COMS W1007 Assignment 4
Due Nov. 21, 2019

1 Theory Part (52 points)

The following problems are worth the points indicated. They are generally based on the lectures about enums, interfaces, Patterns and Java GUIs.

"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 TAs will report to the instructor any suspected violations, and unfortunately this has already occurred this semester. These warnings even apply to `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 Enums (8 points)

Enums come with some unusual methods. Look up, explain, and give an example of what compareTo() does. Explain and give an example of why the two enum methods of name() and toString() are nearly the same, and explain how do they differ. Lastly, there is a "secret" method, values(), that is not part of the API definition! Is this a bug or a feature? How does this method relate to the "varargs" construct (the "three dots") used in passing parameters?

1.2 Patterns: Builder (6 points)

Explain, and give an example, why the Law of Demeter is sometimes called the "use only one dot" rule. Then explain, and give an example, of how the Builder pattern from Assignment 3 uses many dots, but still does not violate the Law of Demeter, and is therefore usually not called a "train wreck".

1.3 Patterns: Iterator (10 points)

Java allows a List to have more than one Iterator object, even within a single method. For example, one iterator can start at the head and proceed forward while another iterator can start at the tail and recede backward. Write a paper program for a method called oneThird() that does the following:

It matches up the elements at the front of the list with two elements at the back of the list. For example, if the list consists of "abcdefghi", it returns "aih" then "bgf" then "ced". You should decide what to do if theList.size() is not an even multiple of 3.

1.4 Patterns: Composite (12 points)

One example of the Composite pattern is the way in which algebraic expressions are computed by a calculator using Reverse Polish Notation (RPN). See: `https://en.wikipedia.org/wiki/Reverse_Polish_notation`. For example, (4/(5-3))*(1+2) first needs to be converted to 453-/12+* by a process you will study if you take a course in Computability or in Compilers. Then the RPN is processed into 42/12+* then 212+* then 23* then 6 (which is a "leaf" RPN).

Give the Java interface for RPN, which has two methods, Compose and Evaluate, where an RPN is really a String[]. The first method returns an RPN when passed some parts, which are two RPNs and an Operator. The second method takes an RPN and simplifies it to a smaller RPN by evaluating it one step. Draw the UML diagram for this Composite pattern, and using ¡¡Stuff¿¿ as a guide. Then, in keeping with "mainline first", assuming all your inputs are error-free, and that all the Strings either represent integers or one of the four characters of a "Four Function Calculator", write the code for Compose and Evaluate.

1.5 GUI (8 points)

The Java API gives seven constructors for the class Rectangle. Notice that they combine ints, Points, Dimensions, and even Rectangle. Some other possible constructors are missing. Using the seven existing constructors as a basis, list *all* the possible missing constructors which would make sense using ints, Points, and Dimension. Note that Dimension is very strange: How is it different from Point? Is there any clear reason why it takes ints in its constructor, but returns double from its getters? Lastly, comment on whether the Builder Pattern would have helped for Rectangle.

1.6 UML (8 points)

Provide the UML for your Programming part, reflecting the structure of what Steps you actually submitted. Hand-drawn diagrams are OK, and there is no extra credit for machine-generated ones. In each class diagram, you *do not* need to include constants, or getters and setters. But you are required to include all other fields and methods. Make sure your class boxes are properly connected with inheritance ("is"), aggregation ("has"), and association ("uses") links, with any necessary multiplicities.

2 Programming Part (48 points)

This assignment asks you to apply the principles of good class construction to a simple interactive console GUI program using the AWT and swing packages of the Java API, and various patterns and interfaces, particularly Composite. We do not require CRC, but we do require UML as one of the questions of the Theory Part, and the usual Javadoc comments within the implementation of your

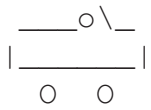
design. You also need to provide test examples, whose output may take the form of screenshots or a short (20-second or less) video. (You can post the video to some depository and then give the URL for it.)

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

2.1 Step 1: Getting started (20 points)

Create a simple GUI display that shows a simple car (with its top down!), moving from left to right at a constant speed in a JFrame. Use the StickFigure code segments on Courseworks as a guide.

The car is stylized as a rectangular body sitting on two circles, with a short line segment representing the windshield, and a smaller circle representing the driver. It sort of looks like the diagram below, except that the body, the wheels, and the driver are all filled in. It will make things much simpler if you use the composite pattern (GeneralPath) to treat the car as a single graphical object.



Download some suggestions for code segments from Courseworks. Make sure you properly attribute the source of this code! As those are only just rough suggestions, make sure you modify the Javadoc and comments appropriately, according to the Boy Scout Rule,

Make sure that the car is drawn using *relative* coordinates, rather than absolute ones, since further Steps will require the reuse of the instances of car with different sizes and locations. This means you need to minimize the use of magic numbers. Instead, you should express all sizes and locations in terms of a UNIT (which could be, for example, the length of the car or the diameter of a wheel) and any mutual relationships (wheels versus body, body versus windshield, relative size of the driver) in terms of CONSTANTS. See the code suggestions on how to organize these x and y values.

