Com S 227
Spring 2024
Assignment 1
100 points

Due Date: Friday, February 9, 11:59 pm (midnight)
5% bonus for submitting 1 day early (by 11:59 pm Feb 8)
10% penalty for submitting 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!

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. Wait until the day before the assignment is due to start using the SpecChecker. Or better yet don't test your code at all, it's such fun to remain in suspense until it's graded!
3. Don't bother reading the rest of this document, or even the specification, especially not the "Getting started" section. Documentation is for losers. Waste time writing lots of 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. You'll also get practice using variables, expression, modular arithmetic, and the Math library.

For this assignment you will create a hot air balloon simulator. Specifically, you are implementing a single class called **`Balloon`**, that models the flight of a hot air balloon taking multiple factors into account, for example: the mass of the balloon, outside air temperature, available fuel, rate of fuel burn, etc.

A hot air balloon is a lighter-than-air aircraft consisting of a bag, called an envelope, which contains heated air. Suspended beneath is a gondola or wicker basket (in some long-distance or high-altitude balloons, a capsule), which carries passengers and a source of heat, in most cases an open flame caused by burning liquid propane. The heated air inside the envelope makes it buoyant, since it has a lower density than the colder air outside the envelope.

https://en.wikipedia.org/wiki/Hot_air_balloon

The simulation models fuel being burned to heat the air inside of a balloon. The heated air inside of the balloon has a lower density than the air outside of the balloon, generating a lifting force. When the lifting force overcomes gravity, the balloon rises. The balloon can also drop when the lifting force is not greater than gravity, but of course, the balloon can't drop below ground level. Also, the balloon is on a tether (rope attached to the ground) that prevents it from getting too high. As time passes in the simulation, the altitude of the balloon is calculated as $A_{balloon}$, as long as it is between ground level and the maximum extent of the tether, using the following equations.

Inputs to the simulation

Air temperature outside the balloon in C.
$T_{outside}$

Rate of fuel burn per second in kBTU/s.
B

Balloon mass.
m

Provided simulation constants

Heat loss factor.
H = 0.1

The volume of air in the balloon in m^3
V = 61234

Acceleration due to gravity in m/s^2.

$A_g = 9.81$

The gas constant in J/kgK.
$R = 287.05$

Standard pressure in hPa.
$P = 1013.25$

Kelvins at 0 degrees C.
$K_c = 273.15$

Simulation calculations

The rate of change in the balloon's air temperature per second (assuming sufficient fuel).
$\Delta T = B + (T_{outside} - T_{balloon}) \times HEAT\_LOSS$

The temperature of the air inside of the balloon after one second.
$T_{balloon\ at\ current\ time} = T_{balloon\ one\ second\ ago} + \Delta T$

Density of the surrounding air (kg/m^3).
$$\rho_c = \frac{P}{R\ (T_{outside} + K_c)}$$

Density of the balloon air (kg/m^3).
$$\rho_u = \frac{P}{R\ (T_{balloon} + K_c)}$$

Force of lift in N.
$F_L = V\ (\rho_c - \rho_u)\ A_g$

Force of gravity in N.
$F_g = m\ A_g$

Net force in upward direction in N.
$F_n = F_L - F_g$
Note: this gives wrong results when the balloon is on the ground or at the end of the tether, the net force should be zero, but we will ignore this detail for the sake of simplicity.

Net acceleration in upward direction.
$A_n = F_n / m$

Velocity in upward direction (assuming 1 second of time) in m/s.
$v_{at\ time\ n} = v_{at\ time\ n-1} + A_n$

The altitude of the balloon after one second. $A_{\text{balloon at time n}} = A_{\text{balloon at time n-1}} + v_{\textit{at time n}}$

Special Assignment Requirements

You are not allowed to use conditional statements in this homework. That means specifically **no "if" or "if-else" statements**, **no loops** of any sort, and **no ternary operators** (don't worry if you don't know what those are yet).

But why?

The purpose of this assignment is to give you practice using variables and mathematical expressions. Wouldn't it be just as easy to use conditionals to solve this particular assignment? Perhaps, but that won't always be the case. Professional programmers use mathematical expression every day to simplify code, make it more readable, and make it more efficient. You're not ready yet to write code for the Linux kernel or the InnoDB database engine or the javac compiler... So, we decided to give you an easier programming assignment and artificially tie your hands (so to speak) by forcing you to implement the logic with only variables and mathematically expressions. Some day when you are adding a patch to the Linux kernel you might thank us for the practice.

