

CSE 11
Winter 2017
Program Assignment #2 (100 points)

START EARLY!

Due: 9 February 2017 at 1159pm (2359, Pacific Standard Time)

PROGRAM #2: DoubleArray11

READ THE ENTIRE ASSIGNMENT BEFORE STARTING

In lecture, we described several common operations that can be performed on arrays. In this assignment you will implement a class, called `DoubleArray11`. This class defines and implements these and other operations on double arrays.

The goals of this assignment are:

1. to become comfortable with defining a class and using it
2. become more comfortable with reading javadoc-created documentation and using it to guide implementation
3. developing code to test your implementation as you write it

This is a much more complicated program than #1, so it will require you to start to develop some of your own debugging skills. You will also modify a program called `DoubleArrayMain` to both write tests for your class and interactively manipulate instances of the `DoubleArray11` class.

TASK 1 : The DoubleArray11 class

You are being provided a skeleton of the `DoubleArray11` class. It has been commented using Javadoc-style comments.

1. Download `DoubleArray11.java`
2. Generate the documentation using the javadoc command-line tool, to see the details of each constructor and public method. See (A-section) [lecture slides](#) for an example of how to run the javadoc command.
3. Go through the signature of all the methods defined in the Javadoc and carefully read method description to understand its operation on the internal array
4. Implement all methods correctly.
5. Put your name in the file after the existing Javadoc `@author` tag (replace YOUR NAME HERE)

When you Implement `DoubleArray11`, **You may NOT:**

- a) Change the signature of any `public` method
- b) Change the return type of any `public` method

- c) Add any new public methods. You may add as many private methods as you desire
- d) Add any public instance or class variables. All class/instance variables should be declared private
- e) Use any other pre-defined classes (e.g., Arrays, ArrayList,...) that provide similar functionality. Your implementation of DoubleArray11 should not have any other import statements than what has been provided to you in the skeleton

You are allowed to add any number of your own **private** methods and variables.

As you look at the DoubleArray11 definition, it should be clear that:

- a. **An instance of the class** internally stores an **array of doubles**. This instance variable makes up a key component of the state of an DoubleArray11 object.
- b. The class must also **define methods that manipulate an instance's internally-stored array**.
- c. For this assignment, the capacity of the internal array may be larger than the number of active elements. The first size elements of the internal array are considered active.
- d. You should also note, that in the skeleton, boolean methods **always return false**. That is **clearly incorrect** in the full implementation. Those return values in the skeleton are present so that the initial code will compile.
- e. Note the return types of the methods.

For clarity, we will list the constructors and methods defined in DoubleArray11 with a brief description. In grading your assignment, we will write our test programs to utilize only your implemented DoubleArray11 class to test its functionality.

The following table illustrates a portion of the Javadoc that was generated after [Step 2](#):

Modifier and Type	Method and Description
boolean	append(double element) Append element at the "end" of the array.
boolean	delete(int index) Delete an element at index.
double[]	getArray() Returns a copy of the internal stored array.
int	getCapacity() Get the capacity of the internally stored array.
java.lang.Double	getElement(int index) Get the Element at index.
static int	getObjects() Determine how many DoubleArray11 Objects have been constructed.
int	getSize() Get the number of elements actually stored in the internal array e.g.
boolean	insert(int index, double element) Insert an element at index in the array.

From the table we can see that **methods that return boolean should return true if method was successful, false if an error would occur**. For example, if you attempted to insert at improper index is out of bounds, you should return false.

Note : Your DoubleArray11 implementation should never generate a runtime

error. It must not print any error messages to the screen (though you will find such error messages useful during debugging). That means that you will need to check your arguments for valid values. Be sure to check that object references passed to your methods are not null.

In the case of a constructor, if passed a null object pointer, you should create a valid instance in which the internal array has no capacity to store elements.

