# Com S 227
# Spring 2023
# Assignment 1
# 100 points

Due Date: Friday, February 10, 11:59 pm (midnight)
5% bonus for submitting 1 day early (by 11:59 pm Feb 9)
10% penalty for submitting 1 day late (by 11:59 pm Feb 11)
No submissions accepted after Feb 11, 11:59 pm

**This assignment is to be done on your own. See the Academic Integrity policy in the syllabus, 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, try office hours, the schedule is on Canvas. Lots of help is also available through the Piazza discussions. Please start the assignment as soon as possible to 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 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.

## Contents

## 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 practice using variables and modular arithmetic.

For this assignment you will implement one class, called **CameraBattery**, that models a removable and rechargeable camera battery. The battery can be charged both directly by the camera when connect to a USB port and by an external "wall wart" battery charger. However, the battery can only be in one place at a time, either connected to the camera, connected to the external charger, or disconnect for any device. The battery has a maximum capacity to which it can be charged. The rate of charge when connect to the camera is fixed, while the rate of charge when connected to the external charger is determined by the charger setting. The charger setting is a number between 0 inclusive and NUM_CHARGER_SETTINGS exclusive. The charger setting is set by the user repeatedly pressing a single settings button. Each press the setting increases by one. However, when the maximum setting is reached the next button press puts it back to setting 0.

The formulas that guide the rate of charging and discharging are as follows:

$$increase\ in\ charge\ for\ camera\ charging = minutes\ \times\ CHARGE\_RATE$$

$$increase\ in\ charge\ for\ extrenal\ charging = minutes\ \times\ setting\ \times\ CHARGE\_RATE$$

$$decrease\ in\ charge\ for\ camrea\ power\ drain = minutes\ \times\ camera\ power\ consumption$$

## Special Assignment Requirements

**You should not use any conditional statements (i.e. "if" statements), loops,** or anything else we haven't covered, for this assignment. The purpose of this assignment is for you to practice using variables and the math tools we have covered in various ways. Yes, it is possible to solve this assignment very neatly with just what we have learned, and the challenge of doing so will make you a better programmer! You will be penalized slightly for using conditional statements, keep in mind that if you really can't figure it out, it's better to turn in something working than nothing at all.

**Hints:** To solve this assignment you will need to declare instance and local variables, that part of the program design is up to you. 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.max()` or `Math.min()`. For the wrapping behavior of the settings button, you can use the mod (`%`) operator. To keep the camera or external charger from charging when the battery is removed, just set the charge capacity of the camera or charger to 0.

## Specification

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

**There are some constants:**

```
public static final int NUM_CHARGER_SETTINGS = 4;
```

Number of external charger settings. Settings are numbered between 0 inclusive and 4 exclusive.

```
public static final double CHARGE_RATE = 2.0;
```

A constant used in calculating the charge rate of the external charger.

```
public static final double DEFAULT_CAMERA_POWER_CONSUMPTION = 1.0;
```

Default power consumption of the camera at the start of simulation.

**There is one public constructor:**

```
public CameraBattery(double batteryStartingCharge,
                     double batteryCapacity)
```

Constructs a new camera battery simulation. The battery should start disconnected from both the camera and the external charger. The starting battery charge and maximum charge capacity are given. If the starting charge exceeds the batteries capacity, the batteries charge is set to its capacity. The external charger starts at setting 0.

**There are the following public methods:**

```
public void buttonPress()
```

Indicates the user has pressed the setting button one time on the external charger. The charge setting increments by one or if already at the maximum setting wraps around to setting 0.

```
public double cameraCharge(double minutes)
```

Charges the battery connected to the camera (assuming it is connected) for a given number of minutes. The amount of charging in milliamp-hours (mAh) is the number minutes times the CHARGE_RATE constant. However, charging cannot exceed the capacity of the battery connected to the camera, or no charging if the battery is not connected. The method returns the actual amount the battery has been charged.

```
public double drain(double minutes)
```

Drains the battery connected to the camera (assuming it is connected) for a given number of minutes. The amount of drain in milliamp-hours (mAh) is the number of minutes times the cameras power consumption. However, the amount cannot exceed the amount of charge contained in the battery assuming it is connected to the camera, or no amount drain if the battery is not connected. The method returns the actual amount drain from the battery.

```
public double externalCharge(double minutes)
```

