COMS W1007 Assignment 5
Due Dec. 11, 2018

1 Theory Part (45 points)

The following problems are worth the points indicated. They cover some additional patterns, and some topics in advanced Java.

"Paper" programs are those which are composed (handwritten or typed) on the same sheet that the rest of the theory problems are composed on, and are not to be text-edited, compiled, executed, or debugged. For fairness, there will be no extra credit for doing any work for paper programs beyond the design and coding necessary to manually produce the objects or object fragments that are asked for, so text editor or computer output will have no effect on your grade. The same goes for the Theory Part in general: scanned clear handwriting is sufficient, and computer-generated documents will earn no extra credit.

An important note about academic honesty: If you use *any* resource, including `stackoverflow.com`, you must properly attribute the source of your answer with the proper citation. Cutting and pasting without attribution is plagiarism. The CAs will report to the instructor any suspected violations. This even goes for `wikipedia.org`–which for this course often turns out to be inaccurate or overly complicated, anyway–since plagiarizing from Wikipedia is still plagiarism. Much better to read, think, and create your own sentences.

1.1 Visitor pattern (6 points)

Calling an Uber in real life is an example of the Visitor pattern, in that a Person accepts the Uber driver, which then does its methods on the Person ("this"). Give another example in real life, where a user willingly relinquishes control of user "data" to a "visitor". Indicate what are the protections for the "data" that either laws or social convention maintain in real life. Then rewrite the five code fragments given on Courseworks for the Visitor pattern so that the small system illustrates your real life case, where the fields and the methods reflect the real life situation.

1.2 Shape interface (10 points)

The Shape interface in Java is one of the most important interfaces for implementing GUIs. Give the complete UML hierarchy tree–where the UML box just has the API class name, like just "Rectangle2D"–for all the classes implementing this interface. There are about 30 of them. Then say what happened to the class Point2D: why does it not implement Shape? (More properly, what methods of the Shape interface just don't make sense for Point?) Lastly, is there an interface in the API for Shape3D? Speculate on why or why not. Note: This question is meant to be easy but perhaps a bit long, but exploring this part of the API will be useful in future GUI design work.

1.3 Advanced Java: Tagging interfaces (10 points)

Some things that look like patterns are not really patterns, but give instructions to the compiler. For example, “implements Serializable” looks like it might be a pattern, but it is a “tagging interface” that makes no promises about methods, and it is unique to Java. Find the other tagging interfaces in the API (that is, interfaces that have no methods at all), and tell what they are used for. Note that the API italicizes interfaces, so this should make the search easier, and to make it easier still, just find all tagging interfaces that end in “-able”. (There are about 30 interfaces that end in “-able” but not all of them are tagging interfaces.) Are tagging interfaces a good thing or a bad thing? Why?

1.4 Advanced Java: Arrays and Reflection (7 points)

The following code is weird. Run it and see why.

```
Line2D[] lines = new Line2D[10];
Shape[] shapes = lines;
shapes[0] = new GeneralPath();
```

Explain why each line compiles. Explain why it nevertheless gives a runtime error. Illustrate your answer with a simple storage diagram showing the boxes for “lines” and “shapes” on the stack, complete with annotations as to type, and whatever boxes they point to in the heap, complete with annotations as to type.

1.5 Frameworks and Applets (5 points)

Sometimes a framework is very complicated. Find the Collections framework in the Java API. How many fields and methods does it have? Then find its source code online at `www.docjar.com`. How many lines, how many imports? Then, find Applet. How many fields, methods, lines, imports? Is a big framework good or bad, and why?

1.6 Advanced Java: Hash code (7 points)

Write a hashcode() to handle the class SuperEnaLotto, which represents an Italian lottery ticket. An instance of this class has a set of six different numbers, chosen from 1 to 90. So, the hashcode() can use a private int[6] array, p, to store these, as long as it satisfies the proper invariant, which is $1 <= p[i] <= 90$ AND if $i \neq j$, then $p[i] \neq p[j]$ AND isSorted(p). Then, a seventh number, the SuperStar, is also chosen, also from 1 to 90, which can match any of the other six.

It is not hard to show that there are 90*89*88*87*86*85/6! * 90 possible SuperEnaLotto tickets, or exactly 56,035,316,700 of them, which is a bigger range than what can be stored in an int. The hashcode() has to map these 7 integers into a single int, which uses as much of the entire range that is available in an int. Then, show a concrete example of how it is possible, when using your hashcode(), for two SuperEnaLotto instances to be different, that is, !mySuperEnaLotto.equals(yourSuperEnaLotto), but still have identical hashcodes, that is, mySuperEnaLotto.hashcode() == yourSuperEnaLotto.hashcode().

