Com S 227  
Spring 2018  
Assignment 1  
100 points

Due Date: Friday, September 14, 11:59 pm (midnight)  
“Late” deadline (25% penalty): Monday, September 17, 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 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 "Sample Usage" or "Getting started" sections. 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 is correct.

For this assignment you will implement one class, called Printer, that is a simple model of ink and paper usage in a printer. It is slightly more complex than the Atom or Basketball classes that you saw in Lab 2, so be sure you have done and understood Lab 2.

The Printer class models the usage of paper and ink in a printer. Paper can be added to the printer at any time, up to a given maximum capacity (that is specified when the printer is constructed). A newly constructed printer has a full ink cartridge containing the quantity of ink given by the constant INK_CAPACITY. The printer can print one-sided or two-sided. For each side printed, a small quantity of ink is used, as given by the constant INK_USAGE. The ink cartridge can be replaced at any time, restoring the ink quantity to the value INK_CAPACITY.

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 few places where you need to choose the smaller of two numbers, which can be done with the method Math.min(). (You will not be penalized for using conditional statements instead, but your code will become longer and more complicated!)

Specification

Your Printer class must include and use the following constant definitions:

```java
/**
 * Capacity, in ounces, of a new ink cartridge.
 */
public static final double INK_CAPACITY = 2.0;

/**
 * Amount of ink, in ounces, used per printed page.
 */
public static final double INK_USAGE = 0.0023;
```

There are two public constructors and seven public methods:

```java
public Printer(int givenCapacity)
    Constructs a printer that has the given capacity (number of sheets of paper it can hold).
    Initially it contains no paper and a full ink cartridge.
```
public Printer(int givenCapacity, int givenNumberOfSheets)
   Constructs a printer that has the given capacity (number of sheets of paper it can hold).
   Initially it contains the given number of sheets of paper and a full ink cartridge. Note that
   if givenNumberOfSheets is greater than givenCapacity, then the printer will initially
   contain givenCapacity sheets of paper.

public void addPaper(int additionalSheets)
   Adds the given number of sheets of paper to this printer, not exceeding the printer's
   capacity.

public int getCurrentPaper()
   Returns the number of sheets of paper currently in this printer. This value is never
   negative.

public int getTotalPaperUse()
   Returns the total number of sheets of paper printed by this printer since its construction.
   Note this is counting sheets of paper, not pages printed, i.e. sheets used for two sided
   printing still count as just one sheet.

public boolean isInkOut()
   Determines whether the ink has run out. Returns true if the amount of ink left is smaller
   than the quantity INK_USAGE needed to print one page.

public void replaceInk()
   Simulates replacement of the ink cartridge, restoring the quantity of ink in the printer to
   INK_CAPACITY.

public void print(int numberOfPages)
   Simulates printing pages in one-sided mode, using the appropriate number of sheets and a
   corresponding quantity of ink. If there is not enough paper, the printer will use up all
   remaining paper and will only use the quantity of ink needed for the sheets actually
   printed. If there is not enough ink, the printer will use up all the ink, and will still use up
   the specified number of sheets of paper (i.e., it just prints a bunch of blank pages after the
   ink runs out).

public void printTwoSided(int numberOfPages)
   Simulates printing pages in two-sided mode, using the appropriate number of sheets and a
   corresponding quantity of ink. If there is not enough paper, the printer will use up all
   remaining paper and will only use the quantity of ink needed for the sheets actually
   printed. If there is not enough ink, the printer will use up all the ink, and will still use up
   the specified number of sheets of paper (i.e., it just prints a bunch of blank pages after the
   ink runs out).
Where's the main() method??

There isn't one. 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, analogous to AtomTest in Lab 2. See the "Sample usage" section below for some examples.

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 "Sample usage".

Sample usage

The best way to think about a specification like this is to try to write some simple test cases and think about what behavior you expect to see. Make up some sample values and work out what the results should be with pencil and paper. Once you figure something out by hand, you can usually write the code using the same sequence of steps. Here are some examples to get you started:

```java
import hw1.Printer;
public class SimpleTests {
    public static void main(String[] args) {
        Printer p = new Printer(100);

        // adding paper should increase paper in printer
        p.addPaper(10);
        p.addPaper(15);
        System.out.println(p.getCurrentPaper()); // expected 25
        System.out.println(p.isInkOut()); // expected false

        // printing should reduce the paper and increase the total
        p.print(5);
        System.out.println(p.getCurrentPaper()); // expected 20
        p.print(10);
        System.out.println(p.getCurrentPaper()); // expected 10
        System.out.println(p.getTotalPaperUse()); // expected 15

        // paper should not go below zero
        p.print(10);
        System.out.println(p.getCurrentPaper()); // expected 0
        System.out.println(p.getTotalPaperUse()); // expected 25

        // can't add more paper than the maximum
        p.addPaper(150);
        System.out.println(p.getCurrentPaper()); // expected 100
    }
}
```
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.

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

2. Create the Printer class in the hw1 package and put in stubs for all the required methods, the constructors, and the required constants. 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 or false. There should be no compile errors.

3. Javadoc the classes 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 method descriptions from this document is perfectly acceptable; don’t forget the @param and @return tags for completeness.)

4. You might start by thinking about paper usage, and forget about ink for a moment. Create some simple test cases for getCurrentPaper() and addPaper() like the ones shown in the "Sample usage" section above. (Usually you would not want your test code in the hw1 package, so make a separate package or use the default package.) What instance variables are needed to implement these methods? The method getCurrentPaper() should return the number of sheets of paper in the printer; therefore, that information has to be stored “permanently” as part of the printer’s state. Similarly, the capacity of the printer (number of sheets of paper it can hold) is a parameter in the constructor, but has to be saved as an instance variable so the value is available in other methods. For addPaper(), since there is a maximum capacity, you have to figure out how much paper is actually added. If a client tries to add 500 sheets when there is only space for...
200, then only 200 sheets are really added. You can use `Math.min()` to figure out which number is smaller.

5. At this point you might start on the `print()` method. Don't worry about the ink at first, just make sure the correct number of sheets are used and the total is updated correctly. Start by writing a few simple test cases. If the printer contains 20 sheets and a client tries to print 100 pages, what should `getTotalPaperUse()` return? Again, `Math.min()` is your friend here.

6. Implement the `printTwoSided()` method. Start with a couple of test cases. How many sheets of paper would be used to print 10 pages two-sided? How about 11 pages? 12 pages? 13 pages? You can use integer division and/or the modulus operator.

7. Finally, you'll need to deal with the ink usage. Start with a test case or two. How many pages can you print before you run out of ink? If you print one more, are you out of ink? If you refill the ink, can you print that many pages again? If you run out of paper, does it use the correct amount of ink?

8. At some point, download the SpecChecker, import it into your project as you did in labs 1 and 2, 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 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? 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 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 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 and pasting the descriptions from the online documentation.
  - 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.
  o 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 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, Printer.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" 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 files Printer.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.