated value – is used in subsequent calculations or assignments.

Code

age++;  
  
++age;

This example shows the basic use of the increment and decrement operators.  
  
The result of this code fragment is 15, 16, 17, 17.

Code

int count=15;  
int a, b, c, d;  
a = count++;  
b = count;  
c = ++count;  
d = count;  
System.out.println("Result = " + a + ", " + b + ", " + c + ", " + d);

In a complex mathematical statement with multiple operators on the same line, the computer needs to pick which operator it should use first. To make mathematical operations consistent, the Java programming language follows the standard mathematical rules for operator precedence.

Operators are processed in the following order:

* operators within a pair of parentheses   
  *A pair or round brackets, or parenthesis, appear on screen.*
* increment and decrement operators   
  *The increment, or double pluses and decrement, or double minuses, operators appears on screen.*
* multiplication and division operators, evaluated from left to right, and   
  *The multiplication, or asterisk character, and division, or forward-slash character, appears on screen.*
* addition and subtraction operators, evaluated from left to right   
  *The addition, or plus character and subtraction, or minus character, appears on screen.*

If standard mathematical operators of the same precedence appear successively in a statement,  
the operators are evaluated from left to right.  
  
This example demonstrates the need for operator precedence.

Code

c = 25 - 5 \* 4 / 2 - 10 + 4;  
  
c = 25 - ((5 \* 4) / 2)) - 10 + 4;

**c = 25 - 5 \* 4 / 2 - 10 + 4;**

*In the first example it's not clear what the intended result is of this equation. If evaluated strictly from left to right, the answer is 34.*

**c = 25 - ((5 \* 4) / 2)) - 10 + 4;**

*If the equation is evaluated according to rules of precedence, as indicated by the parentheses, the actual expression result is 9.*

Your expression will be automatically evaluated with the rules of precedence. However, you  
should use parentheses to provide the structure you intend.

Code

c = (((25 - 5) \* 4) / (2 - 10)) + 4;  
c = ((20 \* 4) / (2 - 10)) + 4;  
c = (80 / (2 - 10)) + 4;  
c = (80 / -8) + 4;  
c = -10 + 4;  
c = -6;

Question

Match the operator to its purpose.

**Options:**

1. %
2. ++
3. \*
4. --

**Targets:**

1. Remainder
2. Increment
3. Multiplication
4. Decrement

Answer

*To determine the remainder, the % operator is used. It finds the remainder of the first number divided by the second number.*

*The ++ operator is used to increment a value by one. This operator can come before or after a variable.*

*The \* operator is used to multiple one value by another.*

*The -- operator is used to decrement a value by one. This operator can come before or after a variable.*

**Correct answer(s):**

Target 1 = Option A

Target 2 = Option B

Target 3 = Option C

Target 4 = Option D

**2. Using promotion and type casting**

Assigning a variable or an expression to another variable can lead to a mismatch between the data types of the calculation and the storage location that you are using to save the result. Specifically, the compiler will either recognize that precision will be lost and not allow you to compile the program, or the result will be incorrect. To fix this problem, variable types have to be either promoted to a larger size type, or type cast to a smaller size type.

Code

int num1 = 53; // 32 bits of memory to hold the value  
int num2 = 47; // 32 bits of memory to hold the value  
byte num3; // 8 bits of memory reserved  
num3 = (num1 + num2); // causes compiler error

The code should work, because a byte, though smaller than an int, is large enough to store a value of 100.   
  
However, the compiler will not make this assignment and, instead, issues a "possible loss of precision" error because a byte value is smaller than an int value.

Code

int num1 = 53; // 32 bits of memory to hold the value  
int num2 = 47; // 32 bits of memory to hold the value  
byte num3; // 8 bits of memory reserved  
num3 = (num1 + num2); // causes compiler error

To fix this problem, you can either type cast the right-side data type down to match the left-side data type, or declare the variable on the left side (num3) to be a larger data type, such as an int.  
  
In this case, the problem is fixed by changing num3 to an int.

Graphic

*The relevant code is  
  
int num3;*

Code

int num1 = 53;  
int num2 = 47;  
int num3;  
num3 = (num1 + num2);

In some circumstances, the compiler changes the type of a variable to a type that supports a larger size value. This action is referred to as a promotion. Some promotions are done automatically by the compiler if data would not be lost by doing so. These promotions are done under the following conditions:

* if you assign a smaller type, displayed on the right of the = sign, to a larger type, displayed on the left of the = sign, and
* if you assign an integral type to a floating point type, because there are no decimal places to lose

This example contains a literal – an int – that will automatically be promoted to another type – a long – before the value 6 is assigned to the variable, which is big of type long.  
  
Because 6 is an int type, promotion works because the int value is converted to a long value.

Code

long big = 6;

Prior to being assigned to a variable, the result of an equation is placed in a temporary location in memory. The location's size is always equal to the size of an int type or the size of the largest data type used in the expression or statement.   
  
For example, if your equation multiplies two int types the container size will be an int type in size, or 32 bits.

If the two values that you multiply yield a value that is beyond the scope of an int type, the int value must be truncated to fit the result into the temporary location in memory.  
  
This calculation ultimately yields an incorrect answer because the variable for your answer receives a truncated value, regardless of the type used for your answer.  
  
To solve this problem, you should set at least one of the variables in your equation to the long type to ensure the largest possible temporary container size.

Graphic

*The example shown is:  
55555\*66666=3,703,629,630,*

Type casting lowers the range of a value, quite literally chopping it down to a smaller size, by changing the type of the value – for example, converting a long value to an int value. You do this so that you can use methods that accept only certain types as arguments, so that you can assign values to a variable of a smaller data type, or to save memory. You put the target\_type, which is the type that the value is being type cast to, in parentheses in front of the item that you are type casting.

Syntax

*identifier*= (*target\_type*) *value*

This is the syntax for type casting a value.

Syntax

*identifier*= (*target\_type*) *value*

*The identifier is the name you assign to the variable.*

*The (target\_type) is the type to which you want to type cast the value. The target\_type must be in parentheses.*

*The value is the value you want to assign to the identifier.*

In this example, the final line of code will cause a compiler error because byte of the result variable is smaller than the sum of two int data types.

Code

int num1 = 53; // 32 bits of memory to hold the value  
int num2 = 47; // 32 bits of memory to hold the value  
byte num3; // 8 bits of memory reserved  
num3 = (num1 + num2); // causes compiler error

In this case the compiler error is fixed by type casting the result to a byte. You should use type casting with caution. For example, if larger numbers for num1 and num2 were used, the type cast to a byte would truncate part of the data, resulting in an incorrect answer.

Code

int num1 = 53; // 32 bits of memory to hold the value  
int num2 = 47; // 32 bits of memory to hold the value  
byte num3; // 8 bits of memory reserved  
num3 = (byte)(num1 + num2); // no data loss

This example presents another problem.  
  
If you type cast a float or double value with a fractional part to an integral type such as an int, all decimal values are lost. However, this method of type casting is sometimes useful if you want to truncate the number down to the whole number, for example 51.9 becomes 51.

Code

int myInt;  
long myLong = 99L;  
myInt = (int) (myLong); // No data loss, only zeroes.  
                        // A much larger number would  
                        // result in data loss.  
int myInt;  
long myLong = 123987654321L;  
myInt = (int) (myLong); // Number is "chopped"

The Java technology compiler makes certain assumptions when it evaluates expressions. You must understand these assumptions to make the appropriate type casts or other accommodations.  
  
Most operations result in int or long, as in these situations:

* byte, char, and short values are promoted to int before the operation
* if either argument is of the long type, the other is also promoted to long, and the result is of the long type, and   
    
  **Code**  
  byte b1 = 1, b2 = 2, b3;  
  b3 = b1 + b2; // Error: result is an int but b3 is a byte
* if an expression contains a float, the entire expression is promoted to float, and literal  
  floating-point values are viewed as double   
    
  **Code**  
  float float1 = (float) 27.9;

In this example, an error occurs because two of the three operands – a and b – are automatically promoted from a short type to an int type before they are added.  
  