Some hints for solving Task 1:

1. You may find an internal helper method that copies the contents of one array into another to be useful.
2. If the `insert` method is invoked, and your internal array is full, you should create a new internal array with expanded capacity. Don't forget to copy the original elements to your new internal array.

TASK 2 : The DoubleArray11Main Program

This is a java program and therefore defines a `main()` method. You are being provided a `DoubleArrayMain` class that allows you to utilize/test instances of `DoubleArray11`. You will extend the `DoubleArrayMain` class to allow testing of all of the methods in `DoubleArray11`.

You are being provided a partially-completed `DoubleArrayMain` program. Don't forget to put your name in the file after the existing Javadoc `@author` tag (replace YOUR NAME HERE)

The main method contains a loop that processes *commands* that create and use methods of the `DoubleArray11` class. Each command is a string input by the user, which is then interpreted by the loop, and causes some action to happen.

The start of the interpreter sets up some initial variables, and the first thing the loop does is print a `>` character and wait for input:

```
DoubleArray11 currentArray = new DoubleArray11();
Scanner scan = new Scanner(System.in);
String cmd = "";

while(true) {
    System.err.print("> ");
    cmd = scan.nextLine();
    String[] parts = cmd.split("\\s+");
```

The input is stored in `cmd`, and then it is split apart on whitespace. The expression `cmd.split("\\s+")` returns an array of strings, where each string returned is a fragment of `cmd` separated by whitespace from the others. So, for example

```
"set 10 1.7".split("\\s+")
```

would produce the array `{"set", "10", "1.7"}`.

The rest of the loop processes the array produced:

```
if(parts.length == 0) { continue; }
switch(parts[0]) {
    case "show":
        System.out.println(currentArray.toString());
        continue;
    case "create":
        if(parts.length != 2 || !isInt(parts[1])) {
            printCreateUsage();
            continue;
        }
        currentArray = new DoubleArray11(Integer.parseInt(parts[1]));
        continue;
}
```

In particular, it matches the string in the first position in the array to see what command to run. In this snippet, we see that if the array starts with “show,” the current array will be printed (using its `toString` method). If the array starts with “create,” the interpreter expects the array to have a second element (index 1), which is an integer. If it doesn’t, it prints an error message and goes back to the top of the loop. If it *does*, it uses that as an argument to the constructor for `DoubleArray11`. This will have the effect of updating the `currentArray` for future iterations of the loop.

You should study this loop in detail so you understand how it works, and ask questions about it on Piazza and in lab to help.

You will use these commands in two ways. First, you will extend the set of commands to cover all the methods on `DoubleArray11`. Second, you will use the commands to *test* your implementation and make sure it does what you expect.

Adding Commands

There are several commands already implemented that call methods, print values, and otherwise manipulate and inspect the array. You will add several more commands, using the examples given to develop the code that calls out to a method.

These commands should not print anything, just have the indicated effect:

- `delete n` -- deletes the item at index `n` in the current array using the `delete` method

- `insert n d` -- adds the element `d` at index `n` by using the `insert` method
- `reverse-from start stop` -- uses the `reverse` method to reverse the array from `start` to `stop` (which should be valid indices)
- `reverse` -- uses the `reverse` method to reverse the whole array
- `swap n m` -- uses the `swap` method to switch the values in `n` and `m`

These commands should print their answer:

- `min` -- uses the `min` method to print the min
- `max` -- uses the `max` method to print the max
- `size` -- uses the `getSize` method to print the current size of the array

Using Commands to Test

Running the program `DoubleArrayMain` after compiling it will allow you to quickly test particular methods. For example, you might have an interaction like this:

```
$ java DoubleArrayMain
> show
[]
> create 5
> append 2.3
> show
[ 2.30]
> append 4.4
> show
[ 2.30, 4.40]
```

This would cause the `toString()` method to be called 3 times (once for each use of `show`), the constructor to be called once (for `create`), and `append` to be called twice. You could be confident in the basic behavior of `append` and the constructor after doing this, though there may be more cases to consider.

