

More on Variables

DECLARING AND USING VARIABLES

Objectives

Understanding rules for declaring variables

Understanding how to change the value of a variable

Understanding the difference between a primitive and object value

- As we have seen, naming is a crucial part of programming. One of the most frequent things you will provide names for when you program is variables.
- This should be a review of some of the material covered previously. Here we will dive a little deeper into each topic.

more on variables

3

Variables

- The syntax for declaring a variable is *dataType name*; You can optionally also assign a value to a variable *type name [= value]*;
 - `World world;`
 - `World earth = new World();`
 - `int x = 0;`
 - `int y = 0;`
- This code illustrates what variables look like to you, the programmer.

more on variables

4

- There are a couple of rules you need to keep in mind when declaring variables:
 - You must declare a variable before you can use it
 - It may only be declared once
 - You can re-use a variable once it has been declared
 - A variable's *type* can include any of the following:
 - primitive types (which start with a lower case letter) hold numbers, characters, and so on, *e.g.*,
int, double, float, char, boolean
 - class types (easy to identify because these types always start with an uppercase letter) hold objects. This could include classes from Java's library, such as **String** or classes created by you or someone else such as **Turtle** and **World**

more on variables

5

- Here is a table describing some of the primitive types (or see [Sun's more detailed information](#))

Type	Size	Description
int	32 bits	Stores an integer number
float	32 bits	Stores a decimal number
double	64 bits	Stores even larger decimal numbers
boolean	1 bit	Stores the values "true" or "false" (as 1 or 0)
char	16 bits	Stores a Unicode character

more on variables

6

Learn by doing

- Which of the following is the correct way to declare a variable that represents the cost of gasoline today?
 - a. `declare double gasPrice = 0.0;`
 - b. `int gasPrice = 0.0;`
 - c. `Double gasPrice;`
 - d. `double gasPrice = 0.0;`
 - e. `integer gasPrice = 0;`

more on variables

7

Our answer

- `declare double gasPrice = 0.0;`
 - That is not quite right. `declare` is not a keyword, just a term that we use to describe how you create a variable.
- `int gasPrice = 0.0;`
 - That is not quite right. To get the right answer, first, decide whether this is a number, something that is true or false, or a piece of text. If you decided it was a number, you need to decide whether it might include fractional amounts or not. Your variable should be a `double`, since the price of gas is a number that may include fractional amounts.
- `DOUBLE gasPrice;`
 - That is not quite right. `DOUBLE` is in all caps, but Java is expecting the word `double` in lowercase. Java is case-sensitive, which means that you have to use the correct capitalization for your code to work.

more on variables

8

Our answer

- `double gasPrice = 0.0;`
 - Good job! Your variable should be a double, since the price of gas is a number that may include fractional amounts. To get the right answer, first, decide whether this is a number, something that fractional amounts. To get the right answer, first, decide whether this is a number, something that might include fractional amounts or not. Java is case-sensitive , , which means that you have to use the correct capitalization for your code to work.
- `integer gasPrice = 0;`
 - That is not quite right. `integer` is not a keyword (`int` is). Also, you should think about whether your number might include fractional amounts. In that case a `double` is more appropriate.

more on variables

9

- Equally important is what is going on behind the scenes. Remember that variable names are really just a way for you to refer to a place in the computer's memory. Sort of like a mailbox holds letters, this place in memory holds the current value of your variable. When you declare a variable, Java uses the type of the variable to decide how much space to allocate (reserve) for it in memory.
- For example, an `int` is 4 bytes long, meaning it can store numbers up to about 4 billion ($2^{32}-1$, to be exact). A `boolean`, in contrast, has only two values, so it can be stored in a single bit.

more on variables

10

- Primitive types are pre-defined in Java, and their size is easy to determine. But what about types that can have variable size?
- For example, a **String** could be short ("Hi") or long ("This is a much longer string"). You might even make a **String** longer after you first create it.
- To handle this, Java assigns the variable to a special spot in memory that holds the address of another spot in memory. That way, if it needs to move the actual data around (for example if it gets larger), Java just updates the address (called a *reference*), and the **String** knows where to find its data.
- In fact, all object variables are handled this way in Java.

more on variables

11

- object variable
 - (definition) An object variable is a variable whose type is a class type instead of a primitive type.

more on variables

12

Variables and memory



- In this image, the following variables are represented:
- `int x=0;` and `World earth = new World();`. The value of `x`, `0`, is placed directly into the part of memory `x` points at because `int` is a primitive data type. The value of `earth` is placed in box `142` (see the picture of `earth` peeking out?). `earth` can find this object because its box holds the number `142`, a reference to the place in memory where the object is located.