In the last line, the values of a and b are converted to int types and the converted values are added to give an int result. Then the assignment operator, =, attempts to assign the int result to the short variable, c. However, this assignment is illegal and causes a compiler error.

Code

short a, b, c;  
a = 1 ;  
b = 2 ;  
c = a + b ; //compiler error

The code works if you do either of the following:

Code

short a, b, c;  
a = 1 ;  
b = 2 ;  
c = a + b; //compiler error

* declare c as an int in the original declaration, or   
  *You use the code:  
    
  int c;*   
    
  **Code**  
  short a, b;  
  int c;  
  a = 1 ;  
  b = 2 ;  
  c = a + b;
* type cast the (a+b) result in the assignment line   
  *You use the code:  
    
  c = (short)(a+b);*   
    
  **Code**  
  short a, b, c;  
  a = 1 ;  
  b = 2 ;  
  c = (short)(a+b);

Just as integral types default to int under some circumstances, values assigned to floating point  
types always default to a double type, unless you specifically state that the value is a float type.  
  
For example, this line of code causes a compiler error. Because 27.9 is assumed to be a double type, a compiler error occurs because a double type value cannot fit into a float variable.

Code

float float1 = 27.9;//compiler error

However, both of the following lines of code will work correctly:

* the F notifies the compiler that 27.9 is a float value, and   
    
  **Code**  
  float float1 = 27.9F;
* 27.9 is cast to a float type   
    
  **Code**  
  float float1 = (float) 27.9;

This code example uses principles that have been covered to calculate a person's age in days and seconds.

Code

public class Person {  
    public int ageYears = 32;  
  
    public void calculateAge() {  
  
        int ageDays = ageYears \* 365;  
        long ageSeconds = ageYears \* 365 \* 24L \* 60 \* 60;  
  
        System.out.println("You are " + ageDays + " days old.");  
        System.out.println("You are " + ageSeconds + " seconds old.");  
    } // end of calculateAge method  
} // end of class

Question

Which statements accurately describe promotion?

**Options:**

1. It lowers the range of a value
2. It should be used with caution
3. It changes the type of variable to one that supports a larger size
4. It can be handled automatically by the compiler

Answer

***Option 1:****Incorrect. Type casting lowers the range of a value, quite literally chopping it down to a smaller size, by changing the type of the value.*

***Option 2:****Incorrect. You should use type casting with caution. For example, consider if num1 is 53, num2 is 47, both are type int, and you want to type cast them to byte. Larger numbers being used for num1 and num2 would cause the type cast to a byte to truncate part of the data, resulting in an incorrect answer.*

***Option 3:****Correct. In some circumstances, the compiler changes the type of a variable to a type that supports a larger size value. This action is referred to as a promotion.*

***Option 4:****Correct. Some promotions are done automatically by the compiler if data would not be lost by doing so.*

**Correct answer(s):**

3. It changes the type of variable to one that supports a larger size  
4. It can be handled automatically by the compiler

**3. Using operators and type casting**

You want to use operators to calculate age in days, minutes, seconds, and milliseconds. Follow these steps in the NetBeans IDE:

Graphic

*The NetBeans interface is open and the loaded project has a number of source packages, which include Customer.java, CustomerTest.java, OrderTest.java, and Person.java. Person.java is open in the code editor and includes the class and method declarations.*

Code

public class Person {  
  
    public int ageYears = 1;  
    public int ageDays;  
    public long ageMinutes, ageSeconds, ageMilliseconds;  
  
    public void calculateAge() {  
        ageDays = ageYears \* 365;  
        ageSeconds = ageDays \* 24 \* 60 \* 60;  
        ageMinutes = ageSeconds / 60;  
        ageMilliseconds = ageSeconds \* 1000;  
          
        System.out.println("You are " + ageDays + " days old.");  
        System.out.println("You are " + ageSeconds + " seconds old.");  
        System.out.println("You are " + ageMinutes + " minutes old.");  
        System.out.println("You are " + ageMilliseconds + " milliseconds old.");  
    } // end of calculateAge method  
}

* Select **File - New File** from the menu, to create a new Java class called Person.   
  *The Person.java file is open.  
    
  The Java class declaration is:  
    
  public class Person {*   
    
  **Code**  
  public class Person {  
    
      public int ageYears = 1;  
      public int ageDays;  
      public long ageMinutes, ageSeconds, ageMilliseconds;  
    
      public void calculateAge() {  
          ageDays = ageYears \* 365;  
          ageSeconds = ageDays \* 24 \* 60 \* 60;  
          ageMinutes = ageSeconds / 60;  
          ageMilliseconds = ageSeconds \* 1000;  
            
          System.out.println("You are " + ageDays + " days old.");  
          System.out.println("You are " + ageSeconds + " seconds old.");  
          System.out.println("You are " + ageMinutes + " minutes old.");  
          System.out.println("You are " + ageMilliseconds + " milliseconds old.");  
      } // end of calculateAge method  
  }
* Using the editor, you add these fields to store age in years, days, minutes, seconds, and milliseconds. It's important to provide meaningful names for all the fields, for example, ageYears.   
  *You declare the variable using this code:  
    
  public int ageYears = 1;  
  public int ageDays;  
  public long ageMinutes, ageSeconds, ageMilliseconds;*   
    
  **Code**  
  public class Person {  
    
      public int ageYears = 1;  
      public int ageDays;  
      public long ageMinutes, ageSeconds, ageMilliseconds;  
    
      public void calculateAge() {  
          ageDays = ageYears \* 365;  
          ageSeconds = ageDays \* 24 \* 60 \* 60;  
          ageMinutes = ageSeconds / 60;  
          ageMilliseconds = ageSeconds \* 1000;  
            
          System.out.println("You are " + ageDays + " days old.");  
          System.out.println("You are " + ageSeconds + " seconds old.");  
          System.out.println("You are " + ageMinutes + " minutes old.");  
          System.out.println("You are " + ageMilliseconds + " milliseconds old.");  
      } // end of calculateAge method  
  }

You also need to create a new public method in this class called calculateAge. This method should

Graphic

*The method is declared with:  
  
public void calculateAge() {*

Code

public class Person {  
  
    public int ageYears = 1;  
    public int ageDays;  
    public long ageMinutes, ageSeconds, ageMilliseconds;  
  
    public void calculateAge() {  
        ageDays = ageYears \* 365;  
        ageSeconds = ageDays \* 24 \* 60 \* 60;  
        ageMinutes = ageSeconds / 60;  
        ageMilliseconds = ageSeconds \* 1000;  
          
        System.out.println("You are " + ageDays + " days old.");  
        System.out.println("You are " + ageSeconds + " seconds old.");  
        System.out.println("You are " + ageMinutes + " minutes old.");  
        System.out.println("You are " + ageMilliseconds + " milliseconds old.");  
    } // end of calculateAge method  
}

* calculate age in days, minutes, seconds, and milliseconds, assigning the value to the relevant field, and   
  *The code for these calculations is:  
    
  ageDays = ageYears \* 365;  
  ageSeconds = ageDays \* 24 \* 60 \* 60;  
  ageMinutes = ageSeconds / 60;  
  ageMilliseconds = ageSeconds \* 1000;*   
    
  **Code**  
  public class Person {  
    
      public int ageYears = 1;  
      public int ageDays;  
      public long ageMinutes, ageSeconds, ageMilliseconds;  
    
      public void calculateAge() {  
          ageDays = ageYears \* 365;  
          ageSeconds = ageDays \* 24 \* 60 \* 60;  
          ageMinutes = ageSeconds / 60;  
          ageMilliseconds = ageSeconds \* 1000;  
            
