Summer Assignment for AP Computer Science

Go to http://codingbat.com/java. Sign up for a free account. Your assignment is to complete ALL of Warm-up 1 (except stringE and everyNth). If you are new to coding, this assignment will take several hours. DO NOT START AT THE LAST MINUTE!!!

I've attached my notes to help you get started. Also there are solutions, notes, and videos on CodingBat.

You will have a test covering Warm-up 1 on Friday Aug 23.

Email: daniel_nagel@dpsk12.org

Over the summer you can communicate through email.

daniel_nagel@dpsk12.org Butler 4

AP Computer Science A: Getting Started Coding

The following is a very basic introduction to coding in Java. The goal is to be able to start coding on **CodingBat.com.** At first, we can think of coding as a game. The game is to solve puzzles by making step by step solutions. We need to learn a new language to write the steps. Let's take a first look at the new language:

Quick and Dirty Java You Should Know to Get Started

Java Code	What it means
int	An integer variable
boolean	True or False variable
String	A collection of characters for example: "Hello"
=	Sets or *assigns* the left hand side to the right hand side
==	Compares or checks if equal
;	Ends a statement
if( )	Control that runs statements only if the expression inside the parenthesis is true
else	Can be used after an if-statement to run code when an if-condition is false
return	Ends a method and returns relevant data like an answer or result.
{ }	Used to encapsulate pieces of code such as methods and bodies of if-statements
&&	Logical AND
||	Logical OR
!	Logical NOT
+	Addition
-	Subtraction
*****	Multiplication
/	Division
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

Consider the problem: Given two distinct integers, a and b, return the larger one.

The following code solves the puzzle:

```
if(a > b) {
      return a;
}
else {
      return b;
}
```

Even if this is the first time you've seen Java code, the idea is easy to decipher/guess. The program checks if a is greater than b. If this is true then it returns a. Otherwise b must be greater than a (since they are distinct) so the code returns the answer b.

The precise use of code such as **if-statements** is introduced further down in these notes. There are also videos and notes on CodingBat to help you get started. For now, notice basic patterns such as the `return` statement returns an answer and then ends the code. Semicolons are used to end statements. We use curly braces like { } to surround pieces of code. We use variables, in this case `a` and `b`, to store data such as numbers. Let's look at another problem.

Suppose we are given two integers `x` and `y`. We need to return their sum, unless they are both the same, in which case we return double their sum.

The following code solves the puzzle:

```
int sum = x + y;
if(x == y){
      return 2*sum;
}
else {
      return sum;
}
```

Notice that we declared a new integer variable called `sum` and set it equal to `a + b`. Notice the `int` part. We have to tell Java what kind of data the variable `sum` is going to store. At first we will only work with three types of variables.

Variables

- `int` Stands for an integer (whole number: positive, negative, or zero)
- `boolean` Stands for boolean… it can only have two values: true or false.
- `String` Stands for String… a collection of characters.

Notice also the difference between = and ==. = is used to set something equal. In this case `sum = a + b.` == is used to check if primitives (in this case the integers) are equal.

One last example and then you can continue reading through the notes below, or jump on CodingBat and try problems in WarmUp-1. There are solutions to each problem. Also check the Notes and Videos on CodingBat for more detailed explanations and lots of examples.

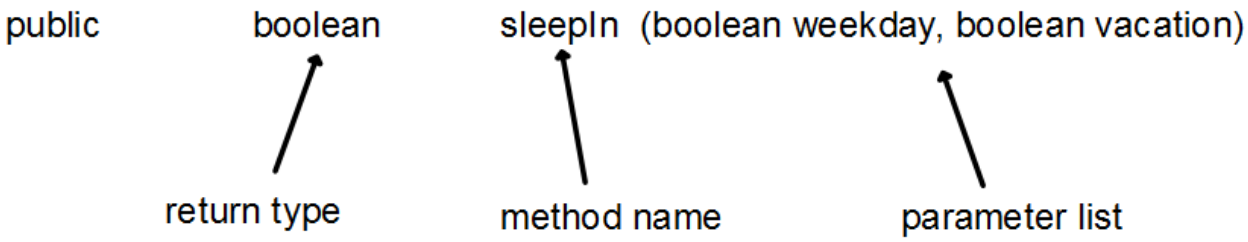
Consider the problem: You are given three integers called `len, wid, ht` and a boolean variable called `area.` In all cases if any of the integers are not positive then you should return 0. If the variable `area` has the value `true` then you should return `len * wid.` Otherwise you return `len * wid * ht.`

The following code solves it:

```
if( len <= 0 || wid <= 0 || ht <= 0){
      return 0;
}
else if( area == true){
      return len * wid;
}
else {
      return len * wid * ht;
}
```

Methods

CodingBat ENTIRELY deals with methods. A method is like a mini program that does a specific task. Methods are essential to Java and will be written and used throughout the course. On CodingBat methods always look something like this:



The code you put in the method directly follows and MUST BE ENCLOSED within braces { }

- For now, let's not worry about the word public.

- We can think of the **return type** as the type of information our method will return when it runs. At first we will only be concerned with three kinds of **types**: `int, boolean,` and `String`.

- The **method name** is what we call our method…duh. In CodingBat this isn't important since we only work with one method at a time and they already decide what the name is anyway. Later we will write programs that have many different methods that do different things and we will want to create names that are simple and clear.

- A method ALWAYS stops executing after it reaches a **return** statement. Later in class we will encounter methods that don't return anything. For now all of our methods will return either an `int, boolean,` or `String.`

Example 1: Suppose we wanted to write a method that computes the area of a rectangle. On CodingBat the beginning might look something like this:

```
public int findRectangleArea(int base, int height){

}
```

Notice the **return type** is an `int`. The **method name** is `findRectangleArea`, and the **parameters** (which in this case are both of **type** `int`) are `base` and `height`.

