Nick Troccoli Midterm Exam

CS 106A July 24, 2017

**CS 106A Midterm Exam – Question Booklet**

This is an open-textbook, closed-note exam. You may refer to the Karel the Robot Learns Java reader and the Art & Science of Java textbook, but you may not use any printed paper resources or computing devices of any kind, including calculators, laptops, cell phones, or other devices.

Unless otherwise indicated, you will be graded on functionality only – but good style helps graders understand what you were attempting. **Remember that you must write the solution to a problem in that problem’s provided answer pages in the answer booklet. Answers to a problem outside of its designated answer pages in the answer booklet, or written in this booklet, will receive no credit.** You do not need to write any import statements. Unless otherwise specified, you may write helper methods to implement required behavior if you like. Do not abbreviate code; you must write out all code that is part of your answer. If you want to utilize code found in the textbooks, you may cite the page number and code sample without rewriting that code. Pay attention to whether a question requires you to write a complete Java program, or just individual methods.

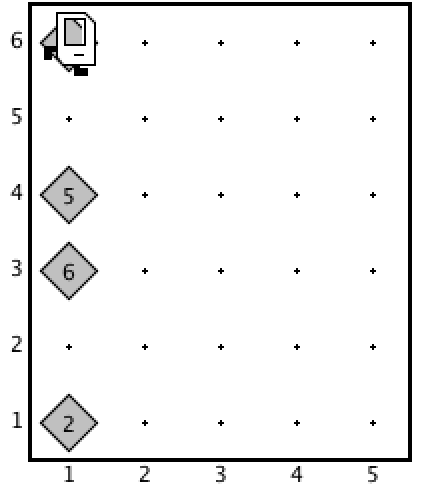
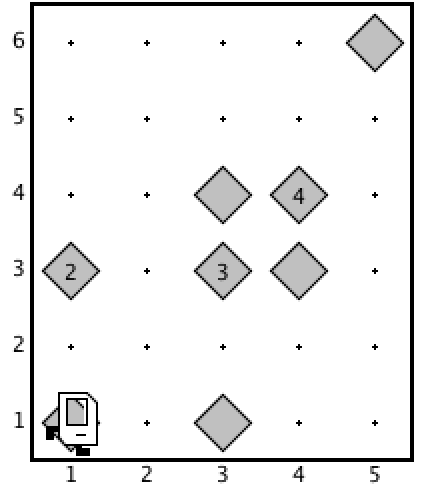
A “Syntax Reference Sheet” of common commands can be found at the end of this booklet.

You have 2 hours to complete this exam.

|  |  |
| --- | --- |
| 1) Karel the Robot………………………... | [20 points] |
| 2) Java Statements and Expressions…….. | [20 points] |
| 3) Console Programs……………………... | [25 points] |
| 4) Graphics Programs……………………. | [20 points] |
| 5) Text Processing………………………… | [35 points] |
|  | Total:120 points |

Problem 1: Karel the Farmer [20 points]

Karel has recently started a new job as a farmer, and needs you to write a program called **FarmerKarel** to help it gather up crops (represented as beepers, of course). Karel starts off in a world of any size, with crops (beepers) scattered around this world. Your program should have Karel, for each row in this world, gather up the beepers in that row and place them on the *leftmost square of that row*. Here is a before-and-after example:

**** *before: after:*

Note that, in each row, Karel has gathered all beepers in that row and placed them on the leftmost square. If there are no beepers in a row, Karel should not place any beepers in that row.

You may assume the following facts about the world:

* Karel starts at (1, 1) facing East, with an infinite number of beepers in its beeper bag. This means Karel cannot put down exactly the number of beepers it previously collected.
* The world may be *any* size, and your program should work for all world sizes.
* There may be any number of beepers, or no beepers, on a given square, and beepers may be placed anywhere in the world.
* There are no walls in the world.
* Karel’s ending location and direction do not matter.

Note that you are limited to the Java instructions shown in the Karel reader. For example, the only variables allowed are loop control variables used within the control section of the for loop. You are *not* allowed to use syntax like local variables, instance variables, parameters, return values, Strings, return or break, etc.