          System.out.println("You are " + ageDays + " days old.");  
          System.out.println("You are " + ageSeconds + " seconds old.");  
          System.out.println("You are " + ageMinutes + " minutes old.");  
          System.out.println("You are " + ageMilliseconds + " milliseconds old.");  
      } // end of calculateAge method  
  }
* print out all the ages in various units, each in a separate line with an appropriate message – for example, "You are 3156000 seconds old"   
  *The code to print the various age units is:  
    
  System.out.println("You are " + ageDays + " days old.");  
  System.out.println("You are " + ageSeconds + " seconds old.");  
  System.out.println("You are " + ageMinutes + " minutes old.");  
  System.out.println("You are " + ageMilliseconds + " milliseconds old.");*   
    
  **Code**  
  public class Person {  
    
      public int ageYears = 1;  
      public int ageDays;  
      public long ageMinutes, ageSeconds, ageMilliseconds;  
    
      public void calculateAge() {  
          ageDays = ageYears \* 365;  
          ageSeconds = ageDays \* 24 \* 60 \* 60;  
          ageMinutes = ageSeconds / 60;  
          ageMilliseconds = ageSeconds \* 1000;  
            
          System.out.println("You are " + ageDays + " days old.");  
          System.out.println("You are " + ageSeconds + " seconds old.");  
          System.out.println("You are " + ageMinutes + " minutes old.");  
          System.out.println("You are " + ageMilliseconds + " milliseconds old.");  
      } // end of calculateAge method  
  }

Save the file to compile the class and then run the PersonTest.java file which has already been created. You can perform several tests, by setting the value of age as 1, 24, and 80 in the Person class.  
  
For example, for one year, the results should be: You are 365 days old. You are 31,536,000 seconds old. You are 525,600 minutes old. You are 31,536,000,000 milliseconds old.

Graphic

*The Person.java file is open in NetBeans.*

Code

public class Person {  
  
    public int ageYears = 1;  
    public int ageDays;  
    public long ageMinutes, ageSeconds, ageMilliseconds;  
  
    public void calculateAge() {  
        ageDays = ageYears \* 365;  
        ageSeconds = ageDays \* 24 \* 60 \* 60;  
        ageMinutes = ageSeconds / 60;  
        ageMilliseconds = ageSeconds \* 1000;  
          
        System.out.println("You are " + ageDays + " days old.");  
        System.out.println("You are " + ageSeconds + " seconds old.");  
        System.out.println("You are " + ageMinutes + " minutes old.");  
        System.out.println("You are " + ageMilliseconds + " milliseconds old.");  
    } // end of calculateAge method  
}

You now need to use casting to ensure that data loss does not occur in your programs:

Code

public class Order {  
    public long orderValue = 0L;  
    public int itemQuantity = 10\_000\_000;  
    public int itemPrice = 555\_500;  
      
    public void calculateTotal(){  
        orderValue = (itemQuantity) \* itemPrice;  
        System.out.println("Order total: "+ orderValue);  
   }  
}

* Create a new Java class called Order and add three fields to the Order class. The underscores used to initialize the int values improve the readability of your code. They have no effect on the actual numeric value of the field. The compiler strips them out. This is one of the new language features of Java 7.   
  *The Order class and three fields' declaration is:  
    
  public class Order {  
     public long orderValue = 0L;  
     public int itemQuantity = 10\_000\_000;  
     public int itemPrice = 555\_500;*   
    
  **Code**  
  public class Order {  
      public long orderValue = 0L;  
      public int itemQuantity = 10\_000\_000;  
      public int itemPrice = 555\_500;  
    
      public void calculateTotal(){  
          orderValue = (long)(itemQuantity) \* (long)itemPrice;  
          System.out.println("Order total: "+ orderValue);  
     }  
  }
* Create a calculateTotal method that will calculate the total order value (quantity \* price) and print it. Type cast both itemQuantity and itemPrice to a long so that the int values will not be truncated. Both of these values are too large for an int type.   
  *The code for the calculateTotal method is:  
    
  public void calculateTotal(){  
         orderValue = (long)(itemQuantity) \* (long)itemPrice;  
         System.out.println("Order total: "+ orderValue);  
     }*   
    
  **Code**  
  public class Order {  
      public long orderValue = 0L;  
      public int itemQuantity = 10\_000\_000;  
      public int itemPrice = 555\_500;  
    
      public void calculateTotal(){  
          orderValue = (long)(itemQuantity) \* (long)itemPrice;  
          System.out.println("Order total: "+ orderValue);  
      }  
  }
* Save Order.java and test it by running OrderTest.java. Verify the result by using a calculator.   
  *The OrderTest.java file has been compiled and run. In the Output pane in the Project05 tab, the Order Total is shown as 5555000000000. Below this, a message reads: BUILD SUCCESSFUL (total time: 1 second).*   
    
  **Code**  
   orderValue = (long)(itemQuantity) \* (long)itemPrice;  
          System.out.println("Order total: "+ orderValue);  
      }  
  }
* Edit the Order.java file to remove the type casting done in the calculateTotal Method by removing (long).   
  *The line of code now reads:  
    
  orderValue = (itemQuantity) \* itemPrice;*   
    
  **Code**  
  public class Order {  
      public long orderValue = 0L;  
      public int itemQuantity = 10\_000\_000;  
      public int itemPrice = 555\_500;  
        
      public void calculateTotal(){  
          orderValue = (itemQuantity) \* itemPrice;  
          System.out.println("Order total: "+ orderValue);  
     }  
  }
* Compile and run OrderTest again. You will see the resulting data loss that occurs without type casting.   
  *The Order Total in the output pane now is 1607286272.*   
    
  **Code**  
      public int itemQuantity = 10\_000\_000;  
      public int itemPrice = 555\_500;  
        
      public void calculateTotal(){  
          orderValue = (itemQuantity) \* itemPrice;  
          System.out.println("Order total: "+ orderValue);  
      }  
  }

You now want to write a program to convert temperature from Fahrenheit to Celsius.

Graphic

*The Project05 project is open in the NetBeans IDE.*

Code

public class Temperature {  
    public float fahrenheitTemp = 78.9F;  
      
    public void calculateCelsius(){  
         System.out.println((fahrenheitTemp - 32) \* 5 / 9);  
    } //end calculateCelsius method  
} // end class

* First you create a new Java class called Temperature. You then add a member field to the Temperature class that stores the temperature in Fahrenheit. You declare the field variable with an appropriate data type, such as int, float, or double.   
  *The variable declaration is:  
    
  public float fahrenheitTemp = 78.9F;*   
    
  **Code**  
  public class Temperature {  
      public float fahrenheitTemp = 78.9F;  
        
      public void calculateCelsius(){  
           System.out.println((fahrenheitTemp - 32) \* 5 / 9);  
      } //end calculateCelsius method  
  } // end class
* Then create a calculateCelsius method. Convert the Fahrenheit temperature to Celsius by subtracting 32, multiplying by 5, and dividing by 9. Be sure to observe the rules of precedence when typing this expression.   
  *The code for the calculateCelcius method is:  
    
  public void calculateCelsius(){  
          System.out.println((fahrenheitTemp - 32) \* 5 / 9);  
     } //end calculateCelsius method*   
    
  **Code**  
  public class Temperature {  
      public float fahrenheitTemp = 78.9F;  
        
      public void calculateCelsius(){  
           System.out.println((fahrenheitTemp - 32) \* 5 / 9);  
      } //end calculateCelsius method  
  } // end class
* You then compile the Temperature class and test it using the TemperatureTest class. Confirm that you get the same result running the program as you do when doing this calculation using a calculator.   
  *The Output pane returns the Celsius value of 26.055555.*   
    
  **Code**  
  public void calculateCelsius(){  
           System.out.println((fahrenheitTemp - 32) \* 5 / 9);  
      } //end calculateCelsius method  
  } // end class
* Test the program using several values of temperature.
* When you have finished experimenting with different values, close the Practice05 project.

