Make a Movie

Contents

1 Introduction

Submit a **zipped folder** that is **only** containing your Java source files (*.java) in course's Black Board.

Please use the following naming convention for the submitted folders:

YourPSLetter_CourseCode_Surname_Name_HWNumber_Semester
Example folder names:

- **PSA_COMP130_Surname_Name_HW4_F19**
- **PSB_COMP131_Surname_Name_HW4_F19**

Additional notes:

- Using private methods are important, **failing** to do so will be **penalized**.
- Using the naming convention properly is important, **failing** to do so will be **penalized**.
- **Do not** use Turkish characters when naming files or folders.
- Submissions with unidentifiable names will be **disregarded** completely. (ex. "homework1", "project" etc.)
- Please **write your name** into the Java source file where it is asked for. **Failing** to do so will be **penalized**.
- If you are resubmitting to update your solution, simply append **v#** where # denotes the resubmission version. (i.e. **v2**)

1.1 Academic Honesty

Koç University's *Statement on Academic Honesty* holds for all the homework given in this course. Failing to comply with the statement will be penalized accordingly. If you are unsure whether your action violates the code of conduct, please consult with your instructor.

1.2 Aim of The Homework

The aim of this project is to allow you to practice coding, testing and debugging using graphics, animation and Strings.

In this project you are expected to create a **Command Line Interface (CLI)** that will allow the user to create animations. This project also expects you to display the created animation to the user.

1.3 Given Code

You are provided a code which has the necessary setup. It contains helper methods; variables and constants for you to start with.

Do not change anything in the code if it is indicated to you with a comment. The code given to you has something called **JavaDoc** comments above all the

methods. These comments allow you to view various information about the method when you mouse over the name of the method. Below are the methods given to you in the code with their explanation.

1.3.1 Given Methods

There are methods provided for each part.

- **void init()**

  This method is implemented in the Graphics Program itself and is guaranteed to be called before the void run(). In this project, this method is used to print the welcome message to the console. Do not modify this method.

- **void run ()**

  This is the entry point for your program. Feel free to modify it as you wish.

1.3.2 Given Variables and Constants

There are some variables and constants provided to you. Feel free to use the variables declared in the given code, meaning you can either use them or create new ones if you need to. HOWEVER, you need to **use the constant variables** provided to you. You can create additional constants if you need to.

All constants provide **JavaDoc** comments above them. Please read these to understand what constant is used for what. **Do not** use another variable or a static value for something if there is a constant variable defined for that purpose.

1.4 Further Questions

For further questions **about the project** you may send an email to **course SLs** at **(comp198-fall19-sls-group@ku.edu.tr]** and **Ayca Tuzmen** at [**atuzmen@ku.edu.tr**]. Note that it may take up to 24 hours before you receive a response so please ask your questions **before** it is too late. No questions will be answered when there is **less than two days** left for the submission.

2 Project Tasks

The project tasks are divided into subsections to make it easier to understand and implement. You are strictly advised to follow the given order of tasks, however should you choose not to follow the tasks in order, it is possible to implement the project in any order.

2.1 Creating a Command Line Interface

A Command Line Interface (CLI) is a means of interacting with a program where the user issues commands in the form of successive lines of texts. In this task you are expected to create a simple CLI that needs to be capable of recognizing the following commands:

- **setback:** Sets the background for the canvas
- **setgrammar:** Sets the grammar for the project.
- **addscene:** Adds a scene to the current project.
- **removescene:** Removes a scene from the current project.
- **listscenes:** Lists all the scenes in the project.
- **play:** Play the scenes in order
- **exit:** Exits the program

2.1.1 Building a Basic CLI

In its most basic form a CLI reads the user input as a string and parses it to call different commands according to the input text. You may want to start by implementing a simple code that asks for user input continuously. Once you have accomplished this, try detecting the input text of "exit". Modify your code so that it asks for a new input from the user as long as the input is not "exit".

Advice: You may want to echo (print back to the screen) the input you receive from the user to clearly see that you can read the input.

2.1.2 Detecting All Commands

Now that you have a basic CLI that is capable of recognizing a specific input, you can proceed by detecting all the commands shown in Section 2.1.

