Com S 227
Spring 2017
Assignment 1
80 points

Due Date: Thursday, February 2, 11:59 pm (midnight)
"Late" deadline (25% penalty): Friday, February 3, 11:59 pm

General information

This assignment is to be done on your own. See the Academic Dishonesty policy in the syllabus, http://www.cs.iastate.edu/~cs227/syllabus.html **, for details.**

You will not be able to submit your work unless you have completed the *Academic Dishonesty policy acknowledgement* on the Homework page on Blackboard. Please do this right away.

If you need help, see your instructor or one of the TAs. Lots of help is also available through the Piazza discussions.

Please start the assignment as soon as possible and get your questions answered right away. It is physically impossible for the staff to provide individual help to everyone the night that the assignment is due!

This is a "regular" assignment so we are going to read your code. Your score will be based partly (about a third) on the specchecker's functional tests and partly on the grader's assessment of the quality of your code. See the "More about grading" section.

Tips from the experts: How to waste a lot of time on this assignment

1. Start the assignment the day it's due. That way, if you have questions, the TAs will be too busy to help you and you can blame the staff when you get a bad grade.
2. Don't bother reading the rest of this document, or even the specification, especially the "Getting started" section. Documentation is for losers.
3. Don't test your code. It's such fun to remain in suspense until you get your score!
4. The main guiding principle is: *Try to write all the code before you figure out what it's supposed to do.*

Overview

The purpose of this assignment is to give you some practice with the process of implementing a class from a specification and testing whether your implementation conforms to the specification.

For this assignment you will implement one class, called **`Television`**, that models some aspects of the behavior of simple television. A Television has a current *channel* and *volume.* The volume level ranges from 0.0 to 1.0. Channels range from zero through *channelMax* - 1, where *channelMax* is a configurable value provided in the constructor. The volume is adjusted with methods **`volumeUp`** and **`volumeDown`**, but never goes above 1.0 or below 0.0. The volume is incremented or decremented by the value of the constant **`VOLUME_INCREMENT`**. The channel is adjusted with methods **`channelDown`** and **`channelUp`**, but never goes below zero or above *channelMax* - 1. The channel can also be set directly with the **`setChannel`** method. The method **`goToPreviousChannel`** sets the channel to most recent previous channel number. There is also a method **`resetChannelMax`** that "reprograms" the Television to have a different range of channels.

Specification

The specification for this asssignment includes this pdf along with any "official" clarifications posted on Piazza.

There is one public constant:

```
public static final double VOLUME_INCREMENT = 0.07;
```

There is one public constructor:

`public Television(int givenChannelMax)`
Constructs a new Television with channels 0 through **`givenChannelMax`** - 1. Iniially the volume is 0.5.

There are the following public methods:

`public void channelDown()`
Changes the channel down by 1, wrapping around to *channelMax* - 1 if the current channel is zero.

`public void channelUp()`
Changes the channel up by 1, wrapping around to zero if the current channel is *channelMax* - 1.

`public String display()`

Returns a string representing the current channel and volume in the form "**`Channel`** *x* **`Volume`** *y*`%`", where *x* is the current channel and *y* is the volume, multiplied by 100 and rounded to the nearest integer. For example, if the channel is 8 and the volume is .765, this method returns the exact string "**`Channel 8 Volume 77%`**".

`public int getChannel()`

Returns the current channel.

`public double getVolume()`

Returns the current volume.

`public void goToPreviousChannel()`

Sets the current channel to the most recent previous channel. If no channel has ever been set for this **`Television`** using one of the methods **`channelDown`**, **`channelUp`**, or **`setChannel`**, this method sets the channel is 0.

`public void resetChannelMax(int givenMax)`

Resets this **`Television`** so that its available channels are now from 0 through **`givenMax`** - 1. If the current channel is greater than **`givenMax`** - 1, it is automatically adjusted to be **`givenMax`** - 1. Likewise, if the previous channel is greater than **`givenMax`** - 1, it is automatically adjusted to be **`givenMax`** - 1.

`public void setChannel(int channelNumber)`

Sets the channel to the given channel number. If the given value is greater than *channelMax* - 1, the channel is set to *channelMax* - 1. If the given value is negative, the channel is set to zero.

`public void volumeDown()`

Lowers the volume by **`VOLUME_INCREMENT`**, but not below 0.0.

`public void volumeUp()`

Raises the volume by **`VOLUME_INCREMENT`**, but not above 1.0.

Where's the main() method??

There isn't one! Like most Java classes, this isn't a complete program and you can't "run" it by itself. It's just a single class, that is, the definition for a type of object that might be part of a larger system. To try out your class, you can write a test class with a main method such the example below.