**4. Summary**

In this topic you've learned how to modify variables using operators. You've also recognized how and when to use promotion and type casting.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/jl_jsef_a02_it_enus/output/html/course_transcript.html#top)

Using Variables, Operators, and Type Casting

Learning Objectives

*After completing this topic, you should be able to*

* *use variables in a field definition*
* *use operators and prevent data loss using type casting*

**1. Exercise overview**

In this exercise, you're required to create Java classes that declare, initialize and manipulate field variables.

This involves the following tasks:

* using variables in a field definition, and
* using operators and performing type casting

**2. Using variables in a field definition**

You're writing a program that will record a stock list for a bookstore.

Question

You have created a class named Book to represent books in a bookstore. The class already has two fields for author and title. You want to add two more fields to the class to represent the price and quantity in stock of a book as integers.  
  
Which statements are valid?

**Options:**

1. public int price;  
   public int quantityInStock = 0;
2. public Char price;  
   public int quantityInStock = 0;
3. public int price, quantityInStock = 0;
4. public int price; quantityInStock = 0;

Answer

***Option 1:****Correct. These statements declare the required fields and initialize the quantityInStock value to zero.*

***Option 2:****Incorrect. The data type char is inappropriate since you need to use integers. Also note that char should all lowercase.*

***Option 3:****Correct. You can also declare variables of the same type on one line if separated by a comma.*

***Option 4:****Incorrect. You only declare variables of the same type on one line if separated by a comma, not a semi-colon.*

**Correct answer(s):**

1. public int price;  
public int quantityInStock = 0;  
3. public int price, quantityInStock = 0;

Question

You decide that to properly reflect prices, the price variable should use decimal places.  
  
Which data type is most suitable based on their range and accuracy?

**Options:**

1. byte
2. float
3. long
4. double

Answer

***Option 1:****Incorrect. The byte data type does not use decimals and is only eight bits in length.*

***Option 2:****Incorrect. While float allows the use of decimal places, you should use the double type when a greater range or higher accuracy is needed.*

***Option 3:****Incorrect. Long variables do not allow decimal places so this would not be suitable.*

***Option 4:****Correct. The double data type allows you to use decimal places and should be used when a greater range or higher accuracy is needed.*

**Correct answer(s):**

4. double

**3. Using operators and type casting**

You are developing a program to calculate the cost of installing flooring in a room.

Question

You have already developed a class to represent the types of flooring and need to include a statement to properly calculate the total quote, including tax based on the area of the room and the price.  
  
Which statement performs this operation correctly?

**Options:**

1. totalCost = ((length + width) \* price) + (1 + taxRate);
2. totalCost = price \* length \* width + 1 + taxRate;
3. totalCost = (length \* width) \* price ;
4. totalCost = ((length \* width) \* price) \* (1 + taxRate);

Answer

***Option 1:****Incorrect. You need to multiply the length of the room by the width to get the area.*

***Option 2:****Incorrect. If you do not include parenthesis, the desired order of operations is not followed.*

***Option 3:****Incorrect. To properly calculate the tax rate, you need to include the variable.*

***Option 4:****Correct. You include the parenthesis and all the required variables to ensure that the cost is calculated correctly.*

**Correct answer(s):**

4. totalCost = ((length \* width) \* price) \* (1 + taxRate);

Question

You want to track the value of the inventory for specific types of flooring. You are required to declare invValue as a long. You had previously declared quantityInStock as an int and price as double.  
  
Which code segment inserted in the designated location will complete this and has the least amount of precision loss?

**Code**  
    public double price;  
    public int quantityInStock = 0;  
    invValue = INSERT THE MISSING CODE

**Options:**

1. (long)((double)quantityInStock \* (double)price);
2. (long)quantityInStock \* (long)price;
3. (double)quantityInStock \* (double)price;
4. (long)quantityInStock \* (long)price

Answer

***Option 1:****Correct. Converting the variables to type double before evaluating the equation will avoid a mathematical loss of precision. However, a data type loss of precision can be avoided by casting the result of the equation to type long.*

***Option 2:****Incorrect. This code segment will correctly compile and execute without error. However, there is a possible loss of precision mathematically when the price variable of type double is cast to type long.*

***Option 3:****Incorrect. You are required to use invValue as a long. A variable of type double cannot be implicitly cast to type long. Therefore, a possible loss of precision compiler error will occur.*

***Option 4:****Incorrect. While it does use the correct long data type, this statement is missing the closing semi-colon.*

**Correct answer(s):**

1. (long)((double)quantityInStock \* (double)price);

Field variables have been used in a class definition and the most suitable data type has been evaluated. Operators have been used and their order of precedence has been evaluated. In addition, type casting has been performed.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/jl_jsef_a02_it_enus/output/html/course_transcript.html#top)

Working with Objects

Learning Objective

*After completing this topic, you should be able to*

* *recognize how to declare, instantiate, and initialize objects*

**1. Working with Java objects**

Objects are accessed via references, and are instantiated versions of their class. Objects consist of attributes and operations. In Java, these are fields and methods.

To work with an object, you need to access it via a reference. A good analogy is using a remote  
control to operate an electronic device.   
  
The buttons on the remote control can be used to modify the operation of the device, for example, a video camera. The camera is like the object that is accessed via the reference, or remote.  
  
You can use the remote to have the camera stop, play, or record by interacting with the remote. The remote is like the reference used to access the camera, or object.

You want to access a simple object based on the Shirt class. The class has four fields– shirtID, description, colorCode, and price – and one method, display().

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 all items  
    // This method displays the details for an item  
    public void display() {  
        System.out.println("Item ID: " + shirtID);  
        System.out.println("Item description:" +  
            description);  
        System.out.println("Color Code: " + colorCode);  
        System.out.println("Item price: " + price);  
    } // end of display method  
} // end of class

Note

*Methods are usually written in this fashion, with the method name followed by a pair of  
parentheses to indicate that it is a method.*

There are three steps to obtaining an object reference:

* Declare the reference.   
  **Syntax**  
  *Classname identifier*;
* Instantiate the object. This code fragment  
  creates the object.   
  **Syntax**  
  new *Classname*();
* Assign the object to the reference. The identifier is taken from the declaration step. To assign the object to a reference, creation and assignment must be in the same statement.   
  **Syntax**  
  *Object reference* = new *Classname*();

The way the assignment operator, an equals symbol, works requires that the reference and the newly created object must be in the same statement.  
  
Statements are ended with the semicolon symbol, and are not the same as lines. The end of a line means nothing to the Java compiler. It only helps make the code more readable.  
  
The assignment operator for assigning objects to references is exactly the same as the assignment operator for assigning primitive values. You shouldn't confuse it with the equality symbol of two equals signs next to each other.

Graphic

*The relevant syntax is  
  
=*

Syntax

*Object reference* = new *Classname*();

Declaring and initializing a reference variable is very similar to declaring and initializing a primitive type variable.  
  
The main difference is that you must create an object instance from a class for the reference  
variable to point to, before you can initialize the object instance.