Charges the battery connected to the external charger (assuming it is connected) for a given number of minutes. The amount of charging in milliamp-hours (mAh) is the number minutes times the charger setting (a number between 0 inclusive and NUM_CHARGER_SETTINGS exclusive) the CHARGE_RATE constant. However, charging cannot exceed the capacity of the battery connected to the camera, or no charging if the battery is not connected. The method returns the actual amount the battery has been charged.

```
public void resetBatteryMonitor()
```

Reset the battery monitoring system by setting the total battery drain count back to 0.

```
public double getBatteryCapacity()
```

Get the battery's capacity.

```
public double getBatteryCharge()
```

Get the battery's current charge.

```
public double getCameraCharge()
```

Get the current charge of the camera's battery.

```
public double getCameraPowerConsumption()
```

Get the power consumption of the camera.

```
public int getChargerSetting()
```

Get the external charger setting.

```
public double getTotalDrain()
```

Get the total amount of power drained from the battery since the last time the battery monitor was started or reset.

```
public void moveBatteryExternal()
```

Move the battery to the external charger. Updates any variables as needed to represent the move.

```
public void moveBatteryCamera()
```

Move the battery to the camera. Updates any variables as needed to represent the move.

```
public void removeBattery()
```

Remove the battery from either the camera or external charger. Updates any variables as needed to represent the removal.

```
public void setCameraPowerConsumption(double cameraPowerConsumption)
```

Set the power consumption of the camera.

## 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 like the examples below in the getting started section.

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 below.

## 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 **`CameraBattery`** class in the **`hw1`** package and put in stubs for all the required methods, the constructor, and the constants. Remember that everything listed is declared **`public`**. For methods that are required to return a value, for now, just put in a "dummy" return statement that returns zero or false. There should be no compile errors. ***WARNING: be careful about COPY AND PASTE from this pdf!*** *This may lead to insidious bugs caused by invisible characters that are sometimes generated by sophisticated document formats.*

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 phrases from the method descriptions here or in the Javadoc is acceptable, however, see waring above.*)*

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 could start by deciding how to keep track of the camera's charge. The presence of a method **`getCameraCharge`** suggests an instance variable might be useful to keep track of the charge. Keep in mind the specification states the battery starts disconnected from the camera,

which means the camera's initial charge should be 0. In a separate class (and file), write a simple test to see if this much is working as it should:

```
public static void main(String args[]) {
    CameraBattery cb = new CameraBattery(1000, 2000);
    System.out.println("Test 1:");
    System.out.println("Battery charge is " + cb.getBatteryCharge()
          + " expected 1000.0");
    System.out.println("Camera charge is " + cb.getCameraCharge()
          + " expected 0.0");
}
```

(*Tip:* you can find code for the simple test case above, along with the others discussed in this section, in the class **`SimpleTests.java`** linked from the assignment page on Canvas.)

6. A call to **`moveBatteryCamera`**, models the battery begin connected to the camera. The camera's charge should be the same as the batteries charge. Think about how the move should update the instance variables you created in the previous steps. A test of the method might look like the following:

```
// battery is connected to the camera for next tests
cb.moveBatteryCamera();

System.out.println();
System.out.println("Test 2:");
System.out.println("Battery charge is " + cb.getBatteryCharge()
      + " expected 1000.0");
System.out.println("Camera charge is " + cb.getCameraCharge()
      + " expected 1000.0");
```

7. Now you can start working on the **`drain`** method. This method models the camera draining the battery. The battery's (and camera's) charge must be decreased by the given number of minutes times the camera's power consumption. However, the charge cannot be decreased below 0. Think about the math library operation that can be used prevent a value going below a threshold amount (in this case 0). The total drain should also be updated to reflect how much the battery has been drained since it was created or last reset.

```
// before calling drain
System.out.println("Total drain is " + cb.getTotalDrain()
      + " expected 0.0");
System.out.println("Camera power consumption is "
      + cb.getCameraPowerConsumption() + " expected " + 1.0);

double drained = cb.drain(100.0);
System.out.println();
System.out.println("Test 4:");
System.out.println("Battery drained by " + drained + " expected 100.0");
System.out.println("Battery charge is " + cb.getBatteryCharge()
      + " expected 900.0");
```

```
System.out.println("Camera charge is " + cb.getCameraCharge()
      + " expected 900.0");
System.out.println("Total drain is " + cb.getTotalDrain()
      + " expected 100.0");

drained = cb.drain(1000.0);
System.out.println();
System.out.println("Test 5:");
System.out.println("Battery drained by " + drained + " expected 900.0");
System.out.println("Battery charge is " + cb.getBatteryCharge()
      + " expected 0.0");
System.out.println("Camera charge is " + cb.getCameraCharge()
      + " expected 0.0");
System.out.println("Total drain is " + cb.getTotalDrain()
      + " expected 1000.0");
```