A very simple implementation of this method might look like:

```
public int findRectangleArea(int base, int height){
     return base * height;
}
```

Example 2: A slightly more involved area method (with the precondition that `numSides` must equal 3 or 4) could be:

```
public int findArea(int numSides, int base, int height){
     if(numSides == 3){
          return (base * height) / 2;
     }
     else {
          return base * height;
     }
}
```

Notice there are two **return** statements. However each execution of the method will only ever reach one of the **return** statements. (Make sure you understand why this is true….)

if-statements

(An introduction. More to follow later.)

The basic structure of an **if-statement** is as follows:

```
if   ( condition that can be true or false) {
     statements to execute
}
```

Notice parenthesis are around the `condition` and braces are around the `statements`. Later we'll learn we can sometimes leave out the braces but it is never wrong to include the braces.
The `statements` will be run by the computer only if the `condition` is `true`. If the `condition` is `false`, the program skips the `statements` and evaluates the first piece of code immediately following the last brace of the **if-statement.**

Example 3:

```
if( 5 > 2) {
     return true;
}
```

This (silly) example will always return `true`.

Example 4: Suppose in this example the variable `gradeLevel` is an `int`.

```
if( gradeLevel < 9){
     return true;
}
return false;
```

In example 4 if `gradeLevel >= 10` then the computer would not execute the code inside the braces. Instead it would skip down to the first statement after the braces (in this case `return false`).

if…else-statement

(An introduction. More to follow later.)

The basic structure of an **if…else-statement** is as follows:

```
if   ( condition that can be true or false) {
     statements to execute
}
else {
     other statements to execute
}
```

If the `condition` is `true` then the program will run `statements to execute` and then skip `other statements to execute`. However if the `condition` is `false` then the `other statements to execute` will be run. Notice the **`else`** part of code does not have its own `condition` in parenthesis. The **`else`** part depends entirely on the original **if-statement's** `condition` and will only run if this `condition` is `false`.

Boolean Operators

(A very brief introduction. More to follow later.)

We can combine boolean expressions to evaluate more complicated situations. Let's look at three of the operators:

Operator	Meaning
&&	AND
||	OR
!	NOT

Example 5: Suppose we want to return `true` if the `int num` is between 10 and 20 and return `false` otherwise. In English, we require that `num` is greater than 10 AND less than 20. Our code could look something like:

```
if ( num > 10 && num < 20){
     return true;
}
else {
     return false;
}
```

Example 6: Let's start with two `ints: num1 and num2`. Suppose we want to return true if `num1` is negative or `num2` is negative or both are negative and return false otherwise. Our simple code could be:

```
if( num1 < 0 || num2 < 0){
     return true;
}
else {
     return false;
}
```

Finally, the NOT operator simply flips a value from `true` to `false` or vice versa.

Strings

A **`String`** is a data *type* similar to how `int` and `boolean` are data *types*. A String stores a sequence of characters and is identified in Java by surrounding the characters with quotation marks.

Example 7:
```
"This is a string"
"So is this."
"A string can be empty like…"
""
"A string can have just numbers and characters like…"
"34234234234  blah blah sdfs****!!!  "
```

Unlike `int` and `boolean` **`Strings`** are not *primitive* types. They are *objects* from the *class* `String`. For now, all that means to us is that we cannot use == to compare Strings. Instead we use the **`.equals`** method.

Example 8:
```
String word1 = "Hello";
String word2 = "Hi";

boolean check = word1.equals(word2);
//check would equal false

boolean checkAgain = word2.equals("Hi");
// checkAgain would equal true
```

String Index

The position of each character in a `String` is given a number called its **index.** Java starts with index 0.

Example 9:
```
String word = "Hello"
```

"H" is at index 0, "e" is at index 1, etc.

```
Character:  H e l l o
Index:      0 1 2 3 4
```

Concatenation Operator

This is a fancy word for the + operator. You can make new Strings by using + between previous strings.

Example 10:
```
String datBoi = "fellow";
String str = "Hello " + datBoi + " kids";
// str would equal "Hello fellow kids"
```

String Methods

The following methods from the String class are on the AP exam and are generally very helpful:

`int length()`

Returns how many characters are in the String.

`String substring(int startIndex)`

Returns a new String that starts with the character at `startIndex` and goes to the end of the String.

`String substring(int startIndex, int endIndex)`

Returns a new String that starts with the character at `startIndex` and ends with the character at `endIndex - 1`. Remember this by saying "You start with the character you **want** and you end with the character you **don't want**."

`int indexOf(String str)`

Returns the index of the first occurrence of `str`. If `str` is not found then it returns -1.

For the following examples let:

```
String word1 = "Hello";
String word2 = "lo";
```

Example 11:

```
int len = word1.length();                  // len would equal 5
```

Example 12:

```
String piece = word1.substring(3);         // piece would equal "lo"
```

Example 13:

```
String piece = word1.substring(1,4);       // piece would equal "ell"
```

Example 14:

```
int firstIndex = word1.indexOf(word2);     //firstIndex would equal 3
```

Out of Bounds Exception

The most common error when working with Strings is
`StringIndexOutOfBoundsException`
This happens when you tell the code to use an index that doesn't exist.

Example 15:

```
String word1 = "Hi";
String newWord1 = word1.substring(0,4);
// StringIndexOutOfBoundsException error
```

Recall, in this case, that Java will try to build a String starting at index 0 and ending at index 3. However index 3 doesn't exist in `word1`.