Code

Shirt myShirt;  
myShirt = new Shirt();

To declare, instantiate, and initialize an object reference variable, you should do the following:

Code

Shirt myShirt;  
myShirt = new Shirt();

* declare a reference to the object by specifying its identifier and the type of object that the reference points to, which is the class of the object   
  *The relevant code is:  
    
  Shirt myShirt;*   
    
  **Code**  
  Shirt myShirt;  
  myShirt =new Shirt();
* create the object instance by using the new keyword, and   
  *The relevant code is:  
    
  new Shirt()*   
    
  **Code**  
  Shirt myShirt;  
  myShirt =new Shirt();
* initialize the object reference variable by assigning the object to the object reference variable   
  *The relevant code is:  
    
  myShirt =*   
    
  **Code**  
  Shirt myShirt;  
  myShirt = new Shirt();

When working with an object reference variable you should consider the following important points:

Code

Shirt myShirt = new Shirt();  
int shirtId = myShirt.shirtId;  
myShirt.display()

**Shirt myShirt = new Shirt();**

*You can declare and initialize the object reference in a single line as opposed to two lines.*

**int shirtId = myShirt.shirtId;**

*You use the dot operator with an object reference to manipulate the values, or to invoke the methods of a specific object. The dot notation is used to access a field of the object, in this case assigning it to a variable named shirtId.*

**myShirt.display()**

*You also use the dot notation to call a method on the object.*

Let's return to the analogy of using a remote control to operate an electronic device. To operate an electronic device with a remote, you need to pick up the remote, and possibly turn it on, and then press a button on the remote to do something on the video camera.  
  
Similarly, to do something with a Java object, you need to get its remote by creating an object and getting a reference to it – for example, myShirt. You then press its buttons by calling methods to get the object to do something – for example, display().

Code

Shirt myShirt = new Shirt();  
myShirt.display();

To extend the analogy just a little further, to work with a different object, say a flat-screen television, you need a remote for that object. In the Java world, you need a reference of the correct type for the object you are referencing.  
  
You can ignore the fact that there is such a thing as a universal remote controller.  
  
For the moment, let's just say that a reference of the same type as an object is one of the reference types that can be used, and is a good place to start exploring the world of Java objects.

This code shows objects being accessed with matching reference types. For example, the Shirt reference type is used to refer to a Shirt object, and the Trousers reference type is used to refer to a Trousers object.  
  
The type of the reference doesn't have to be identical to the type of the object, but it must be compatible with it. This flexibility is a great strength of Java.

Code

Shirt myShirt = new Shirt();  
myShirt.display();  
  
Trousers myTrousers = new Trousers();  
myTrousers.display();

References point to a particular object in memory. In this example, there are two objects in memory although they are both of type Shirt, and there are two Shirt references pointing to these two Shirt objects.

Graphic

*A diagram displays the myShirt object at memory address 0x034009. The shirtID is 12, the price is $15.99, and the colorCode is B. The yourShirt object is displayed at memory address 0x099f311, and its shirtID is also 12, the price is $15.99, and colorCode is B. A counter of 10 is also displayed.*

Code

int counter = 10;  
Shirt myShirt = new Shirt();  
Shirt yourShirt = new Shirt();

If the myShirt reference, after having its own object, is now assigned the reference yourShirt, the myShirt reference will drop its current object and be reassigned to the same object that yourShirt has.   
  
Now two references, myShirt and yourShirt, point to the same object. Any changes to the object made by using one reference can be accessed using the other reference and vice versa.  
  
Another effect of assigning the reference yourShirt to the reference myShirt is that if the previous object referred to by myShirt has no other references, it will now be inaccessible. In due course, it will be *garbage collected*, meaning that its memory will become available to store other objects.

Code

int counter = 10;  
Shirt myShirt = **new** Shirt();  
Shirt yourShirt = **new** Shirt();  
  
myShirt = yourShirt;

You can use either reference to make a change or get a value from the object. References yourShirt and myShirt refer to the same object, so making a change or getting a field value by using one reference is exactly the same as doing it with the other reference.

Code

int counter = 10;  
Shirt myShirt = newShirt();  
Shirt yourShirt = new Shirt();  
  
myShirt = yourShirt;  
  
myShirt.colorCode = 'R';  
yourShirt.colorCode = 'G';

Since myShirt and yourShirt now refer to the same object after the code completes, the colorCode field of the object will be G if you use this code.

Graphic

*The relevant code is  
  
System.out.println("Shirt color: " + myShirt.colorCode);*

Code

int counter = 10;  
Shirt myShirt = new Shirt();  
Shirt yourShirt = new Shirt();  
  
myShirt = yourShirt;  
  
myShirt.colorCode = 'R';  
yourShirt.colorCode = 'G';  
  
System.out.println("Shirt color: " + myShirt.colorCode);

If you use this code, you'll get the same result.  
  
Going back to the example of the television remote, it's the same as if you and a friend both have working remotes to the same television.

Graphic

*The relevant code is  
  
System.out.println("Shirt color: " + yourShirt.colorCode);*

Code

int counter = 10;  
Shirt myShirt =new**S**hirt();  
Shirt yourShirt = new Shirt();  
  
myShirt = yourShirt;  
  
myShirt.colorCode = 'R';  
yourShirt.colorCode = 'G';  
  
System.*out*.println("Shirt color: " + yourShirt.colorCode);

Question

Which statements accurately describe working with object references?

**Options:**

1. The reference and the newly created object must be in the same statement
2. The assignment operator for assigning objects to references is the same as for assigning primitive values
3. Declaring and initializing a reference variable is the same as declaring and initializing a primitive type variable
4. The dot operator is used to create a new object instance

Answer

***Option 1:****Correct. The way that the assignment operator, an equals symbol, works requires that the reference and the newly created object must be in the same statement – for example, Object reference = new Classname();.*

***Option 2:****Correct. The assignment operator for assigning objects to references is exactly the same as the assignment operator for assigning primitive values. You must not confuse it with the double equals sign equality symbol.*

***Option 3:****Incorrect. Declaring and initializing a reference variable is very similar to declaring and initializing a primitive type variable but it isn't the same. The main difference is that you must create an object instance from a class for the reference variable to point to before you can initialize the object instance.*

***Option 4:****Incorrect. The dot, or period symbol, operator can be used with an object reference to manipulate the values or to invoke the methods of a specific object. It isn't used to create a new object reference.*

**Correct answer(s):**

1. The reference and the newly created object must be in the same statement  
2. The assignment operator for assigning objects to references is the same as for assigning primitive values

**2. Creating and manipulating Java objects**

You can create instances of a class and manipulate these instances in several ways. You'll first create and initialize object instances and then manipulate object references.  
  
A Customer class is provided for you in an existing source. In this source, you can create, compile, and execute a CustomerTest class. In this test class, you create objects of the Customer class and set values to its member fields.

Graphic

*The NetBeans IDE 7.0.1 toolbar has a menu bar and a toolbar. The Projects window has three tabs: Projects, Files, and Services. No projects are currently open.*

To do this, you can create a new project from the existing source. Then you follow these steps:

* set the Source Package Folder to point to the required directory, change the Source/Binary Format property, and   
  *The New Java Project with Existing Sources dialog box has two panels: Source Package Folders and Test Package Folders. Each panel has Add Folder and Remove buttons. C:\labs\les06 is listed in the Source Package Folders list box.*
* open the Customer.java file in the editor and examine its member fields and its method   
  *The Projects window displays the tree structure for the project Practice06. The Source Packages folder contains the files Customer.java and PersonTwoTest.java. There is also a Libraries folder. The Customer.java file has the method displayCustomerInfo(), and the fields: customerID, emailAddress, and name.*

Next you create the CustomerTest class as a Java Main Class type. Since this class will be run by the Java executable, it must contain a main method.

Graphic

*The main class is:  
  
public class Customer (  
public int customerID = 0;  
public String name = "--name required-";  
public String emailAddress = "-email required-";*

The NetBeans IDE provides the skeleton of a main class for you. You follow these steps:

* select **File - New File** from the NetBeans menu,   
  *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.*
* in the Choose File Type window, select **Java Main Class** from the File Types column, click **Next**, and   
  *The New File wizard is displayed at the Choose File Type step. The Categories pane includes Java, Swing GUI Forms, JavaBeans Objects, and AWT GUI Forms. Java is selected. The File Types pane lists the available Java file type options, which include Java Class, Java Interface, Java Enum, and Java Exception. Java Main Class is selected in the File Types pane.*
* name the file CustomerTest, and click **Finish**   
  *The Name and Location page of the wizard is open. The Class Name field contains Customer Test, the Project field contains Practice06, the Location field contains Source Packages, the Package field is empty, and the Created File field contains C:\labs\les06\CustomerTest.java. A warning message suggests that Java classes are not placed in the default package.*

In the main method of CustomerTest, you need to add code to declare and initialize two instances of the Customer class. Within the body of the main method, you declare two fields of type Customer.