**Problem 2: Java Statements and Expressions [20 points]**

*Write your answers to the questions below in your answer booklet – DO NOT WRITE HERE!*

1. For each expression in the left-hand column, indicate its value. Be sure to list a constant of the appropriate type (e.g. 7.0 rather than 7 for a double, Strings in double quotes, chars in single quotes, true/false for a boolean, etc.).
   1. **1 + (2 + “B”) + ‘A’**
   2. **11 / 2 > 5 || 5 % 2 == 1**
   3. **(char)(‘B’ + 2) + “” + 4 + 27 / 3**
   4. **21 / 2.0 + 3 % 4 – 23 / 2**
   5. **!(3 / 2 < 1.5) && (4 > 5 || 2 % 3 == 0)**
2. What are the color, dimensions and location of **rect** on the canvas?

public class Problem2b extends GraphicsProgram {

public void run() {

int num2 = 13;

int num1 = 9;

int width = mystery1(num2, num1);

int height = mystery1(5, num2\*3);

GRect rect = new GRect(0, 0, width\*3, height);

rect.setFilled(true);

rect.setColor(Color.BLUE);

mystery2(rect, num1);

add(rect);

}

private int mystery1(int num1, int num2) {

num2 += 3;

String str = "Hello " + num1;

int num3 = num2 - str.length();

return num3;

}

private void mystery2(GRect otherRect, int num2) {

otherRect.setLocation(num2, num2);

otherRect.setColor(Color.RED);

}

}

Problem 3: Movie Kiosk [25 points]

A local movie theater has hired you to write a ConsoleProgram for their ticket purchase kiosks called MovieKiosk that continually prompts the user for movie name, # tickets and price per ticket, and when they are done outputs the names of all movies they want to see, as well as the total cost (see input at right).

Movie name: **Cars**

# tickets: **2**

Ticket price: **14.50**

Movie name: **WALL-E**

# tickets: **4**

Ticket price: **14.00**

Movie name: **Up**

# tickets: **3**

Ticket price: **10.50**

Movie name: **[ENTER]**

Movies: Cars and WALL-E and Up

Total: $116.5

The program should stop prompting the user once they submit ENTER for the movie name. You can assume that if they enter a movie name, they will always enter a valid ticket quantity and price. At the end, you should print out all entered movie names, as well as the total price of all purchased tickets. You do not have to worry about the number of digits displayed for any numbers. If no movies are entered, you should print out “Movies: None” and not print out the total price.

For movie names, print “Movies: ” followed by all movie names joined with “ and ”. For instance, if the user enters “Cars”, “WALL-E” and “Up”, you should print out “Movies: Cars and WALL-E and Up”. If they enter “Cars” you should print “Movies: Cars”.

dfdfdfdfdfd

Movie name: **Cars**

# tickets: **2**

Ticket price: **14.50**

You have won a $10 voucher for your next purchase!

Movie name: **WALL-E**

# tickets: **2**

Ticket price: **3.50**

Movie name: **Up**

# tickets: **2**

Ticket price: **10**

Movie name: **[ENTER]**

Movies: Cars and WALL-E and Up

Total: $49.0

The movie theater would also like you to award randomly-chosen customers *vouchers* (credits) towards their next movie to encourage future purchases. Specifically, every time the user selects tickets for a movie (after entering the price per ticket for a movie), there is a 10% chance this user gets a voucher towards their next entered movie (if there is one). The voucher should be for a random integer dollar amount between $5-25.