It's useful, once you've written a good test, to have a way to verify that it doesn't break later. One way you can do this is by using a feature of the *shell* (the thing running in your terminal), to save a test and then run it later. You can create a file with one command per line, ending in `exit`, for example the test above would be written as:

```
show
create 5
append 2.3
show
append 4.4
show
exit
```

Then you can use the `<` shell operator to send that file to a program instead of user input (assuming the file was saved in `test.txt`):

```
$ java DoubleArrayMain < test.txt
```

This has the effect of running `DoubleArrayMain`, while treating the lines in `test.txt` *as though you had typed them yourself*. Each use of `scan.nextLine()` will read a full line of the input file and process it as user input. This makes it so you can save the above test in a file, and re-run it easily without having to retype all the commands.

It's *also* useful to be able to run a number of tests at once, if you've saved several different files. We've given you some support for doing this with the script `run_tests`. If you run the command:

```
$ ./run_tests
```

The script will look for pairs of files named `testinput1.txt`, `testoutput1.txt`, `testinput2.txt`, `testoutput2.txt`, and so on. **It will run the main program on the inputs in the files named `testinput`, and compare the output to the corresponding `testoutput` file.** That is, `DoubleArrayMain` when reading `testinput` is expected to give output identical to the `testoutput` file. The `run_tests` program will report a success if they match, and a test error if they don't. For example, success would look like:

```
$ ./run_tests
Success for test tests/testinput1.txt
1 tests passed, 0 failed
```

while failure would look like:

```
$ ./run_tests
Test tests/testinput1.txt failed, expected:
[ 2.30]
[ 2.30, 4.40]

but got:
[]
[]

0 tests passed, 1 failed
```

We've provided one test to get you started, and starting out, this kind of failure is what you'll see.

You are required to submit test files along with your submission, and you will be graded on whether or not your tests are correct and thorough. Your tests, when taken together,

should cover every command at least once in a meaningful way. “Meaningful” means, for example, that you if you want to test `append`, you should use it and then later use `show` to make sure that the element is present. You might use `append` without `show` to check that e.g. `size` works correctly, but you should also include a meaningful test for `append`. You should also check for error conditions – for example, test for `getElement` on an empty array, or for inserting at an invalid index.

Submit exactly 6 different test files (`testinput1.txt` through `testinput6.txt`) with corresponding expected outputs (`testoutput1.txt` through `testoutput6.txt`), and among those files have at least one meaningful test for each command. In order to do so, you will actually use each command many times.

Note: Testing also helps you make sure that your program is correct, and is a huge aid in debugging. We recommend that you run `./run_tests` quite frequently when developing -- running it after each time you compile your program is not too often. Once you get a small number of tests passing, it becomes easy and fun to build on them and see how robust your implementation is, and you get the added benefit of being able to tell if anything breaks later when you go back to make other edits.

To emphasize: The goal of this part of the assignment is for you to practice writing code that helps you in debugging. A good test should have a known outcome that you can verify.

Some hints for solving Task 1 and Task 2:

1. DO NOT ATTEMPT TO CODE EVERY METHOD BEFORE YOU START DEBUGGING. BUILD IN STAGES
2. Build your `DoubleArray11` class in stages. (yes, I am repeating myself on purpose) Define a method, then define a way to test for the method. Once you are satisfied that the method you have just coded is correct, move onto the next one.
3. Make sure that you handle edge conditions (border cases) correctly. For example, insert at index `k` when `k == length` of the array.
4. Utilize private methods as you need them. An internal (helper) method to copy the contents of one array to another might be useful
5. Do NOT assume that arrays of only a certain size will be constructed.
6. Properly indent all code code, but do not be overly concerned with comments. A line or two commenting what a test is trying to do is helpful.
7. If you write a test and it works the first time, change your `DoubleArray11` code in some small way to so that it should NOT pass the test. The idea is to validate that your test is doing what you think it does. Don't forget to return your `DoubleArray11` code to the proper state!