Graphic

*The relevant code is  
  
cust1 = new Customer()  
and  
cust2 = new Customer()*

Code

     \*/  
    public static void ***main***(String[] args) {  
        Customer cust1, cust2;  
        cust1 = new Customer();  
        cust2 = new Customer();  
    }  
}

Syntax

*variable name* = new *class name*();

Finally, you finish coding the main method by following these steps:

Code

     \*/  
    public static void ***main***(String[] args) {  
        Customer cust1, cust2;  
        cust1 = new Customer();  
        cust2 = new Customer();

* assign values to all of the member fields of one of the Customer objects and use the reference variable to qualify the field name   
  *The relevant code is  
    
  cust1.customerID = 1;  
  cust1.name = "Mary Smith";  
  cust1.emailAddress = "mary.smith@gmail.com";*   
    
  **Code**  
       \*/  
      public static void ***main***(String[] args) {  
          Customer cust1, cust2;  
          cust1 = new Customer();  
          cust2 = new Customer();  
            
          // Assign values to member fields  
          cust1.customerID = 1;  
          cust1.name = "Mary Smith";  
          cust1.emailAddress = "mary.smith@gmail.com";  
      }  
  }
* assign different values to each member field of the other Customer object, and   
  *The relevant code is  
    
  cust2.customerID = 2;  
  cust2.name = "Frank Jones";  
  cust2.emailAddress = "frank.jones@gmail.com";*   
    
  **Code**  
       \*/  
      public static void ***main***(String[] args) {  
          Customer cust1, cust2;  
          cust1 = new Customer();  
          cust2 = new Customer();  
            
          // Assign values to member fields  
          cust1.customerID = 1;  
          cust1.name = "Mary Smith";  
          cust1.emailAddress = "mary.smith@gmail.com";  
          cust2.customerID = 2;  
          cust2.name = "Frank Jones";  
          cust2.emailAddress = "frank.jones@gmail.com";  
     }  
  }
* invoke the displayCustomerInfo method of each object   
  *The relevant code is  
    
  cust1.displayCustomerInfo();  
  cust2.displayCustomerInfo();*   
    
  **Code**  
       \*/  
      public static void ***main***(String[] args) {  
          Customer cust1, cust2;  
          cust1 = new Customer();  
          cust2 = new Customer();  
            
          // Assign values to member fields  
          cust1.customerID = 1;  
          cust1.name = "Mary Smith";  
          cust1.emailAddress = "mary.smith@gmail.com";  
          cust2.customerID = 2;  
          cust2.name = "Frank Jones";  
          cust2.emailAddress = "frank.jones@gmail.com";  
            
          // Display customer info for each object  
          cust1.displayCustomerInfo();  
          cust2.displayCustomerInfo();  
     }  
  }

Next you click **Save** to compile and then run the CustomerTest.java file. Check the output to be sure that each Customer object displays the distinct values you assigned.

Code

run:  
\*\*\*\*\*\*\*\*Customer Information\*\*\*\*\*\*\*\*  
Customer ID: 1  
Name: Mary Smith  
Email: mary.smith@gmail.com  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*\*\*\*\*\*\*\*Customer Information\*\*\*\*\*\*\*\*  
Customer ID: 2  
Name: Frank Jones  
Email: frank.jones@gmail.com  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
BUILD SUCCESSFUL (total time: 1 second)

You also need to assign the value of one object reference to another object reference. To do this, you edit the main method of the CustomerTest class to assign one object reference to another object reference just above the first line of code that invokes the displayCustomerInfo method.

Graphic

*The relevant code is  
  
cust2 = cust1;*

Code

     \*/  
    public static void ***main***(String[] args) {  
        Customer cust1, cust2;  
        cust1 = new Customer();  
        cust2 = new Customer();  
          
        // Assign values to member fields  
        cust1.customerID = 1;  
        cust1.name = "Mary Smith";  
        cust1.emailAddress = "mary.smith@gmail.com";  
        cust2.customerID = 2;  
        cust2.name = "Frank Jones";  
        cust2.emailAddress = "frank.jones@gmail.com";  
          
        // Display customer info for each object  
        cust2 = cust1;  
        cust1.displayCustomerInfo();  
        cust2.displayCustomerInfo();  
   }  
}

Then you save and run the CustomerTest.java file. Check the output of the displayCustomerInfo methods for both objects. Both of the object references now point to the same object in memory so both of the displayCustomerInfo method outputs should be identical.

Code

run:  
\*\*\*\*\*\*\*\*Customer Information\*\*\*\*\*\*\*\*  
Customer ID: 1  
Name: Mary Smith  
Email: mary.smith@gmail.com  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*\*\*\*\*\*\*\*Customer Information\*\*\*\*\*\*\*\*  
Customer ID: 1  
Name: Mary Smith  
Email: mary.smith@gmail.com  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
BUILD SUCCESSFUL (total time: 4 seconds)

Question

What is the output from this code fragment?

**Code**  
Car myCar = new Car();  
Car yourCar = new Car();  
myCar.colorCode = "Red";  
yourCar.colorCode = "Green";  
System.out.println("Car color: " + myCar.colorCode);

**Options:**

1. Red
2. Green
3. Car color: Red
4. Car color: Green

Answer

***Option 1:****Incorrect. The string Car color: precedes the color value returned from this code fragment.*

***Option 2:****Incorrect. The string+ Car color: precedes the color value returned from this code fragment.*

***Option 3:****Correct. The output of this code fragment is the string Car color: followed by the value of myCar.colorCode, which is Red.*

***Option 4:****Incorrect. The output of this code fragment is the string Car color: followed by the value of myCar.colorCode, which is Red. It does not return the value of yourCar.colorCode, which is Green.*

**Correct answer(s):**

3. Car color: Red

**3. Summary**

In this topic, you've learned how to declare, instantiate, and initialize objects.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/jl_jsef_a02_it_enus/output/html/course_transcript.html#top)

Using the String and StringBuilder Classes

Learning Objective

*After completing this topic, you should be able to*

* *to use the String and StringBuilder classes*

**1. Concatenating strings**

The String class is one of the many classes included in the Java class libraries. The String class provides you with the ability to store a sequence of characters. Because you'll use the String class frequently throughout your programs, it's important to understand some of the special characteristics of strings in the Java programming language.

A String object

Syntax

String *object name*= " *value* ";

* is immutable so its value can't be changed, and   
  **Syntax**  
  String *object name*= " *value* ";
* can be used with the string concatenation operator symbol, the plus sign, for concatenation   
  **Syntax**  
  String *object name*= " *value* ";

A String object can be instantiated without using the new keyword. This is preferred.  
  
The new keyword can be used, but it's not best practice.

Code

String hisName = "Fred Smith";  
  
String herName = new String("Anne Smith");

Creating a String object using the new keyword creates two String objects in memory, while  
creating a String object by using a string literal creates only one object.   
  
Because of this, the latter practice is more memory-efficient.

Code

String hisName = "Fred Smith";

String objects support the use of a special concatenation operator – a plus sign – for concatenating two or more strings. Because a literal string returns a String reference, string literals and String references can be intermixed in an expression that concatenates a number of strings.  
  
When you use a string literal in Java code, it is instantiated and becomes a String reference. The concatenation creates a new string and the String reference – theirNames – now points to this new string.

Graphic

*The relevant code is  
  
theirNames*

Code

String name1 = "Fred"  
theirNames = name1 + " and " +   
             "Anne Smith";

Because String is immutable, concatenating two strings requires creating a new string. For example, you can have a String object myString containing the string Hello, which references the memory address 0x034009.