Sample usage

A good way to think about the specification is to try to write some simple test cases and think about what behavior you expect to see. Here are a few examples:

```
public class TelevisionTest
{
  public static void main(String[] args)
  {
    Television tv = new Television(5);

    // try out the volume controls
    System.out.println(tv.getVolume());  // expected 0.5
    tv.volumeUp();
    tv.volumeUp();
    System.out.println(tv.getVolume());  // expected 0.64
    tv.volumeUp();
    tv.volumeUp();
    tv.volumeUp();
    tv.volumeUp();
    tv.volumeUp();
    tv.volumeUp();
    System.out.println(tv.getVolume());  // expected 1.0

    // setting channels
    System.out.println(tv.getChannel());  // expected 0
    tv.setChannel(4);
    System.out.println(tv.getChannel());  // expected 4
    tv.channelUp();
    tv.channelUp();
    System.out.println(tv.getChannel());  // expected 1
    tv.channelDown();
    System.out.println(tv.getChannel());  // expected 0
  }
}
```

There is also a specchecker (see below) that will perform a lot of functional tests, but when you are developing and debugging your code at first you'll always want to have some simple test cases of your own as in the `main` method above.

Suggestions for getting started

*Smart developers don't try to write all the code and then try to find dozens of errors all at once; they work **incrementally** and test every new feature as it's written. Here is a rough guide for how an experienced coder might go about creating a class such as this one:*

0. Be sure understand the basics of defining a class as in Sections 3.1 - 3.3 of the text and as practiced in Lab 2.

1. Create a new, empty project and add a package called **`hw1`**.

2. Create the **`Television`** class in the **`hw1`** package and put in stubs for all the required methods and the constructor. (For methods that are required to return a value, just put in a "dummy" **`return`** statement that returns zero or false.)

3. Download the specchecker, import it into your project as you did in labs 1 and 2, and run it. There will be lots of error messages appearing in the console output, since you haven't actually implemented the methods yet. *Always start reading from the top.* All you really want to check at this point is whether you have a missing or extra public method, if the method declarations are incorrect, or if something is really wrong like the class having the incorrect name or package. **Any such errors will appear *first* in the output and will usually say "Class does not conform to specification."**

4. Look at each method. Mentally classify it as either an *accessor* (returns some information without modifying the object) or a *mutator* (modifies the object, usually returning **`void`**). The accessors will give you a lot of hints about what instance variables you need.

5. Before you write code for a method, always write a simple usage example or test case, similar to the **`main`** method shown above. This will make sure you understand what the code is really supposed to do, and it will give later you a way to check whether you did it correctly. Of course, if you are really *not* sure what a method is supposed to do, bring up your question for discussion on Piazza!

6. You might start with getting and setting channel numbers. Look at **`getChannel()`**. It's an accessor that returns the current channel. That information has to be stored somehow within the **`Television`** object, which tells you that you probably need an instance variable to represent the channel. Then **`setChannel()`** is also simple to write. Note there is a requirement that if someone tries to set a negative channel, it gets set to channel 0. You can do this easily using **`Math.max`** to select the larger of two values, for example

```
int newChannel = Math.max(0, givenChannel);
```

Likewise, you can use **`Math.min`** to handle the possibility of a given channel that is too large. The largest possible channel number is *channelMax* - 1, where *channelMax* is the value originally given in the constructor. Notice that in order for this piece of information to be available in the **`setChannel`** method, it has to be stored in an instance variable by the constructor.

7. For `channelUp()` you're just adding 1, but after adding, take the remainder mod *channelMax* to make it wrap around to zero. (Try it!) `channelDown()` is similar, but with a twist: you can't just subtract 1 and take the result mod *channelMax*. For example, suppose *channelMax* is 5. If you subtract 1 from channel 0, you want to be at channel 4. But in Java, `(-1 % 5)` gives you -1, not 4. If you also add *channelMax* (5 in this example) *every* time you subtract 1, then taking the remainder will always give the right answer. (Try it!)

8. You can implement and test the volume controls independent of the channels. Again, you'll need an instance variable to store the current volume. To keep the volume from going over 1.0, or below 0.0, you can use the methods `Math.min` and `Math.max`, respectively (see the note in the next section).

Additional notes

1. You do NOT need conditional statements ("`if`" statements) or loops for this assignment. We will start covering conditional statements later next week, and you won't be penalized if you use them, but you would just be making things more complicated. (If you have done some programming before and are tempted to use a bunch of if-statements, make it a challenge for yourself to write simpler code without them.)

a) If you need to find the smaller or larger of two numbers (e.g. to keep the volume from going over 1.0), just use the methods `Math.min()` or `Math.max()`. As an example,

```
int x = Math.min(3, 4); // now x is 3
```

b) To round to the nearest integer, use `Math.round()` *Note:* one little "gotcha" with `Math.round` is that it returns a type of integer value called a `long`, so you generally have to "cast" it to convert to the `int` type to use it in your code. It would look like this:

```
int x = (int) Math.round(y);
```

c) To wrap around the station numbers, you can use the modulus operator (`%`). (See the Getting Started section above for more detail.)

2. Do not add any additional public methods. (You can add your own methods if you feel compelled to do so, but they must be declared `private`.) Do not create any additional Java classes.