8. The implementation of **`cameraCharge`** is very similar to drain. The battery (and the camera) is charged by the minutes times the CHARGE_RATE constant. However, the charge cannot exceed the battery's total capacity.

```
double charged = cb.cameraCharge(50.0);
System.out.println();
System.out.println("Test 6:");
System.out.println("Battery charged by " + charged + " expected 100.0");
System.out.println("Battery charge is " + cb.getBatteryCharge()
      + " expected 100.0");
System.out.println("Camera charge is " + cb.getCameraCharge()
      + " expected 100.0");
```

9. You also need to implement the **`removeBattery`** and **`moveBatteryExternal`** methods. They are similar to **`moveBatteryCamera`**.

```
// battery is removed for the next couple tests
cb.removeBattery();

drained = cb.drain(50.0);
System.out.println();
System.out.println("Test 7:");
System.out.println("Battery drained by " + drained + " expected 0.0");
System.out.println("Battery charge is " + cb.getBatteryCharge()
      + " expected 100.0");
System.out.println("Camera charge is " + cb.getCameraCharge()
      + " expected 0.0");
System.out.println("Total drain is " + cb.getTotalDrain()
      + " expected 1000.0");

cb.cameraCharge(50.0);
System.out.println();
System.out.println("Test 8:");
System.out.println("Battery charge is " + cb.getBatteryCharge()
      + " expected 100.0");
System.out.println("Camera charge is " + cb.getCameraCharge()
      + " expected 0.0");
```

```
System.out.println("Total drain is " + cb.getTotalDrain()
      + " expected 1000.0");

// battery is moved to the external charger for the next few tests
cb.moveBatteryExternal();

cb.externalCharge(50.0);
System.out.println();
System.out.println("Test 9:");
System.out.println("External charger setting is " + cb.getChargerSetting()
      + " exptected 0");
System.out.println("Battery charge is " + cb.getBatteryCharge()
      + " expected 100.0");
System.out.println("Camera charge is " + cb.getCameraCharge()
      + " expected 0.0");
System.out.println("Total drain is " + cb.getTotalDrain()
      + " expected 1000.0");

cb.buttonPress();
cb.buttonPress();
cb.externalCharge(50.0);
System.out.println();
System.out.println("Test 10:");
System.out.println("External charger setting is " + cb.getChargerSetting()
      + " exptected 2");
System.out.println("Battery charge is " + cb.getBatteryCharge()
      + " expected 300.0");
System.out.println("Camera charge is " + cb.getCameraCharge()
      + " expected 0.0");
System.out.println("Total drain is " + cb.getTotalDrain()
      + " expected 1000.0");
```

10. Next is the `buttonPress` method. This method increments the setting by one each time it is called. The settings are 0, 1, 2, and 3. When setting 3 is reached, the setting returns back to 0. Again, think of the mathematical operations we have learned that could be used to implement this functionality.

```
cb.buttonPress();
cb.buttonPress();
cb.externalCharge(50.0);
System.out.println();
System.out.println("Test 10:");
System.out.println("External charger setting is " + cb.getChargerSetting()
      + " exptected 2");
System.out.println("Battery charge is " + cb.getBatteryCharge()
      + " expected 300.0");
System.out.println("Camera charge is " + cb.getCameraCharge()
      + " expected 0.0");
System.out.println("Total drain is " + cb.getTotalDrain()
      + " expected 1000.0");

cb.buttonPress();
cb.buttonPress();
cb.externalCharge(50.0);
```

```
System.out.println();
System.out.println("Test 11:");
System.out.println("External charger setting is "
      + cb.getChargerSetting() + " exptected 0");
System.out.println("Battery charge is " + cb.getBatteryCharge()
      + " expected 300.0");
System.out.println("Camera charge is " + cb.getCameraCharge()
      + " expected 0.0");
System.out.println("Total drain is " + cb.getTotalDrain()
      + " expected 1300.0");
```

11. Now finish the last remaining methods. The method **`externalCharge`** is similar to **`cameraCharge`**. There a several methods that are straight forward accessors (getters) which only return the value of an instance variable.

11. 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 conditional statement when you could just be using **`Math.min`**? 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****! 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, **`CameraBattery.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", linked on the Course Information page on Canvas.

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 **CameraBattery.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.