Graphic

*A diagram displays a myString object with a reference to memory address 0x034009, and the value Hello.*

Code

String myString = "Hello";

In this example, the string World is concatenated to the original string. The concat()method is being used here, but whether you use that or the concatenation operator of a plus sign, a new String object will be created and a new String reference is returned that points to this new object.  
  
For example, if the myString originally referenced a memory address of 0x034009, after concatenation, it will refer to a new address, such as 0x99f311. Because the original object is no longer referred to, it's now inaccessible and will be garbage collected.

Graphic

*The original memory address of 0x034009 is grayed out on the diagram.*

Code

String myString = "Hello";  
myString = myString.concat(" World");

Finally, on concatenating another string, this time using the concatenation operator, the same  
thing happens again. A new object is created and the reference for this object is assigned to  
myString.

Graphic

*The new Hello World object is now located at memory address 0x74cd23. The object at memory address 0x99f311 is now grayed out.*

Code

String myString = "Hello";  
myString = myString.concat(" World");  
myString = myString + "!"

Question

Identify the features of a String object.

**Options:**

1. It must be instantiated using the new keyword
2. Its value can be changed
3. It can be used with the string concatenation operator
4. It is immutable

Answer

***Option 1:****Incorrect. To avoid the unnecessary duplication of String objects in memory, you can create String objects without the new keyword.*

***Option 2:****Incorrect. A String object is immutable, so its value cannot be changed.*

***Option 3:****Correct. String objects support the use of a special concatenation operator, the plus sign, for concatenating two or more strings.*

***Option 4:****Correct. A String object is immutable, so its value cannot be changed. Because of this, concatenating two strings requires creating a new string.*

**Correct answer(s):**

3. It can be used with the string concatenation operator  
4. It is immutable

Like most classes, the String class has a number of useful methods. Almost all of these methods do their useful work by returning a single value, because Java only allows a single return from a method. The return type, essentially the type of the method, can be a primitive or a reference to an object.  
  
To be able to use the return value in your code, you will typically use the assignment operator to  
assign the value, or reference, to a type that you have declared for this purpose.

Graphic

*The relevant code  
  
length()*

Code

String hello = "Hello World";  
int stringLength = hello.length();

A method call can return a single value of any type. This code shows an example of a method of primitive type int.  
  
In this case, the code shows the use of reference hello to call the method length(). Because the object this reference refers to is the string Hello World, this method call will return the value 11 and place it in the variable stringLength. The method call length() is of type int.

Code

String hello = "Hello World";  
int stringLength = hello.length();

It's possible to have several method calls that return object references.

Code

String greet = " HOW ".trim();  
String lc = greet + "DY".toLowerCase();

**trim()**

*The String object HOW is instantiated and has trim called on it. As a string literal returns an object reference, this is exactly the same as calling the method trim() on the reference.*

**" HOW "**

*The string HOW has two spaces on either side of the word. The string returned will be just three characters long because these spaces will be removed. This new string will be referenced by greet.*

**toLowerCase()**

*A method call doesn't have to be assigned to a type, but can simply be used in an expression. The method toLowerCase() is called on the string uppercase DY, returning dy. The string lc will now reference an object containing "HOWdy".*

An alternative version with parentheses ensures that the two strings are concatenated creating a new string, before toLowerCase() is called. The string lc now references an object containing "howdy".

Code

String lc = (greet + "DY").toLowerCase();

Some method calls require an argument to be passed into the method. For example, the substring() method needs an index, an int, to indicate where to split the string.  
  
The method returns a new string that consists of the remaining part of the string starting at W, so in this case it returns World. substring indexes from zero. It begins with the character at the specified index and extends to the end of this string. W is at index six.

Graphic

*The relevant code is  
  
substring*

Code

String theString = "Hello World";  
String partString = theString.substring(6);

The method endsWith() requires a String reference to be passed as an argument. It returns a boolean because it simply determines if the string ends with the sequence of characters passed in. In this case it does, so true will be returned.

Graphic

*The relevant code is  
  
endsWith*

Code

boolean endWorld = "Hello World".endsWith("World");

**2. Using the StringBuilder class**

StringBuilder provides a mutable alternative to String. This is because Stringbuilder

* is a normal class, and is instantiated using the new keyword
* has an extensive set of methods for append, insert, and delete
* has many methods that return reference to the current object, so there is no instantiation cost, and
* can be created with an initial capacity best suited to need

String is still needed because

* it may be safer to use an immutable object
* a class in the API may require a string, and
* it has many more methods not available on StringBuilder

Unlike String, there is no shortcut to instantiate a StringBuilder. It's simply instantiated like any other object using the new keyword. StringBuilder is not a complete replacement for String, but it's more suitable if many modifications are likely to be made to the string represented by the data type.

There are a number of advantages to using StringBuilder over String for concatenation.  
  
When the strings Hello and World are concatenated, a new String object is created, and the reference for that object is assigned to  
myString.

Graphic

*The diagram displays myString with a reference to Hello World at memory address 0x74cd23.*

Code

String myString = "Hello";  
myString.concat(" World");

A new StringBuilder is instantiated, populated with the string Hello, and the reference for this new object is assigned to mySB.

Graphic

*The diagram displays mySB with a reference to Hello at memory address 0x034009.*

Code

StringBuilder mySB = new StringBuilder("Hello");

To append the string World, all you need to do is call the append() method and pass in World. No assignment is necessary because there already is a reference to the StringBuilder object. This StringBuilder object now contains a representation of the combined strings Hello World.  
  
Even if you did assign the return type of the append() method, which is StringBuilder, there would still be no object creation cost, because the append() method modifies the current object and returns the reference to that object, the one already contained in mySB. This can be useful to know if the entire method call is used as a type.

Graphic

*The relevant code is  
  
append(" World")*

Code

StringBuilder mySB = new StringBuilder("Hello");  
mySB.append(" World");

You need to create and initialize String objects and print their contents. You also need to create and manipulate StringBuilder objects.  
  
Follow these steps:

Code

/\*  
 \* To change this template, choose Tools | Templates  
 \* and open the template in the editor.  
 \*/  
  
/\*\*  
 \*  
 \* @author Administrator  
 \*/

* Create a new Java class called PersonTwo.   
    
  **Code**  
  /\*  
   \* To change this template, choose Tools | Templates  
   \* and open the template in the editor.  
   \*/  
    
  /\*\*  
   \*  
   \* @author Administrator  
   \*/  
  public class **PersonTwo** {
* Declare and instantiate two member fields of type StringBuilder to hold the person's name and phone number, respectively. For the name field, initialize the capacity of the StringBuilder object to eight.   
    
  **Code**  
      public StringBuilder name = new StringBuilder(8);  
      public StringBuilder phoneNumber = new StringBuilder();
* Create a new method called displayPersonInfo.   
    
  **Code**  
        
      public void **displayPersonInfo**(){

In the body of the displayPersonInfo method, you need to populate and then display the name object, and ensure that the total number of characters in the name exceeds the initial capacity of the object, eight in this case. To do this, follow these steps:

Code

/\*\*  
 \*  
 \* @author Administrator  
 \*/  
public class **PersonTwo** {  
    public StringBuilder name = **new** StringBuilder(8);  
    public StringBuilder phoneNumber = **new** StringBuilder();  
      
    public void **displayPersonInfo**(){  
        // Populate the name object  
        name.append("Fernando");  
        name.append(" ");  
        name.append("Gonzalez");  
        // Display the name and the capacity of the object  
        System.*out*.println("Name: "+ name.toString());  
        System.*out*.println("Name object capacity: " + name.capacity());

**append a first name**

Use the append method of the StringBuilder class to append a first name.  
*The relevant code is  
  
name.append("Fernando");*

**add a space, and then a last name**

Use the same method in two separate invocations to first add a space, and then a last name. Ensure that total number of characters that you have added to the name object exceeds eight.  
  
