**Java Example Solution Code**

This page shows Java solution code for some common problem types

* [If-Boolean Logic](http://codingbat.com/doc/java-example-solution-code.html#logic)
* [Strings](http://codingbat.com/doc/java-example-solution-code.html#string)
* [Arrays](http://codingbat.com/doc/java-example-solution-code.html#array)
* [Recursion](http://codingbat.com/doc/java-example-solution-code.html#recursion)
* [must return type X](http://codingbat.com/doc/java-example-solution-code.html#mustreturn) error

The examples are geared to help with the [CodingBat java coding problems](http://codingbat.com/java). See the [Code Help+Videos](http://codingbat.com/doc/code-help-videos.html) page for a complete list of code help.

**If-Boolean Logic**

In this example, the aIsBigger() method should return true if the int parameter **a** is larger than**b** by 2 or more. This code uses an if with && ("and") to return true if the condition is met. Alternately, the run falls through to the return-false at the bottom. This is a pretty simple way to do it.

public boolean aIsBigger(int a, int b) {

if (a > b && (a - b) >= 2) {

return true;

}

return false;

}

Alternately it can be done with an if/else structure like this:

public boolean aIsBigger(int a, int b) {

if (a > b && (a - b) >= 2) {

return true;

} else {

return false;

}

}

And in fact, since the boolean test is true when we want to return true, and false when we want to return false, it can be written as a one-liner like this:

public boolean aIsBigger(int a, int b) {

return (a > b && (a - b) >= 2);

}

See [Java If Boolean Logic](http://codingbat.com/doc/java-if-boolean-logic.html) for more information

**Strings**

Make a string out of text by enclosing it in double quotes "like this", and use + to combine strings to make bigger strings. For this example, the withNo() method takes in a string and returns a new string with "No:" added at the front.

public String withNo(String str) {

return "No:" + str;

}

With a string, str.substring(i, j) returns the String that starts at index i and goes up to but not including j. The first char in the String is at index 0, so str.substring(0, 2) returns a string of the first two chars. The method str.length() returns the length of a string. Compare two strings like this: str1.equals(str2). Do not compare two strings with == which looks reasonable but does not work correctly in all cases.

This twoE() example method returns true if the string contains exactly two 'e' chars. The code:  
"for (int i=0; i<str.length(); i++) { ..." is very standard code to look at each position in a String.

public boolean twoE(String str) {

int count = 0;

for (int i=0; i<str.length(); i++) {

String sub = str.substring(i, i+1);

if (sub.equals("e")) count++;

}

if (count == 2) return true;

return false;

// last 2 lines can be written simply as "return (count == 2);"

}

The "char" type in Java represents a single character and is written in single quotes like this: 'e'. Here's a version of the twoE() method which uses str.charAt(i) to access each char of a string. Use == to compare chars.

public boolean twoE(String str) {

int count = 0;

for (int i=0; i<str.length(); i++) {

if (str.charAt(i) == 'e') count++;

}

if (count == 2) return true;

return false;

// this last if/else can be written simply as "return (count == 2);"

}

See [Java String Introduction](http://codingbat.com/doc/java-string-introduction.html) for more information.

**Arrays**

This pair13() example method returns true if the int array contains a pair of 13's next to each other.

public boolean pair13(int[] nums) {

int count = 0;

for (int i=0; i<(nums.length-1); i++) {

if (nums[i]==13 && nums[i+1]==13) return true;

}

return false; // if we get here, there was not a pair of 13's

// note: the i loop stops one short of the full length,

// so the code can refer to nums[i+1] in the loop.

}

This new6() method makes and returns a new int array of size N that is filled with the value 6.

public int[] new6(int n) {

int[] result = new int[n];

for (int i=0; i<result.length; i++) {

result[i] = 6;

}

return result;

}

See [Java Array Loops](http://codingbat.com/doc/java-array-loops.html) for more information.

**Recursion**

Recursive code begins with a **base case** if-statement which checks for one or more cases that are so simple, that the answer can be returned immediately. That is followed by a **recursive**case which calls the same method with slightly smaller inputs, and then fixes up what it returns. "Smaller" inputs means at least one step towards the base case.

Here is a simple recursive method that counts the number of "A" in the given string.

public int countA(String str) {

// Base case -- return simple answer

if (str.length() == 0) {

return 0;

}

// Deal with the very front of the string (index 0) -- count "A" there.

int count = 0;

if (str.substring(0, 1).equals("A")) {

count = 1;

}

// Make a recursive call to deal with the rest of string (the part beyond the front).

// Add count to whatever the recursive call returns to make the final answer.

// Note that str.substring(1) starts with char 1 and goes to the

// end of the string.

return count + countA(str.substring(1));

}

**Must Return Type X**

**This method must return a result of type XXX** -- this compile error results if it appears that the method can exit without calling **return**. For example, this code will not compile:

// This does not work

public boolean foo() {

if (something) {

return true;

} else if (something else) {

return false;

}

}

In this case the compiler gets confused: what happens if both if statements are false? The simplest correct form has a last catch-all return, so some value is always returned, like this:

// This works

public boolean foo() {

if (something) {

return true;

}

return false;

}

Or fill out the if/else structure so that every path has a return:

// This works

public boolean foo() {

if (something) {

return true;

}

else {

return false;

}

}