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 example, the balloon has a minimum and maximum allowable altitude, ground level (0) and the height of the tether respectively. These functions can be used to enforce the altitude boundaries. For the wrapping (circular) behavior, you can use the mod (`%`) operator.

Yes, it is possible to solve this assignment very neatly with just what we have learned so far in class, 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.**

More about grading

This is a "regular" assignment so we are going to read your code. Your score will be based partly (about 2/3) 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, but 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:

- The class and **every method, constructor and instance variable, whether public or private, must have a meaningful Javadoc style 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. The javadoc for instance variables can be a very short description of the purpose of the variable.
- 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.

The SpecChecker

A significant portion (but not all) of the points for this assignment are based on the specchecker score. It is not advised to wait until the last day to run the SpecChecker on your code.

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. Submit the zip to the Canvas assignment.

The UI

For your amusement, and as a reward for when everything is implemented, a UI has been provided with this assignment. Be aware that UIs are generally not a good tool for testing and debugging code. The UI might give you a hint when something is wrong, but it won’t catch every error and it won’t tell you much about the cause of an error. The bottom line is that you should test your code by writing simple tests and print out the values of calculations to help understand the cause of bugs.

To use the UI, download **`hw1ui.jar`**. Follow the same instructions as the specchecker to setup and run the UI. Running the jar as an application should start the UI.

Specification

The specification for this assignment includes this pdf along with any "official" clarifications announced on Canvas. The specification is for a single class **Balloon**. You may (are encouraged to) create a second class called **SimpleTests** (as described in the *Suggestions for getting started* section) for your own testing purposes, but you will not be graded on that class.

There are 6 constant values defined in the table of formulas in the *Overview* section of this document. Use best programming practices when incorporating these constant values in your code.

There is one public constructor:

```
public Balloon(double airTemp, double windDirection)
```

Constructs a new balloon simulation. The simulation starts with the given **`airTemp`** (outside air temperature in C) and **`windDirection`** (in degrees). It is assumed **`windDirection`** is between 0 (inclusive) and 360 (exclusive). The balloon temperature (air inside the balloon) is initialized to the same temperature as the outside air. The simulation time is initialized to the default value 0. The balloon's altitude, remaining fuel, fuel burn rate, balloon mass, velocity, and tether length are all initialized to 0.

There are the following public methods:

```
public double getFuelRemaining()
```

Gets the remaining fuel that can be used to heat the air in the balloon.

```
public void setFuelRemaning(double fuel)
```

Sets the remaining fuel that can be used to heat the air in the balloon.

```
public double getBalloonMass()
```

Gets the mass of the balloon (m in the formulas).

```
public void setBalloonMass(double mass)
```

Sets the mass of the balloon (m in the formulas).

```
public double getOutsideAirTemp()
```

Gets the outside air temperature ($T_{outside}$ in the formulas).

```
public void setOutsideAirTemp(double temp)
```

Sets the outside air temperature ($T_{outside}$ in the formulas).

```
public double getFuelBurnRate()
```

Gets the fuel burn rate (B in the formulas).

```
public void setFuelBurnRate(double rate)
```

Sets the fuel burn rate (B in the formulas).

```
public double getBalloonTemp()
```

Gets the balloon temperature ($T_{balloon}$ in the formulas).

```
public void setBalloonTemp(double temp)
```

Sets the balloon temperature ($T_{balloon}$ in the formulas).

```
public double getVelocity()
```

Gets the balloon velocity (v in the formulas).

```
public double getAltitude()
```

Gets the balloon altitude.

```
public double getTetherLength()
```

Gets the length of the tether.

```
public double getTetherRemaning()
```

Gets the length of the tether minus the current altitude of the balloon.

```
public void setTetherLength(double length)
```

Sets the length of the tether.

```
public double getWindDirection()
```

Gets the direction of the wind in degrees, a number between 0 (inclusive) and 360 (exclusive).

```
public void changeWindDirection(double deg)
```

Updates the wind direction by adding the giving value (which is assumed to be between -360 and 360 exclusive) on to the current wind direction. The wind direction must always be between 0 (inclusive) and 360 (exclusive).