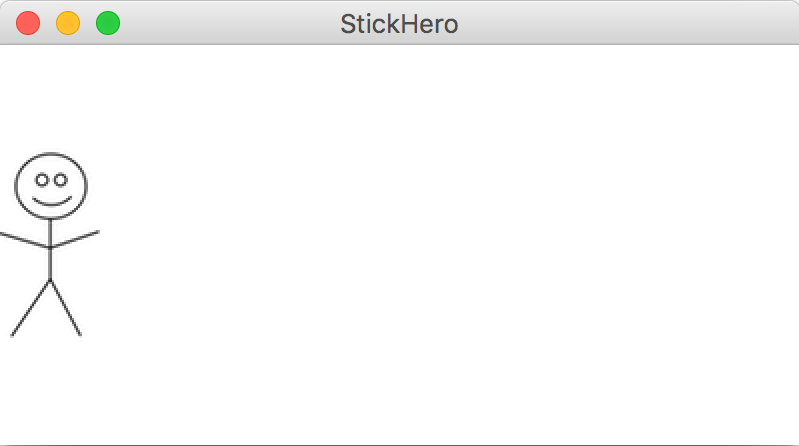
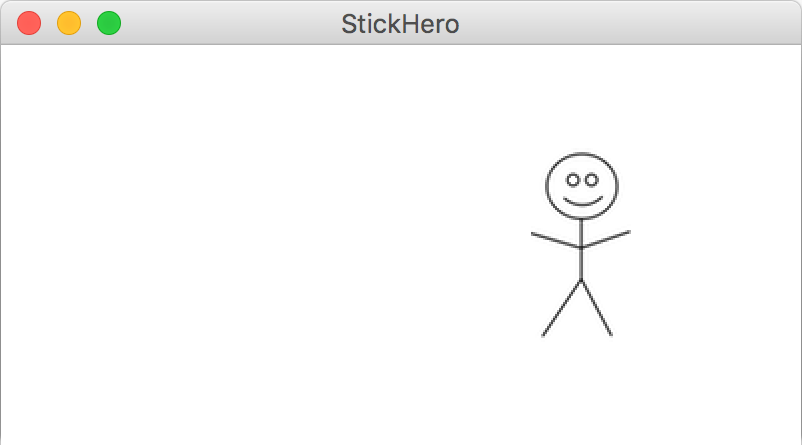
On their next purchase, if the voucher amount is *less than* the total for that movie, simply deduct the voucher amount from that total. If the voucher amount is *greater than or equal* to the total for that movie, then they get those tickets for free, and the remainder disappears. For instance, at right, the user receives a $10 voucher for *WALL-E* after purchasing *Cars* tickets; they only spend $7 on *WALL-E* tickets, so they get those tickets for free and the remaining $3 disappears; it does not carry over to their *Up* ticket purchase. If, in this example, the user instead spent $14 on *WALL-E* tickets, then after the voucher is applied only $4 would be added to their total.

*Hint: the RandomGenerator method nextBoolean takes a double parameter which is the percent chance of returning true.*

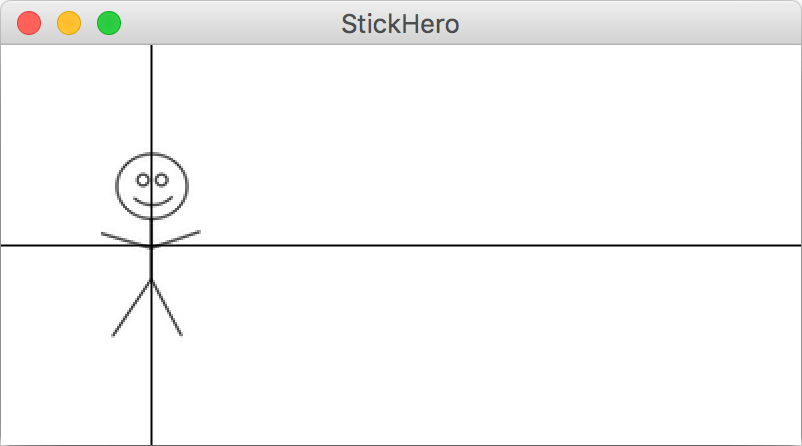
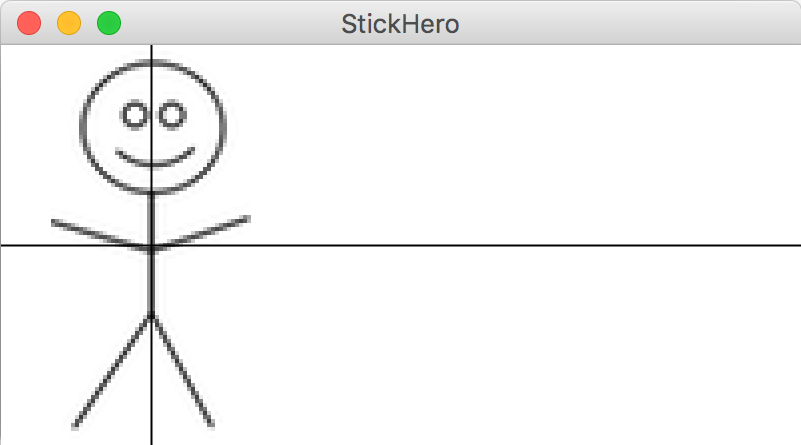
Problem 4: StickHero [20 points]