more on variables

13

- To summarize, the variables with primitive types like **int** and **boolean** are stored directly into memory.
- In contrast, the values of other types like **String** and other objects may take up more space.
- Java in this case stores an *address* that it can use to find the actual information associated with an object.

more on variables

14

Memory Mapping

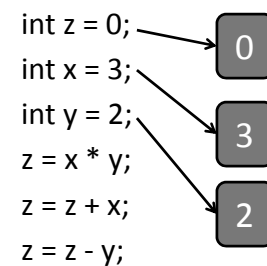
- Now let's look in more depth at how Java stores values in memory and retrieves them from memory. Whenever you refer to a variable in code, you may either tell Java to retrieve a value from memory, or to store a new variable in memory.
- The following illustrates six lines of code. The first three lines are used to declare three variables. The following lines all retrieve information from one or more variables, complete a calculation, and store the result in one of the three variables.

```
int z = 0;  
int x = 3;  
int y = 2;  
z = x * y;  
z = z + x;  
z = z - y;
```

more on variables

15

- The first three lines of the code declare int variables and assign int values to those variables: z with 0, x with 3, and y with 2. These variables are primitive variables because int is one of the primitives of Java. The last three lines of the code perform a series of arithmetic calculations and store the result to the variable.




more on variables

16

- Firstly, the code performs the multiplication of x and y. During the course of calculation, it retrieves the values from each of x and y (3 and 2 respectively) and do the calculation.
- $z = 3 * 2;$
- Notice that we are just retrieving the values from variables, so the values of x and y do not change. The result of the multiplication is stored back to z, which means the value of z is changed from 0 to 6.

```
int z = 0;
int x = 3;
int y = 2;
z = x * y;
z = z + x;
z = z - y;
```




A diagram showing three variables: z, x, and y. Each variable is represented by a text label followed by an arrow pointing to a rounded square box containing a number. The arrow from 'int z = 0;' points to a box containing '6'. The arrow from 'int x = 3;' points to a box containing '3'. The arrow from 'int y = 2;' points to a box containing '2'.

more on variables

17

- Same steps are used for the remaining steps.
 - $z = 6 + 3;$
 - $z = 9 - 2;$
- At the end of the code, the value of z is now changed into 7.

```
int z = 0;
int x = 3;
int y = 2;
z = x * y;
z = z + x;
z = z - y;
```



A diagram showing three variables: z, x, and y. Each variable is represented by a text label followed by an arrow pointing to a rounded square box containing a number. The arrow from 'int z = 0;' points to a box containing '7'. The arrow from 'int x = 3;' points to a box containing '3'. The arrow from 'int y = 2;' points to a box containing '2'.

more on variables

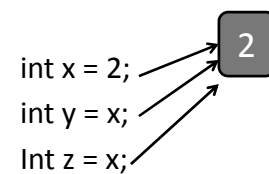
18

- Each int variable can take 4 bytes of space. From this code, total of three variables are declared (x, y, and z), so 12 bytes are used for this code for the variables.
- Note that by retrieving the values of variables, we can reuse those variables and its values over and over again throughout the code. This implies that we are not necessary to declare the same variables again to use them.

more on variables

19

- In a similar way, several variables can reference the same object or spot in memory.
- The following is the valid statement:
- `int x = 2; int y = x; int z = x;` Here, y and z reference the same spot in memory, which is x. Both y and z have the value 2, which is retrieved from x.
- This is a process called "memory mapping" which is extremely valuable.



more on variables

20

- Expert programmers will frequently use this process when they encounter a difficult problem with their code. Beginning programmers will find the process equally beneficial. You can draw a memory map on paper very easily -- for example, you can go through the same process with a pencil, eraser, and piece of paper.
- The next exercise asks you to draw a memory map for four variables. Notice that one of them is not a primitive variable.

more on variables

21

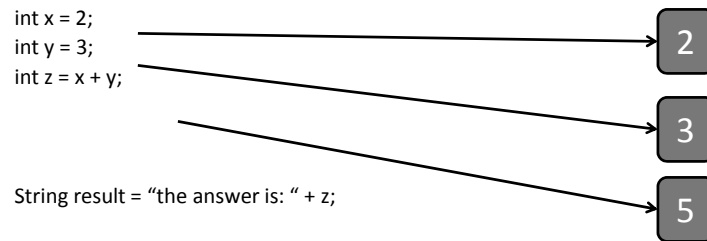
Learn by doing

- On a piece of paper, draw the memory used for
 - `int x = 2, int y = 3, int z = x + y` and `String result = "The answer is: " + z.`
- Which of these variables uses a references as opposed to directly storing the value?