Hint: You can implement temporary methods for all the commands you have detected and for now print a text that states the command is executed properly.

2.1.3 Distinguishing Commands From Output

If you have implemented the suggested way of checking your **CLI**, you may have noticed that the input and the output lines are mixing up. To prevent this confusion you are expected to print a constant string before taking an input from the user. This is the string defined as **CLI_INPUT_CONST** in the given code.

Hint: You will not need to animate anything else after this point.

2.2 Creating a New Movie

In this task you will create a new movie from scratch. You will define the grammar to be used in creating the movie. You will start adding scenes to the movie using the grammar you defined.

2.2.1 Defining the Grammar

The grammar allows the user to change the way they want to describe the scene. As a real world example, you can see that Turkish and English have different grammars. Below is a list of all grammar descriptors defined for this project.

- **scale:** A single word describing the scale of an image. Example: 1.8
- **image:** A single word that is the name of a image file. Example: fish
- **from:** A two word descriptor that starts with the keyword **from** and is followed by a location. Example: from left
- **to:** A two word descriptor that starts with the keyword **to** and is followed by a location. Example: to right
- **time:** A three word descriptor that starts with the keyword **for** and is followed by an integer value which is followed by either the word **second(s)** or **millisecond(s)**.

  Note: The locations described in the **from** and **to** descriptors can take the following values: **left, right, top, bottom, center**.

Below are examples of grammar definition and corresponding usage of the grammar.

[time scale image from to]
for 5 seconds 0.5 car from left to right
[from to scale image time]
from left to right 0.5 car for 5 seconds
[scale image time to from]
0.5 car for 5 seconds to right from left
[from time scale image to]
from left For 5 seconds 0.5 car to right

2.2.2 Creating a New Project, Actually.

In this task you are required to start implementing the commands that your CLI can detect.

You may start with setting the background for the movie. The **setback** keyboard will ask the user for the name of the color to be used. The colors that can be used for the canvas are **White, Green, Blue and Magenta**.

You may proceed with the **setgrammar** command. This command will take the ordering of the grammar descriptors defined in Section 2.2.1 from the user. You will later on use this to parse the animation scenes.

Hint: You may want to save the grammar to either a string, an array of strings or something else as it is later on required.

As you have acquired the grammar specification, you may now allow the user to start inputting new scenes. Implement the **addscene** command so that the user can add a scene by describing it according to the grammar that they defined. This command will read a single line of user input.

Hint: String class has a method split(String str) which splits the string it is called on and returns an array of strings that were originally separated by str. (Example: "Thisisateststring".split("") will return a string array of "This",

"is", "a", "test", "string") Note that the string the method is called on can be a variable as well.

After an excessive usage of addscene command a user may lose track of the scenes added, and may want to see what is currently in the project. At this point you are expected to implement the **listscenes** command that will list all the current scenes in order, starting their enumeration from 1. (Meaning the first scene is the scene number 1)

Once seeing all the scenes present in the project, a user may want to remove a scene from the animation. Notice how you enumerated the scenes in the listscenes command, you are now required to implement **removescene** command. This command will ask the user to input a scene number, and will remove it from the project. After a removal, the enumeration of the scenes may change if the scene removed was not the last scene. (In an animation with four scenes, the removal of second scene would make the third the second and the fourth the third.)

2.3 Making Your Film (Where the Magic Happens)

Up until now, you worked with the boring infrastructure of the project. Now is the time for you to become the director of your (or maybe someone else's) film! The tasks below will guide you through the process of animating the scenes.

2.3.1 Creating an Animation Loop

An animation consists of at least one scene. However one scene is hardly enough to express anything meaningful. You may find it beneficial to create a loop that iterates over all the scenes, before starting the animation part itself. In the next task you may simply fill in this loop to actually animate all the scenes.

2.3.2 Animating the Scenes, Actually

Animating a scene is nothing more than extracting meaningful information from the scene according to the grammar and applying your graphics knowledge to it. You are expected to load and scale the image file in the scene according to the scale factor given in the scene description.