Make Copies of your Program Files as you go along, If you make a big mistake you can go back to the previously working code

TASK 3 : Turning in your Program

YOU MUST BE LOGGED ON TO IENG6.UCSD.EDU FOR THIS TO WORK. PLEASE VERIFY WELL BEFORE THE DEADLINE THAT YOU CAN TURNIN FILES

You will be using the “bundlePR2” program that will turn in the files

DoubleArray11Main.java
DoubleArray11.java

tests/testinput1.txt
tests/testinput2.txt
tests/testinput3.txt
tests/testinput4.txt
tests/testinput5.txt
tests/testinput6.txt

tests/testoutput1.txt
tests/testoutput2.txt
tests/testoutput3.txt
tests/testoutput4.txt
tests/testoutput5.txt
tests/testoutput6.txt

Note that you can have *more* than 6 test files in the directory, but only these files will be turned in and graded. No other files will be turned in and they **must be named exactly as above**. BundlePR2 uses the department’s standard turnin program underneath.

To turn-in your program, you must be in the directory that has your source code and then you execute the following

\$ **bundlePR2**

The output of the turnin should be similar to what you saw in your first programming assignment. We will announce when the bundlePR2 command is available.

You can turn in your program multiple times. The turnin program will ask you if you want to overwrite a previously-turned in project. **ONLY THE LAST TURNIN IS USED!**

Don't forget to turn in your best version of the assignment.

Frequently asked questions

Q. Can I add extra output? No. We attempt to autograde your programs to the extent possible. Having extra output can cause you to lose points.

Q. What if my programs don't compile? Can I get partial credit? No. The bundle program will not allow you to turn in a program that does not compile.

Q. I know about ArrayList, can I use it? Absolutely NOT! That is a key part of the assignment. You may not import any other java classes in your code.

Q. I know about other classes that easily provide similar functionality, can I use them? No. one of the purposes of this assignment is write and debug some common array methods using only the language's native array support.

Q. Can my DoubleArray11 class extend an existing class (using the extends keyword)? No.

Q. Will you grade program style? Yes. In particular, indentation should be proper, variable names should be sensible. We will also look for code clarity, too. Overly long or complex codes are frowned upon.

Q. I don't understand how you will grade DoubleArray11Main? The readers will inspect your code to see if looks like it does meaningful tests. We'll make sure that your code passes your tests. In other words, your testing code has to run properly on your code.

Q. I don't understand how you will grade DoubleArray11? We will run our test program against your DoubleArray11 class (autograde) and the readers will inspect your code for clarity. It is essential that you do NOT change the signatures of public methods or their return values – otherwise the autograder will fail and you will not receive points.

Q. Can I add other commands in DoubleArray11Main.java? Yes.

Q. Is there a maximum size array that can be stored in my DoubleArray11 class? The only practical limit is the size of memory. Do not hard code a maximum sized array that your class can support. Instead, your code will extend the capacity of the internal storage array as needed.

Q. My instance variable that stores the array has N slots, and when I want to append

(so that I store $N+1$) elements, there is no room. What should I do? You must create a new array of sufficient size to hold at least $N+1$ elements. You must copy the original array elements to this new array and then store the new element in the correct location. This is the state variable that is internal to your class definition. You are NOT creating a new DoubleArray11 instance.

Q. When I delete an element, should I make a new internal array? There is no reason to.

Q. When I insert an element at location k , what happens to the existing elements? See [lecture notes](#). But you have to make room in your internal array to store the new data by moving the existing elements over by one index. The number of stored elements will increase by one. If there isn't room in your internal array, you have to make a new one and copy the data, just like the append case.

Q. Is my DoubleArray11.java supposed to have a main method? No.

START EARLY! ASK QUESTIONS