2 Programming Part (55 points)

This assignment is intended to explore the applet framework. It creates a web-enabled animation of a kind of RPSLK battle. We do not require CRC nor UML, but do require the usual Javadoc comments within the implementation of your design. You also need to provide test examples, whose output may take the form of a short (20-second or less) video.

Please recall that all programs must compile, so keep a working version of each part before your proceed to the next.

Part of this assignment is to give you a small experience of the real word, in which successful code gets maintaining and enhanced. So, this assignment it asks you to use some of what you wrote for Assignment 1. Please follow the Boy Scout rule and clean up what you wrote ten weeks ago!

2.1 Step 1 (5 points): Getting started.

Import the Playground.java code that is online in Courseworks. Make sure you properly attribute the source of this code! Also, import Playground.html, which is in Playground.html.txt. Note that you will have to rename that file; the ".txt" is added to keep your browser from trying to run the html.

Find out how Eclipse *simulates* the html file (using the "Run" then "Run Configurations" menu item). Infer from the *.java file what the parameter names and values should be; the values are yours to choose appropriately. Get the Playground applet running in Eclipse.

Verify that the system allows you to adjust the size of the Applet window, and that the start and stop methods actually affect the Timer (although you do not have to prove this by submitting any physical output).

Then, create a real HTML text file, using the template that is in the javadoc of the java file and the example that is in the html. You can do this in Eclipse, if you want, by right-clicking on the package name, then doing "New" then "Other" then "Web" then "HTML". Or, you can just use a regular old text editor. Then move this text file somewhere on your platform, outside of the Eclipse environment, and copy the java file into the same directory. Then compile the java file, and run "appletviewer", which should be available as part of your Java environment. You would then say something like this (depending on your operating system):

```
appletviewer Playground.html
```

If you stop at this Step, simply submit the html file you created as proof, and a single screenshot of the Appletviewer window.

Special Note: It may be possible to run this through your browser, or through a friend's browser. This requires a lot of permission setting in both your system's Java control panel and the browser's security settings, and generally is quite complex and dependent on the exact version of Java and the browser. Since systems vary a lot, feel free to exchange notes about how to do this on Piazza. In the past, only Firefox has proven to be easy to get to work properly.

2.2 Step 2 (10 points): Two-dimensionality.

Using Step 1, make the string movable in both dimensions, x and y, but still at a velocity determined by the HTML. And, the string should be able to start in a location that can be specified in the HTML.

That is, instead of starting at the same place and moving to the left, it can start anywhere within the applet (how do you make sure it is within the applet?) and it can move in nine different ways at a constant speed. Using compass directions, this would be N, NE, E, SE, S, SW, W, NW, or it can stay in the same place. You will have to extend the HTML to allow two parameters for the string that tell it where to start (x and y), and two more parameters that say what the horizontal and vertical velocity direction of the string should be. You can indicate the horizontal as (-1, 0, 1) or as ("W", " ", "E"), and similarly the vertical, depending on what you think best.

You will now also have to detect when the string hits any of the *four* borders of the applet, not just the left one. And, at encountering the border, the string should wrap around to the other side of the applet like you saw in Step 1. Be careful of when a string approaches a corner!

If you do this Step, you do not have to submit separate output from Step 1. But you have to submit your code and either some screenshots or a 20-second video, as you did for Assignment 4.

2.3 Step 3 (10 points): Many colorful strings and a black hole.

Enhance Step 2 to allow two or more strings. They all still start where the HTML says, and move at the constant speeds in one of the nine directions according to their HTML, and they "pass through" each other. Pay careful attention to the data structure, and how you structure the HTML. You are allowed to refactor the HTML to make it shorter, for example, if all the velocities are in common, or if all the starting points have the same row or column or both. You may find the Eclipse html editor helpful.

Most importantly, the strings *must* be visibly different, since many studies have shown that viewers perceive graphics best if colors, textures, and sizes are suggestive of the meaning of the associated text. So, instead of just saying "Rock", the String should *look* like a Rock: maybe in thick brown fixed font. "Paper" should be thin, "Lizard" could be in a wiggly-looking font, etc. Please consider that good CS is also good art!

Then, add a Black Hole, so it is RPSLKH, which, of course, is black and roundish. You will find that in the next Step a Black Hole always wins.

If you do this Step, you do not have to submit separate output from Steps 1 and 2, just the code and a 20-second movie.

2.4 Step 4 (20 points): RPSLKH Modern Warfare.

Using the strings Rock, Paper, Scissors, Lizard, Spock, detect *collisions* between strings when they are "close enough" to each other when the timer goes off. You will probably want to check in the API for a Rectangle method called "intersects" to get some ideas on how to do this. You will also need to have a data structure that allows a quick check of this "closeness"; you may even want to use some kind of a sort to help you.