To display your output, capture some screen shots, which you include in your zip file, or record with your cellphone a video of no longer than 20 seconds, which can you include in your submission, or deposit in a publicly available forum with the appropriate URL. Please note: If you use a video, if it is longer than 20 seconds, it will be *ignored*, and you will not get any credit for it. You can decently get by with just 10 seconds anyway

2.2 Step 2: Composition and Decoration (14 points)

Using Step 1, make a kind of road race (a composition) that is controllable (a decoration).

This would consists of a small number (from 3 to 7) of cars of random sizes (limited to 0.5 to 2.0 times the usual UNIT), in random places relative to the vertical axis and to each other (again reasonably limited, in terms of UNITs). They can be drawn "on top of each other"; and the filled-in body, wheels, and driver will block any car "in back of" them. But you should still be able to "see through" the open space behind the windshield and in front of the driver. This form of blocking helps establish the illusion of who is "closer" to the viewer.

Generally, it will be more realistic to a have cars that are drawn smaller to be vertically higher, so that they appear to be moving in three-dimensional perspective. If you *really* want to do it right, look up the laws of "one point perspective", for example at: `https://www.wikihow.com/Draw-Perspective`. This enhancement is not required for the assignment, but it is quite simple to implement and adds immeasurably to the illusion of depth.

Then, add a JSlider and its ChangeListener to control this group of cars (this "peloton") so that they all move together as a group in a horizontal direction at the same constant speed. (Note that if you did the one-point perspective, the "farther" cars will *appear* to travel faster, further increasing the illusion of depth.) You might consider using GeneralPath again, to make the construction of this entire group easier, as a single composite of multiple cars. The slider should be displayed horizontally, and it controls the speed, whether to the left, to the right, or even stationary (but be careful here, as you might encounter an ArithmeticException if you attempt to divide by zero).

As usual, if you do this Step, you do not have to submit separate output from Step 1. Again, any videos must obey the restriction of 20 seconds in length.

2.3 Step 3: Creativity (14 points)

Using Step 2, pick *one* of the following enhancements, which should be straightforward once you have the infrastructure above. If you do this Step, you do not have to submit separate output from Step 1 or Step 2.

1. Add a flock of birds (simply drawn as a "v" or "V") that flies toward the viewer. That is, the flock expands in size (but not in count) very time, or every Nth time, that the Timer ticks. The flock can have a fixed direction (left-right, or up-down) and a fixed speed otherwise. The interface defines a flock, which can be an individual bird, a collection of birds, or a collection of collections. Look up how to make this realistic, again using a variation on perspective: the size should not simply get bigger at a uniform rate, but should appear to "zoom" closer, and appear to expand away from a center, sort of like an inflating balloon.

2. Add random up and down bumps for each car, as if they are hitting obstacles. These obstacles "decorate" each car. Then, add to each group of cars larger and wider random up and down bumps, as if they are going over a hill together. These hills "decorate" each peloton. Note that the hills can then have bumps, so using patterns, you can decorate decorations, just as you can compose compositions.

3. Add smoky exhaust to each individual car, which leaves a persistent trail of small ellipses of random sizes behind it. You should look up the data structure called queue, a variation on LinkedList, in order to do this properly in the manner of "mouse trails": the older puffs of smoke should eventually disappear. So, if the car looks like [oo] then it first goes *[oo] then *O[oo] then *O.[oo] then O.@[oo] then .@o[oo], then @o*[oo], etc., as the car moves to the right.

4. This one is quite tricky, but aesthetically pleasing. Modify the apparent geometry of the JFrame to make the GUI window appear infinite, by making the cars get progressively smaller as they approach the border, so that they never really reach it. This is not as hard as

it sounds. This is called a "hyperbolic space", and is sometimes used to display the contents of large databases. See:

`https://www.youtube.com/watch?v=8bhq08BQLDs`

or look up Escher's woodcuts in his "Circle Limit" series to get a general idea, at:

`https://www.google.com/search?tbm=isch&q=escher+circle+limit`,

or you might want to Google up "Poincare disk model wiki". The interface here would generalize the concepts of drawing something at (x, y), depending on whether the geometry is the usual Euclidean one, or a hyperbolic one in the y dimension (it will look like a cylinder on its side), or a hyperbolic one in both x and y dimensions (it will look like a sphere).

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, which is fundamentally about GUIs, AWT, and swing, we do not require CRC, but we do require 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 class, 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 "HW4Runner", for example), and *inside* it there should be an overview of the components of your system. That class should contain instructions on where to find and how to run the video. 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.

Please note: 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,"HW4"), 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_HW4", 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.