Com S 227
Spring 2021
Assignment 1
100 points

Due Date: Friday, February 19, 11:59 pm (midnight)

5% bonus for submitting 1 day early (by 11:59 pm Feb 18)

10% penalty for submitting 1 day late (by 11:59 pm Feb 20)

No submissions accepted after February 20, 11:59 pm

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 questionnaire* on the Assignments page on Canvas. 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 afternoon 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 night it's due. That way, if you have questions, the TAs will be too busy to help you and you can spend the time tearing your hair out over some trivial detail.
2. Don't bother reading the rest of this document, or even the specification, especially not the "Getting started" section. Documentation is for losers. Try to write lots of code before you figure out what it's supposed to do.
3. Don't test your code. It's such fun to remain in suspense until it's graded!

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. You'll also get a lot of practice with modular arithmetic.

For this assignment you will implement one class, called **`TV`**, that models some aspects of the behavior of simple television. A TV has a current *channel* and *volume.* The volume level ranges from 0.0 to 1.0. The possible range of channels is determined by two parameters that we will refer to as *start* and *numChannels*. In general there will always be *numChannels* consecutive channels, the lowest of which is *start*; that is, the range of values is from *start* up to *start* + *numChannels* – *1*, inclusive. 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 **`channelUp`** and **`channelDown`**, but never goes below *start* or above *start* + *numChannels* – *1*. The channel can also be set directly with the **`setChannel`** method. The method **`goToPreviousChannel`** sets the channel to the most recent previous channel number. It is also possible to "reprogram" the TV to set a different start channel or number of channels.

Please note that you do not need any conditional statements (i.e. "if" statements) or anything else we haven't covered, for this assignment. There will be a couple of places where you need to choose the larger or smaller of two numbers, which can be done with the methods **`Math.min()`** or **`Math.min()`**. For the wrapping behavior of the channel numbers, you can use the mod (%) operator.

(You will not be directly penalized for using conditional statements, but your code will become longer and more complicated, and if it becomes overly complicated and hard to read, the TA will start taking off points.)

Specification

The specification for this asssignment includes this pdf along with any "official" clarifications announced on Canvas.

There is one public constant:

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

There is one public constructor:

```
public TV(int givenStart, int givenNumChannels)
```

Constructs a new TV with channels `givenStart` through `givenStart + givenNumChannels` – 1. Initially the volume is 0.5 and the channel is equal to `givenStart`.

There are the following public methods:

`public void channelDown()`

Changes the channel down by 1, wrapping around to *start* + *numChannels - 1* if the current channel is *start*.

`public void channelUp()`

Changes the channel up by 1, wrapping around to *start* if the current channel is *start* + *numChannels – 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 `TV` using one of the methods `channelDown`, `channelUp`, or `setChannel`, this method sets the channel to *start*.

`public void resetStart(int givenStart)`

Resets this `TV` so that its available channels are now from `givenStart` through `givenStart` + *numChannels* – 1. If the current channel or previous channel is outside the new range of channel numbers, their values are adjusted to be the nearest value within the new range.

`public void resetNumChannels(int givenNumChannels)`

Resets this `TV` so that its available channels are now from *start* through *start* + `givenNumChannels` – 1. If the current channel or previous channel is outside the new range of channel numbers, their values are adjusted to be the nearest value within the new range.

`public void setChannel(int channelNumber)`

Sets the channel to the given channel number. However, if the given value is greater than *start* + *numChannels – 1*, it is set to *start* + *numChannels – 1*, and if the given value is less than *start*, it is set to *start*.

`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.

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 getting started section.

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. Since this is our first assignment, here is an example of some incremental steps you could take in writing this class.*

0. Be sure you have done and understood Lab 2.

1. Create a new, empty project and then add a package called **`hw1`**. ***Be sure to choose "Don't Create" at the dialog that asks whether you want to create module-info.java***.

2. Create the **`TV`** class in the **`hw1`** package and put in stubs for all the required methods, the constructor, and the required constant. Remember that everything listed is declared **`public`**. For methods that are required to return a value, just put in a "dummy" return statement that returns zero. **There should be no compile errors**.

3. Briefly javadoc the class, constructor and methods. This is a required part of the assignment anyway, and doing it now will help clarify for you what each method is supposed to do before

you begin the actual implementation. (Copying from the method descriptions here or in the Javadoc is perfectly acceptable; however, ***DO NOT COPY AND PASTE from the pdf or online Javadoc!*** *This leads to insidious bugs caused by invisible characters that are sometimes generated by sophisticated document formats.)*

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. You might start with the volume operations `getVolume`, `volumeUp`, and `volumeDown`. Start with some sample usage, that is, a very simple test case, such as this one:

```
public class SimpleTests
{
  public static void main(String[] args)
  {
    TV tv = new TV(0, 5);
    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
  }
}
```

The presence of an accessor method `getVolume` suggests that you might need an instance variable to store the current volume. Remember that every time you define an instance variable, you should initialize it in the constructor. (*Tip: you can find these sample test cases in the file SimpleTests.java which is posted along with the homework spec.)* Notice that to keep the volume from going over 1.0, you can just use the method `Math.min`, something like this:

```
      myVolume = Math.min(myVolume, 1.0);
```

Write a similar test case for `volumeDown`.