Hint: If the direction goes below 0 or to 360 or higher you need to get it back to between 0 to 360. There is more than one way to do this. Do be careful that the modulus of a negative number has an unexpected result in Java. For example, -1 % 2 is -1. One simple way to deal with this is to add an extra 360 degrees to avoid any negative values in the first place.

```
public long getMinutes()
```

Gets the number of full minutes that have passed in the simulation. For example, if the simulation time is 179, minutes = 2 and seconds = 59.

```
public long getSeconds()
```

Gets the number of seconds passed the number of full minutes. For example, if the simulation time is 179, minutes = 2 and seconds = 59. The seconds should always be between 0 and 59 inclusive.

```
public void update()
```

Calling this method represents 1 second of simulated time passing. Specifically, the simulation time is incremented by 1. The fuel remaining is consumed at the set fuel burn rate, but it can never drop below 0. If the fuel burn rate is more than the available amount of fuel then consume as much fuel as is available but no more. The temperature inside of the balloon is updated (see formulas in the Overview section).

The velocity and position of the balloon is also updated based on the formulas. Note that the calculations for velocity and position at time n depend on the velocity and position at time n-1 (one second ago). In other words, each calculation of velocity and position depends on the previous calculation.

There are two exceptions to the calculation of the position of the balloon; the balloon can never drop below ground level (an altitude of 0) and can never rise above the length of the tether. That is to say, the ground level and the tether length are the minimum and maximum altitudes respectively. The velocity and altitude can only be calculated after updating the balloon temperature.

```
public void reset()
```

Calling this method resets the simulation to its initial state (the same state it had immediately after the constructor was called). Pay attention that the outside air temperature and wind direction are reset to the values that were provided to the constructor. (Hint: How will you do that when those parameters are not in the scope of this method? You have learned all the tools necessary; but it will take some planning to implement the solution.)

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 **`Balloon`** 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. It's better to re-type it by hand.*

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 balloon's fuel. The presence of a method **`getFuelRemaining`** suggests an instance variable might be useful to keep track how much fuel remains in the tank. Keep in mind the specification states the fuel remaining starts empty. 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[]) {
    Balloon b = new Balloon(20, 90);
    System.out.println("Test 1:");
    System.out.println("Fuel remaining is "
          + b.getFuelRemaining() + " expected 0.");
    System.out.println("Adding 1000 to fuel.");
    b.setFuelRemaning(1000);
    System.out.println("Fuel remaining is "
          + b.getFuelRemaining() + " expected 1000.");
```

(*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. In addition to fuel remaining, there are 5 other values described in the specification for the constructor that have simple getters and setters associated with them (we will skip time, wind direction and velocity for now). Implements these getters and setters and update the constructor.

```
System.out.println("Test 2:");
System.out.println("Air temp is " + b.getOutsideAirTemp()
      + " expected 20.");
System.out.println("Balloon temperature is "
      + b.getBalloonTemp() + " expected 20.");
System.out.println("Fuel burn rate is " + b.getFuelBurnRate()
      + " expected 0.");
System.out.println("Tether length is " + b.getTetherLength()
      + " expected 0.");
System.out.println("Balloon mass is " + b.getBalloonMass()
      + " expected 0.");

b.setOutsideAirTemp(18);
b.setBalloonTemp(25);
b.setFuelBurnRate(5);
b.setTetherLength(100);
b.setBalloonMass(110);

System.out.println("Air temp is " + b.getOutsideAirTemp()
      + " expected 18.");
System.out.println("Balloon temperature is " + b.getBalloonTemp()
      + " expected 25.");
System.out.println("Fuel burn rate is " + b.getFuelBurnRate()
      + " expected 5.");
System.out.println("Tether length is " + b.getTetherLength()
      + " expected 100.");
System.out.println("Balloon mass is " + b.getBalloonMass()
      + " expected 110.");
```

7. Now you can start working on two of the slightly less strait forward methods, the **`getMinutes`** and **`getSeconds`** methods. These methods imply you need some way of keeping track of the current time. Should it be two separate variables or a single variable that can be used to derive minutes and seconds? That is up to you. The only way time can increment is when **`update`** is called. You don't need to implement all of **`update`**, but for now just add the code to increment time by one second.