The SpecChecker

You can find the SpecChecker online; see the Piazza Homework post for the link. Import and run the SpecChecker just as you practiced in Labs 1 and 2. It will run a number of functional tests and then bring up a dialog offering to create a zip file to submit. Remember that error messages will appear in the *console* output. There are many test cases so there may be an overwhelming number of error messages. ***Always start reading the errors at the top and make incremental corrections in the code to fix them.*** When you are happy with your results, click "Yes" at the dialog to create the zip file. See the document "SpecChecker HOWTO", which can be found in the Piazza pinned messages under "Syllabus, office hours, useful links," if you are not sure what to do.

More about grading

This is a "regular" assignment so we are going to read your code. Your score will be based partly (about a third) on the specchecker's functional tests and partly on the TA's assessment of the quality of your code. This means you can get partial credit even if you have errors, and it also means that even if you pass all the specchecker tests you can still lose points. Are you doing things in a simple and direct way that makes sense? Are you defining redundant instance variables? Some specific criteria that are important for this assignment are:

- Use instance variables only for the "permanent" state of the object, use local variables for temporary calculations within methods.
    - You will lose points for having lots of unnecessary instance variables
    - All instance variables should be **`private`**.
- **Accessor methods should not modify instance variables**.

See the "Style and documentation" section below for additional guidelines.

Style and documentation

Roughly 15% of the points will be for documentation and code style. Here are some general requirements and guidelines:

- Each class, method, constructor and instance variable, whether public or private, must have a meaningful and complete Javadoc comment. Class javadoc must include the **`@author`** tag, and method javadoc must include **`@param`** and **`@return`** tags as appropriate.
    - Try to state what each method does in your own words, but there is no rule against copying and pasting the descriptions from this document.
    - Run the javadoc tool and see what your documentation looks like! You do not have to turn in the generated html, but at least it provides some satisfaction :)

- All variable names must be meaningful (i.e., named for the value they store).
- Your code should not be producing console output. You may add **`println`** statements when debugging, but you need to remove them before submitting the code.
- Use the defined constant **`VOLUME_INCREMENT`**. Don't embed the number 0.07 in your code.
- Internal (//-style) comments are normally used inside of method bodies to explain *how* something works, while the Javadoc comments explain *what* a method does. (A good rule of thumb is: if you had to think for a few minutes to figure out how something works, you should probably include an interal comment explaining how it works.)
    - Internal comments always *precede* the code they describe and are indented to the same level. In a simple homework like this one, as long as your code is straightforward and you use meaningful variable names, your code will probably not need any internal comments.
- Use a consistent style for indentation and formatting.
    - Note that you can set up Eclipse with the formatting style you prefer and then use Ctrl-Shift-F to format your code. To play with the formatting preferences, go to Window->Preferences->Java->Code Style->Formatter and click the New button to create your own "profile" for formatting.

If you have questions

For questions, please see the Piazza Q & A pages and click on the folder **`assignment1`**. If you don't find your question answered, then create a new post with your question. Try to state the question or topic clearly in the title of your post, and attach the tag **`assignment1`**. *But remember, do not post any source code for the classes that are to be turned in.* It is fine to post source code for general Java examples that are not being turned in. (In the Piazza editor, use the button labeled "pre" to have Java code formatted the way you typed it.)

If you have a question that absolutely cannot be asked without showing part of your source code, make the post "private" so that only the instructors and TAs can see it. Be sure you have stated a specific question; vague requests of the form "read all my code and tell me what's wrong with it" will generally be ignored.

Of course, the instructors and TAs are always available to help you. See the Office Hours section of the syllabus to find a time that is convenient for you. We do our best to answer every question carefully, short of actually writing your code for you, but it would be unfair for the staff to fully review your assignment in detail before it is turned in.

Any posts from the instructors on Piazza that are labeled “Official Clarification” are considered to be part of the spec, and you may lose points if you ignore them. Such posts will always be placed in the Announcements section of the course page in addition to the Q&A page. (We promise that no official clarifications will be posted within 24 hours of the due date.)

What to turn in

Note: You will need to complete the "Academic Dishonesty policy questionnaire," found on the Homework page on Blackboard, before the submission link will be visible to you.

Please submit, on Blackboard, the zip file that is created by the SpecChecker. The file will be named **`SUBMIT_THIS_hw1.zip`**. and it will be located in whatever directory you selected when you ran the SpecChecker. It should contain one directory, **`hw1`**, which in turn contains one file, **`Television.java`**. Please LOOK at the file you upload and make sure it is the right one!

Submit the zip file to Blackboard using the Assignment 1 submission link and verify that your submission was successful by checking your submission history page. If you are not sure how to do this, see the document "Assignment Submission HOWTO" which can be found in the Piazza pinned messages under “Syllabus, office hours, useful links.”

> We recommend that you submit the zip file as created by the specchecker. If necessary for some reason, you can create a zip file yourself. The zip file must contain the directory **hw1**, which in turn should contain the file **Television.java**. You can accomplish this by zipping up the **src** directory of your project. The file must be a zip file, so be sure you are using the Windows or Mac zip utility, and NOT a third-party installation of WinRAR, 7-zip, or Winzip.