6. Next you might try getting and setting channel numbers. Start with a simple usage example:

```
    tv = new TV(2, 10);
    System.out.println(tv.getChannel());  // expected 2
    tv.setChannel(8);
    System.out.println(tv.getChannel());  // expected 8
    tv.setChannel(42);
    System.out.println(tv.getChannel());  // expected 11
    tv.setChannel(1);
    System.out.println(tv.getChannel());  // expected 2
```

Having an accessor method `getChannel` suggests you need an instance variable storing your current channel number. For `setChannel`, notice that you can use `Math.max` and `Math.min` to keep the channel from being out of range. But that means the method needs to know the range, which implies that the constructor parameters for *start* and *numChannels* need to be stored in instance variables too.

7. Next you might think about `channelUp`. What is it supposed to do?

```
    tv = new TV(2, 10);
    tv.setChannel(8);
    tv.channelUp();
    System.out.println(tv.getChannel());  // expected 9
    tv.channelUp();
    System.out.println(tv.getChannel());  // expected 10
    tv.channelUp();
    System.out.println(tv.getChannel());  // expected 11
    tv.channelUp();
    System.out.println(tv.getChannel());  // expected 2
```

Once the channel is incremented past the maximum channel number, it's supposed to "wrap" back around to the start. If you are familiar with conditional statements, you may be tempted to implement this using conditionals, but that's really not necessary. It can be done simply using modular arithmetic. Suppose you have ten values, 0 through 9. Try this:

```
public class TryOutModularArithmetic
{

  public static void main(String[] args)
  {
    int numValues = 10;
    int x = 7;
    x = (x + 1) % numValues;
    System.out.println(x);
    x = (x + 1) % numValues;
    System.out.println(x);
    x = (x + 1) % numValues;  // wraps around to zero
    System.out.println(x);
```

```
   }
}
```

The only difference between the example above, and the range of channels, is that your channels begin at the *start* value, not necessarily at zero. So just subtract *start* from the channel, do the increment, and add *start*. (Or, you could just store the channel number internally as *channel – start*, and add *start* in the method `getChannel`.)

8. For `channelDown` there is one tiny wrinkle, due to the fact that the Java % operator always gives a negative answer if one of the operands is negative. That is, `-1 % 10` gives us `-1`, not `9`, which is what we want. This is easy to work around by adding the number of channels each time you decrement, for example:

```
    x = 2;
    x = (x - 1 + numValues) % numValues;
    System.out.println(x);
    x = (x - 1 + numValues) % numValues;
    System.out.println(x);
    x = (x - 1 + numValues) % numValues;   // wraps around to 9
    System.out.println(x);
```

Write a short test case using a different range of channels and make sure you're getting the right answers.

9. For `resetNumChannels`, the interesting part is that you may have to update the current channel, as well as the previous channel, to make sure they are still in range. Start with a concrete test case to make this behavior clear:

```
    tv = new TV(2, 10);
    tv.setChannel(8);
    tv.resetNumChannels(5);
    System.out.println(tv.getChannel());  // should now be 6
```

Since the new range of channel values is only 2 through 6, the current channel value 8 is supposed to be reset to the nearest value that is within range, in this case 6. (This is easy using `Math.min`.)

10. The behavior is similar for `resetStart`. Here, the current channel value 8 is reset to the nearest value within the new range 10 through 19.

```
    tv = new TV(2, 10);
    tv.setChannel(8);
    tv.resetStart(10);
    System.out.println(tv.getChannel());  // should now be 10
```

14. At some point, download the SpecChecker, import it into your project as you did in lab 1 and run it. *Always start reading error messages from the top.* If you have a missing or extra public method, if the method names or 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." **Always fix these first.**

The SpecChecker

You can find the SpecChecker on Canvas. Import and run the SpecChecker just as you practiced in Lab 1. 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", link #10 on our Canvas front page, 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? Are you using a loop for something that can be done with integer division? 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 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 javadoc comment**. The javadoc for the class itself can be very brief, but must include the `@author` tag. The javadoc for methods must include `@param` and `@return` tags as appropriate.
  - Try to briefly state what each method does in your own words. However, there is no rule against copying the descriptions from the online documentation. *However: **do not literally copy and paste from this pdf** or the online Javadoc! This leads to all kinds of weird bugs due to the potential for sophisticated document formats like Word and pdf to contain invisible characters.*
  - 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.

- 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 internal 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 **`hw1`**. 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 **`hw1`**. *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 announcements from the instructors on Canvas 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 Canvas. (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 Assignments page on Canvas, before the submission link will be visible to you.

Please submit, on Canvas, 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, **`TV.java`**. Please LOOK at the file you upload and make sure it is the right one!

Submit the zip file to Canvas using the Assignment 1 submission link and verify that your submission was successful. If you are not sure how to do this, see the document "Assignment Submission HOWTO", link #9 on our Canvas front page.

> 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 files **TV.java**. You can accomplish this by zipping up the **src** directory of your project. **Do not zip up the entire 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.