Your friends are working on a new video game and, knowing that you are in CS 106A, have enlisted your programming help for the main character, StickHero, who has the superpower to switch between “double size” and “original size”. If the user clicks on StickHero when it is original size, it will go to double size, and vice versa. The player image, player.png, has been provided by your teammates in the res folder; original size is defined to be the size of this image, and double size is defined to be double the dimensions of this image.

Write a GraphicsProgram called StickHero that starts with original-size StickHero at x = 0, centered vertically. When the program starts, StickHero should start animating to the right 5 pixels each iteration, with a delay of 30ms.

 *initial state after a few seconds*

As soon as any part of StickHero goes off the right edge of the screen, it should go back to its starting position at the left side of the screen and continue animating to the right. If, at any point, the user clicks on StickHero, it should toggle between double size and original size. Notably, however, the x/y center point of StickHero should never change.

 *original size double size*

As shown above, if you mark StickHero’s center (note: these lines do not actually appear onscreen), it remains unchanged as you toggle between original and double size.

Your program should work for any screen size and any reasonable size of player.png (e.g. that fits onscreen). You do not need to account for the screen resizing after the program launches.

Problem 5: Mentions [35 points]

After finishing up their StickHero video game, your friends have now moved on to working on a new social network. They need you to help them implement mentions, as described below. *Note that this is a two-part problem where you will implement a single method and a ConsoleProgram.*

1. Write a method replaceMention that accepts a string as a parameter and, if it is a *mention*, returns a new string that is the mention’s *expanded name*. A *mention* is defined as an ‘@’ symbol followed by 1 or more upper-camel-cased names; upper-camel-case means the first letter is uppercase and all other letters are lowercase. For instance, @NickTroccoli, @Nolan and @AleksanderPaulDash are all valid mentions, But @ and @nicktroccoli are not valid mentions.

The *expanded name* of a mention is all the upper-camel-case names in the mention with a space in between them, except for the *last* name in the mention, which should be abbreviated with only its first initial and a period. However, if the mention contains only 1 name, the expanded name is just that name (without the ‘@’). The following table shows some sample inputs and outputs to the replaceMention method.

|  |  |
| --- | --- |
| Call | Value Returned |
| replaceMention(“@NickTroccoli”) | “Nick T.” |
| replaceMention(“@AleksanderPaulDash”) | “Aleksander Paul D.” |
| replaceMention(“@Nolan”) | “Nolan” |
| replaceMention(“hello!”) | “hello!” |

The input is guaranteed to be a single token without any spaces, and is guaranteed to be entirely a mention, or not contain a mention at all. Note that if the input is *not* a mention (including the empty string), the output is simply the same as the input.

1. Write a ConsoleProgram called ReplaceMentions that prompts the user for a valid text filename in the res folder, and prints out that file to the console with all of the mentions replaced with their expanded names. You should reprompt the user until they enter a valid filename. If an error occurs reading the file, print out an error message. You may assume that the text file specified has only 1 line of text, and that each word in the file is separated by a single space. For example, if the file myinput.txt contains the following text:

Head TA @RishiPaulBedi - friends with @Nick - rocks socks!

Then the output of the ReplaceMentions program would look like the following (user input bolded and underlined):

Filename: myinput.txt

Head TA Rishi Paul B. - friends with Nick - rocks socks!