You can accomplish the same thing using a String object and concatenating additional values. However, this would be inefficient because a new String object is created with each concatenation. String object capacity cannot be increased as Strings are immutable.  
*The relevant code is  
  
name.append(" ");  
name.append("Gonzalez");*

**display the entire name value, and**

Use the System.out.println method to display the entire name value. You can embed the toString method of the name object within the System.out.println method.  
*The relevant code is  
  
System.out.println("Name: "+ name.toString());*

**display the capacity of the name object**

Display the capacity of the name object, using the capacity method. The StringBuilder object will have dynamically increased the capacity to contain all of the values that you have appended.  
*The relevant code is  
  
System.out.println("Name object capacity: " + name.capacity());*

Then you click **Save** to compile and then run the PersonTwoTest.java file. The capacity has been increased from the initial setting of eight to accommodate the full name.

Graphic

*The output displays the following result:  
  
run:  
Name: Fernando Gonzalez  
Name object capacity: 18  
BUILD SUCCESSFUL (total time: 0 seconds)*

To populate and manipulate the phoneNumber object, you need to append a string of digits and then use the insert method to insert dashes at various index locations, achieving the format nnn-nnn-nnnn.  
  
You begin by using the append method of the StringBuilder class to append a String value consisting of ten numbers.

Graphic

*The relevant code is:  
  
phoneNumber.append("5551234567");*

Code

public void **displayPersonInfo**(){  
        // Populate the name object  
        name.append("Fernando");  
        name.append(" ");  
        name.append("Gonzalez");  
        // Display the name and the capacity of the object  
        System.*out*.println("Name: "+ name.toString());  
        System.*out*.println("Name object capacity: " +  
             name.capacity());  
          
        // Replace first name  
        System.*out*.println("First Name: "+ name.substring(0, 8));  
                  
        // Populate the phoneNumber object  
        phoneNumber.append("5551234567");

Then you insert a dash at offset position three. This puts the dash at the fourth position in the String.  
  
For example, given the string 5551234567, offset position three occurs at the number one, if the index begins at zero. If the dash is inserted at offset position three, it pushes the number currently at that position, and all remaining numbers, over to the next offset position.

Graphic

*The relevant code is:  
  
phoneNumber.insert(3, "-");*

Code

public void **displayPersonInfo**(){  
        // Populate the name object  
        name.append("Fernando");  
        name.append(" ");  
        name.append("Gonzalez");  
        // Display the name and the capacity of the object  
        System.*out*.println("Name: "+ name.toString());  
        System.*out*.println("Name object capacity: " +  
             name.capacity());  
          
        // Replace first name  
        System.*out*.println("First Name: "+ name.substring(0, 8));  
                  
        // Populate the phoneNumber object  
        phoneNumber.append("5551234567");  
        phoneNumber.insert(3, "-");

Syntax

*object reference*.insert(*int offset*, *String str*);

Next you follow these steps:

Code

public void **displayPersonInfo**(){  
        // Populate the name object  
        name.append("Fernando");  
        name.append(" ");  
        name.append("Gonzalez");  
        // Display the name and the capacity of the object  
        System.*out*.println("Name: "+ name.toString());  
        System.*out*.println("Name object capacity: " +  
             name.capacity());  
          
        // Replace first name  
        System.*out*.println("First Name: "+ name.substring(0, 8));  
                  
        // Populate the phoneNumber object  
        phoneNumber.append("5551234567");  
        phoneNumber.insert(3, "-");

* Insert a dash at offset position seven where the number four is currently placed.   
    
  **Code**  
          phoneNumber.insert(7, "-");
* Use System.out.println to display the output from the StringBuilder object's toString method.   
    
  **Code**  
          System.*out*.println("Phone number: " +  
              phoneNumber.toString());  
      }  
  }
* Click **Save** to compile, and run the PersonTwoTest.java file. Check the output from the displayPersonInfo method and ensure that the dashes appear between the third and fourth digits, and between the sixth and seventh digits.   
  *The output displays the following result:  
    
  run:  
  Name: Fernando Gonzalez  
  Name object capacity: 18  
  Phone number: 555-123-4567  
  BUILD SUCCESSFUL (total time: 0 seconds)*

The substring method of the StringBuilder class is used to get just the first name value in the name object. This substring method takes the start index and the end index for the substring.  
  
The start index of the substring method is inclusive, and is the actual index of the first character you want returned. The end index is exclusive, and is the index of the character just to the right of the last character of your substring.

Code

public void **displayPersonInfo**(){  
        // Populate the name object  
        name.append("Fernando");  
        name.append(" ");  
        name.append("Gonzalez");  
        // Display the name and the capacity of the object  
        System.*out*.println("Name: "+ name.toString());  
        System.*out*.println("Name object capacity: " +   
                name.capacity());  
          
        // Replace first name  
        System.*out*.println("First Name: "+ name.substring(0, 8));  
                  
        // Populate the phoneNumber object  
        phoneNumber.append("5551234567");  
        phoneNumber.insert(3, "-");  
        phoneNumber.insert(7, "-");  
                System.out.println("Phone number: " +   
        phoneNumber.toString());  
   }  
}

Syntax

*object reference*.substring(int start, int end);

Note

*Indexes for characters in the StringBuilder class, much like array indexes, are zero-based. The first character in the StringBuilder is located at position, or index, zero.*

You save and again run PersonTwoTest.java. Check the output and make any adjustments necessary to the index numbers in order to get the correct first name value.

Graphic

*The output displays the following result:  
  
run:  
Name: Fernando Gonzalez  
Name object capacity: 18  
First name: Fernando  
Phone number: 555-123-4567  
BUILD SUCCESSFUL (total time: 0 seconds)*

Question

Identify the features of a StringBuilder object.

**Options:**

1. It's instantiated using the new keyword
2. Methods return references to other objects, so there is an instantiation cost
3. It has an extensive set of methods for insert, and delete, but not append
4. It doesn't replace String objects

Answer

***Option 1:****Correct. Unlike String, there is no shortcut to instantiate a StringBuilder. It is simply instantiated like any other object by using the new keyword.*

***Option 2:****Incorrect. Many of its methods return a reference to the current object, so there is no instantiation cost.*

***Option 3:****Incorrect. StringBuilder has an extensive set of methods for insert, and delete, as well as append.*

***Option 4:****Correct. StringBuilder is not a complete replacement for String, but it is more suitable if many modifications are likely to be made to the string represented by the data type.*

**Correct answer(s):**

1. It's instantiated using the new keyword  
4. It doesn't replace String objects

**3. Summary**

In this topic, you've learned how to use the String and StringBuilder classes.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/jl_jsef_a02_it_enus/output/html/course_transcript.html#top)

Using the Java API Documentation

Learning Objective

*After completing this topic, you should be able to*

* *look up classes and methods in the API documentation*

**1. Using the Java API documentation**

Question

Which statements accurately describe the Java API documentation?

**Options:**

1. It is only available with Java SE 7
2. It documents Java technology class libraries
3. It is a series of hyperlinked XML pages
4. To find a class, you must know its package

Answer

***Option 1:****Incorrect. The Java API is also available for earlier versions of Java.*

***Option 2:****Correct. The documentation lists all the classes in the API, including descriptions of what the classes do and a list of their constructors, methods, and fields.*

***Option 3:****Incorrect. The class library specification is a series of Hypertext Markup Language, or HTML, web pages that you can load in your web browser.*

***Option 4:****Incorrect. Java classes are organized into packages, but if you don’t know the package of a particular class, you can select****All Classes****in the documentation.*

**Correct answer(s):**

2. It documents Java technology class libraries

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/jl_jsef_a02_it_enus/output/html/course_transcript.html#top)

© 2016 Skillsoft Ireland Limited