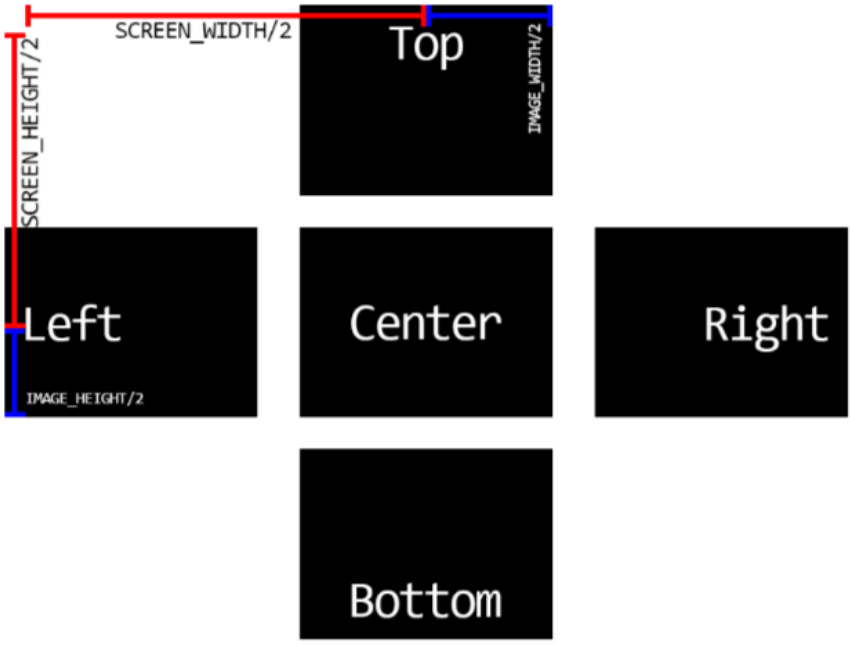
Image files reside in the "lib/" directory of your project folder. The path to this folder is specified to you as a constant at the end of the project file. To load a file from this path, append the path to the beginning of the filename.

Once you scaled correctly the image file, you can add it to the location specified by the from descriptor. Note that you are expected to place the image in the **center** of the specified location. (i.e. placing an image to the left means that the middle of the left edge of the image must coincide with the middle of the left edge of the screen.)

Now that you have the image on the screen, you need to move it to the location specified by the to descriptor. Note that this movement must be uniform and must take exactly the amount of time specified by the for descriptor.

2.4 End of Project

Your project ends here. You may continue to tinker with the code to implement any desired features and discuss them with your section leader. Below in the



Section 3 are two further tasks for you to implement if you are willing to continue practicing the topics. **Do not** include any additional features that you implement after this point in to your submission.

Final Warning: Do not include anything beyond this point to your submission. Points may be deducted from your grade as Section 3 alters the normal behavior of the simulation.

3 Bonus Task

Feel free to complete the bonus section if you would like to get extra points from this homework. You can add additional commands into the grammar or to the project which is not required but only will be used as an optional command.

4 Further Tasks

Tasks described in this section are not supposed to be included to your project, but are provided for studying the topics further. **Do not** submit your project with any of these tasks completed. You will only be graded for the tasks in **Section 2 and 3**. Also note that tasks below are meant to be implemented on their own but may function together as well.

4.1 Understanding Coordinates

Modify your program so that it can also understand a location given the XY value in the following format: (X, Y). When a coordinate is specified in this format, place the center point of the image to the given coordinate.

4.2 Quality Assurance

> A QA engineer walks into a bar, and orders a beer.
> Then he orders 0 beers.
> Then he orders 999999999999 beers.
> Then he orders an aardvark.
> Then he orders nothing.
> Then he orders -1 beers.
> Then he orders null beers.
> Then he orders asnwikfjsdf.
> Finally, the QA engineer leaves without paying,
> comes back, and asks for the tab.

The excerpt above is a funny analogy of the life of an quality assurance engineer. Programs are tested with a wide range of both sensible and senseless inputs. The goal is to fix the program so that there exists no unexpected behavior. Apply the same procedure to your program and try to handle all the errors that may be caused from illegal inputs. An example of this may be only allowing scenes that are correct according to the given grammar. Another example can be handling an error raised by trying to open an file that does not exist.