more on variables

22

Our answer



- Any variable whose type is int is a primitive variable.
- Primitive variables are directly stored in memory.
- A variable of type String is an object variable, and will hold a reference to the actual object.
- All calculations are completed before the results are stored in memory.

more on variables

23

Variable Practice

- It is important that you develop a clear understanding of how variables work. This section includes a series of exercises demonstrating different implications of the ways in which variables are stored and updated.
- The first set of exercises explores a series of statements involving primitive variables. You will be asked to reason about when a variable is assigned, when its value is changed, and so on

more on variables

24

Learn by doing

```
int z = 0;
int x = 3;
int y = 2;
z = x * y;
z = z + x;
z = z - y;
```

1. What is the value of z at the end?
2. How many variables were declared?
3. How many bytes of space does z take?
4. How many bits of information can be stored in a boolean type variable?
5. Are we able to store any integer value in a variable of int type?

more on variables

25

Our answer

1. What is the value of z at the end?
 - Each new calculation updates the value of z. The final result is 7.
2. How many variables were declared?
 - x, y, and z are all variables. A variable is declared using the syntax type name or type name = value.
3. How many bytes of space does z take?
 - z is an int, so it takes up 4 bytes of space. No matter what values we assign to z, once it is declared, it will always take up 4 bytes, since that is the amount of memory allocated to it.
4. How many bits of information can be stored in a boolean type variable?
 - The answer is 1. A boolean can store two values (true and false), so this can be stored in a single bit, since a bit can be either on or off, 1 or 0.
5. Are we able to store any integer value in a variable of int type?
 - The answer is no. An int is four bytes, or 32 bits, so it can only store numbers between -2^{31} and $2^{31} - 1$ (a total of 2^{32} values)

more on variables

26

- The second set of exercises explores a series of statements involving an object variable.
- You will be asked to reason about when the variable is assigned, how many objects are being created, and so on.
- This is tougher because we have not focused on Objects. Try and see if you have a basis of understanding about objects.
- When in doubt, try the code out in your compiler.

more on variables

27

Learn by doing

```
String str1 = "Hello World!";
str1.toLowerCase();
str1 = str1.toUpperCase();
```

- If you check out the API's for the String class you will find that `str1.toLowerCase()` will convert the text to lower case – but will it overwrite what is stored in `str1`?

1. What is the value of `str1` after line two is executed?
2. What is the value of `str1` after line three is executed?
3. How many variables were declared?
4. How many objects were created?
5. We write the statement: `String str2 = str1;` How many objects does the statement above create, assuming `str1` is a valid reference to a String?

more on variables

28

Our answer

1. What is the value of str1 after line two is executed?
 - A variable is only modified when you use the equal sign to assign a new value to it. str1 contains "Hello World!" until you assign something else to it. toLowerCase() is a method that will compute a version of "Hello World" in lowercase, but it does not change the value of str1.
2. What is the value of str1 after line three is executed?
 - The correct answer is HELLO WORLD! A variable is modified when you use the equal sign to assign a new value to it. toUpperCase() returns the string "HELLO WORLD!" which is then assigned to str1, overwriting its previous value.
3. How many variables were declared?
 - A variable declaration should have a type and a name. String str1 = "Hello World!" is the only line of code that declares a variable here.

more on variables

29

Our answer

4. How many objects were created?
 - "Hello World", "hello world" and "HELLO WORLD" are all separate objects. Each new string created is a new object: this includes "Hello world") and the result of str1.toUpperCase("HELLO WORLD").
5. We write the statement: String str2 = str1; How many objects does the statement above create, assuming str1 is a valid reference to a String?
 - When we assign a variable to an object, it creates a reference to that object. It doesn't make a copy or create any new objects. In this case, when we assign str2 to str1 they now point to the same string. In this case, str2 and str1 point to the same string because str2 is assigned to str1.

more on variables

30

Summary

- You can declare a variable by specifying its type and name. A variable can be assigned a value either when it is declared or at any later time. Any previous values are overwritten.
- Object variables store a reference to the place in memory that their object occupies. Primitive variables simply store the value assigned to them.
- Rules for declaring variables
 - You must declare a variable before you can use it, using the syntax type name; or type name = value;
 - It may only be declared once
 - You can re-use a variable once it has been declared
 - A variable's type can include any of the following:
 - primitive types (which start with a lower case letter)
 - e.g., int, double, char, boolean
 - class types (easy to identify because these types always start with an uppercase letter)
 - This could include classes from Java's library, such as String or classes created by you or someone else such as World

more on variables

31