```
System.out.println("Test 3:");
System.out.println("Minutes is " + b.getMinutes() + " expected 0.");
System.out.println("Seconds is " + b.getSeconds() + " expected 0.");
b.update();
b.update();
b.update();
System.out.println("Minutes is " + b.getMinutes() + " expected 0.");
System.out.println("Seconds is " + b.getSeconds() + " expected 3.");
```

8. The next set of methods to implement are the **`getWindDirection`** and **`changeWindDirection`** methods. You must ensure that the direction always remains between 0 (inclusive) and 360 (exclusive). See the specification of **`changeWindDirection`** for some helpful hints.

```
System.out.println("Test 4:");
System.out.println("Wind direction is " + b.getWindDirection()
      + " expected 90.");
b.changeWindDirection(180);
System.out.println("Wind direction is " + b.getWindDirection()
      + " expected 270.");
b.changeWindDirection(90);
System.out.println("Wind direction is " + b.getWindDirection()
      + " expected 0.");
b.changeWindDirection(-90);
System.out.println("Wind direction is " + b.getWindDirection()
      + " expected 270.");
```

9. Before continuing, it is time to implement the **`reset`** method. This method is mostly straight forward, but you will need to devise a plan for how to set the air temperature and wind direction back to the values that were passed to the constructor.

```
System.out.println("Test 5:");
b.reset();
System.out.println("Air temp is " + b.getOutsideAirTemp()
      + " expected 20.");
System.out.println("Wind direction is " + b.getWindDirection()
      + " expected 90.");
System.out.println("Fuel remaining is " + b.getFuelRemaining()
      + " expected 0.");
System.out.println("Balloon temperature is " + b.getBalloonTemp()
      + " expected 20.");
System.out.println("Fuel burn rate is " + b.getFuelBurnRate()
      + " expected 0.");
System.out.println("Tether length is " + b.getTetherLength()
      + " expected 0.");
System.out.println("Balloon mass is " + b.getBalloonMass()
      + " expected 0.");
```

10. Finally, it is time to implement the **`update`** method. For this method you will need to implement the formulas described in the Overview section of this document. Remember that you can use local variables to break up longer calculations into smaller parts. Math functions such as min and max may be helpful when implementing the upper and lower bounds on the altitude.

```
System.out.println("Test 6:");
b.setBalloonMass(100);
b.setBalloonTemp(70);
b.setFuelBurnRate(5);
b.setFuelRemaning(10);
b.setTetherLength(100);
System.out.println("Balloon temperature is " + b.getBalloonTemp()
      + " expected 70.");
```

```
System.out.println("Balloon velocity is " + b.getVelocity()
      + " expected 0.");
System.out.println("Altitude is " + b.getAltitude()
      + " expected 0.");
b.update();
System.out.println("Balloon temperature is " + b.getBalloonTemp()
      + " expected 70.");
System.out.println("Balloon velocity is " + b.getVelocity()
      + " expected 0.72...");
System.out.println("Altitude is " + b.getAltitude()
      + " expected 0.72...");
b.update();
System.out.println("Balloon temperature is " + b.getBalloonTemp()
      + " expected 70.");
System.out.println("Balloon velocity is " + b.getVelocity()
      + " expected 1.45...");
System.out.println("Altitude is " + b.getAltitude()
      + " expected 2.18...");
// note: at time point fuel has run out
b.update();
System.out.println("Balloon temperature is " + b.getBalloonTemp()
      + " expected 65.");
System.out.println("Balloon velocity is " + b.getVelocity()
      + " expected 1.27...");
System.out.println("Altitude is " + b.getAltitude()
      + " expected 3.46...");
b.update();
System.out.println("Balloon temperature is " + b.getBalloonTemp()
      + " expected 60.5.");
System.out.println("Balloon velocity is " + b.getVelocity()
      + " expected 0.24...");
System.out.println("Altitude is " + b.getAltitude()
      + " expected 3.70...");
b.update();
System.out.println("Balloon temperature is " + b.getBalloonTemp()
      + " expected 56.45.");
System.out.println("Balloon velocity is " + b.getVelocity()
      + " expected -1.56...");
System.out.println("Altitude is " + b.getAltitude()
      + " expected 2.14...");
```

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.**

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, **`Balloon.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 **Balloon.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.

Document updates:

01/30 On page 16, the name of the file the SpecChecker puts into the zip is called "Balloon.java".

01/30 Fixed example test code in “Suggestions for getting started” section to match with provided SimpleTests.java file.

02/02 Method name change to setFuelRemaning to match specchecker (see announcement). Added comment “assuming sufficient fuel” to formula (see announcement).