Then, apply the rules (and some of the code!) for RPSLK you designed in Assignment 1 to determine which strings survive a collision. So, if "Rock" and "Paper" collide, only "Paper" continues; “Rock” is removed entirely. But if a “Rock” collides with another “Rock”, both should survive. Note that this can get tricky, since more than two strings can collide at once, depending on how you define “close enough”. What happens if two “Rocks” collide with one “Paper”? You might even have to make special rules, such as, if a “Rock”, a “Paper”, and a “Scissors” (or a similar trio) collide all at once, maybe all survive. Or maybe none survive. Or maybe a randomly chosen one survives. Or maybe a randomly chosen one doesn’t.

Note that a Black Hole always survives, and that anything that it collides with does not– including another Black Hole.

It gets a bit complicated, since you can have simultaneous collisions of very many strings in several different places. And, of course, you must document these decisions and code somewhere.

Note that there are the usual corner cases, too. How many Strings can your window handle? Can they move so quickly that they never intersect at all (due to digitization, can they “hop” over each other)? Can all Strings be stationary, and if not, how do you alert the user? Do you have to worry if two Strings have exactly the same content, location, and movement, so then you see only one? What happens if there hasn’t been any collisions for a while–does the display keep running?

Note that there are some simple testing strategies for this Step, since you can easily arrange that some of the strings are stationary. Then they can easily be “hit” by other strings that start off in the same row or column going straight toward them.

As usual, if you do this Step, you do not have to submit separate output from the prior Steps, just the usual code and video.

2.5 Step 5 (10 points): Creative user interface.

Whenever a collision is detected, the display should pause, and print in the applet at the collision some description of what collision(s) have been detected and what the rules say will happen. Something like: “P covers R” or “R ties R”, or “H devours K”. Then after a fixed amount of time, the applet continues and the strings continue moving. To help the user further, the winner(s) should *grow in size*, by an algorithm of your choice. Note that a bigger winner might cause an immediate further collision. Your applet should stop when there has been no collisions for a fixed period of time that you pass as an html parameter.

Once more: if you do this Step, you do not have to submit separate output from the prior Steps, just code and video.

3 General Notes

For each Step, design and document the system, text edit its components into files using Eclipse, compile them, debug them, and execute them on your own test data. When you are ready, submit the text of its code, a copy of your testing, and whatever additional documentation is required. Make sure that the source code is clear and its user interface is comprehensive and informative. For this assignment, we do not require CRC nor UML.

Write your classes clearly, with a large block comment at the beginning describing what the class does, how it does it (i.e., in some form, talk about its design assumptions), and justify how it will be tested. Clear programming style will account for a substantial portion of your grade for both the Theory Part and the Programming Part of this assignment. For one reasonable set of suggested practices and stylistic guidelines (but one that does not refer to the Eclipse defaults), see the website of the textbook used in COMS W1004, at: `http://www.horstmann.com/bigj/style.html`

4 Checklist

Here is an approximate checklist for submission. Please note that this is designed to be a *general* guide, and you are responsible for *all* things asked for in the text of the assignment above, whether or not they appear below. *Please check with Piazza* to see if there are any additional requirements or suggestions from the TAs.

4.1 For the Theory Part

Answers, either handwritten then scanned, or in machine-readable form. No extra points for using a text editor or other formatting tools; hand-drawn diagrams are fine. Submitted electronically in Courseworks by the time specified (i.e., by the beginning of class, exactly). The theory file should be separate from the files for the programming. Name and UNI must be prominently included.

4.2 For the Programming Part

Answers submitted electronically in Courseworks by the time specified (i.e., by the beginning of the review session, exactly). All code (*.java files) documented according to Javadoc conventions and all trial runs. These files should be separate from the files for the theory. Name and UNI must be prominently included in each class (use "@author") and on each output. Rather than providing a separate ReadMe file, one of your classes should have an obvious name that will attract the attention of the TAs (like "Playground.java", for example, since Applets don't have runners), and *inside* it there should be an overview of the components of your system. The classes, in whatever state of completion they are, *must compile*. They must also produce the output that you submit; the TAs will penalize and report any mismatch between what the code actually does and what you say it produced. If you include a video, it must be no longer than 20 seconds, even if it has been deposited somewhere other than your submission.

4.3 For both Parts

Please put all your submission files (theory, *.java, example runs) in one directory (named, for example,"HW5"), and then compress the contents of that directory into one file. Submit that compressed file to Courseworks. The name of that compressed file should be "myUNI_HW5", and it should have the appropriate extension like ".zip", except that "myUNI" should of course be replaced with your real Columbia UNI. *Please* use this convention as it makes the TAs' job much easier:

You may submit electronically as often as you wish, but it will be your *last* electronic submission before the deadline which is the official one that will be graded and (if need be) executed.