Silly Premise

During a cool October night, Dwight scours the internet for a video to pass on the Schrute family farm legacy to his chosen successor, Mose. But of course no video exists that can encapsulate all the intricacies and beauty of the Schrute beet farm. The Schrutes come from “a long line of fighters”. Dwight himself is fast, between “a snake and a mongoose...And a panther.” After throwing his computer across the office in frustration, Dwight realizes he of course must take matters into his own hands and make an adequate cartoon for the Schrute family farm. The

---

1 Note that these demos were done in an older Java graphics package (Swing), so elements may show up differently if you try to replicate specific parts.
world is not ready for his undivided attention. Good thing for Dwight, who is such a hard worker, he knows that he should start early, start today, start yesterday on his Cartoon and therefore he left himself with plenty of time to finish his Cartoon!

Get ready! Make the best cartoon Scranton has ever seen!

**Collaboration Policy Reminder**

From the [collaboration policy](#):

Collaboration on project design is not allowed, except for the conceptual topics on mini-assignments. To plan design, you will also have a one-on-one check-in with one of your section TAs. Otherwise, **no collaboration is allowed on project-specific details.** You may not discuss implementation or debugging of code for projects with anyone except the course staff.

**Importantly**, though, you may absolutely discuss general (i.e., not most assignment-specific concepts) CS15 concepts with anyone, including other current students. The following falls into this category:

- Non-coding questions that are explicitly asked on the mini-assignment
- Going over CS15 lecture slides, our (non-assignment) handouts, Javadocs, etc.
- Discussing object-oriented programming concepts, such as polymorphism
- General syntax questions. For example, “How do I declare an instance variable?”
- How to work remotely, and how to move and hand in files

Note that, when discussing topics that aren’t on the mini-assignment, any examples used must be from the lectures or your own creativity – you may not freely discuss how even broad design concepts, like containment, pertain to a specific assignment if they are not asked on the mini-assignment.

**New Concepts Covered**

- GUIs (Graphical User Interfaces)
- Graphical Containment
- Layout Management
- Complex association and sharing references
- Events and event handling
- JavaFX
- Refactoring code using helper methods
Assignment Specifications

Here’s your chance to be creative with JavaFX. While we require that you implement some standard base functionality, we also want you to be creative with this. **This means you may not simply make the alien from the lecture slides.** The assignment specification, at a high level, is to create a graphical user interface that accomplishes a few small tasks and uses a composite shape (a class that contains more than one instance of a `javafx.scene.shape.Shape`).

After you have the base specs working, use your imagination, artistic ability (let's see it, you VISA majors!), and programming skills to make something that wows your friends. As with all projects, you are expected to follow CS15 design conventions. This includes the use of a top-level `PaneOrganizer` class (see “Programming Tips” section and Help Slides posted online), constants, and helper methods as needed.

To provide some direction, we require the following elements as minimum functionality. Before you even start thinking about extensions to this assignment, make sure you have the following:

- A `javafx.animation.Timeline` that animates your Cartoon.
- A composite shape that moves based on the `javafx.animation.Timeline`
  - Composed of at least 5 shapes, using at least two different types of `javafx.scene.shape.Shape`.
  - Some of the classes you could use are `Circle`, `Rectangle`, and `Polygon`.
- A `javafx.scene.control.Label`
  - For **minimum functionality**: your Label must change based on the movement or location of your composite shape. However, your Label can be updated either based on the timeline or with a key input. This means your label can change when the user presses a key or when your `Timeline` updates.
  - For **full credit**: the text content of your Label should **change** as the **timeline is updated**. The Label should be based on the movement or location of your composite shape that is directly dependent on your timeline. This means that the composite shape your Label is based on cannot only move with key input. If your Label **only changes with key input**, your grade will receive a deduction. The Label must be updated with your `Timeline`.

---

2 Be creative with how the label can achieve this—don’t just increment a counter when a key is pressed.
• A KeyHandler (or equivalent) class that implements EventHandler<KeyEvent> and can visually change an element of your cartoon based on keyboard input (think moving shapes, changing colors, etc.) without quitting.
  • i.e. your program must be set up so that pressing a button on the keyboard causes a visible change in the cartoon.
• A Pane that contains your composite shape.
• The use of the javafx.scene.layout package so that your components line up nicely within your GUI.
  • Some useful layout containers are BorderPane, VBox, HBox, FlowPane, and GridPane classes.
• A javafx.scene.control.Button that quits your program properly.
  • This button will need an EventHandler that calls System.exit(0) or Platform.exit(), which exits the program.
• A README file, including what key inputs affect the cartoon and how the Cartoon is personal to you.

Recommended: Personal Connection

This is the first project that’s completely your own - make it personal! We’re not giving you any support code so when you’re finished you’ll have a project you made completely from scratch. We’d love to see cartoons that mean something to you. You can make a cartoon of your pet, your childhood home, your favorite vacation spot, your favorite childhood toy, your favorite food, your favorite childhood cartoon character, or literally anything else you love. Make sure to tell us about the personal connection in your README and how your cartoon features it! When Andy chooses to showcase his favorite cartoons, he loves cartoons with personal significance! *wink*

Programming Tips

JavaFX
The stencil code we give you consists of just an App that takes care of the mainline. It’s your job to override and fill in the public void start(Stage stage) method. Here you can instantiate your top-level PaneOrganizer. Also, make sure to set a new scene and pass in your root pane.

Cartoon class
In the past, our top-level class has been LiteBrite or FruitNinjaFrame. Now, the top-level class is PaneOrganizer. However, PaneOrganizer should really only be responsible for graphical elements that affect your entire application. Imagine an application that
had several different games happening at once—it would be a lot of work for the PaneOrganizer to keep track of every single shape in every single game.

For the Cartoon project, your PaneOrganizer should not be responsible for controlling your shapes and animation. Instead, PaneOrganizer can delegate most of the responsibility for animation and user input to another class—your Cartoon class! See the Help Slides online for more on what this class contains.

**Working Incrementally**

It is a very good idea to code this assignment *incrementally*, making sure it compiles and does what you expect at each step. **Coding incrementally will save you a significant amount of time on this project.**

Feel free to work in any order that works for you, but the Help Slides break down one way to approach the project in chunks so that you can get small parts working with each step.

**JavaFX**

Cartoon is your first project without support code written by the TAs—congratulations! From here on out, you’ll be using lots of JavaFX code. Just like support code, all of this is written for you, so you don’t need to worry about how any of this is implemented; all you need to do is call methods as they are defined.

We *highly* recommend that you review the lecture slides of [Graphics I](#), [Graphics II](#), and [Graphics III](#), as well as read through [JavaFX Shapes Documentation](#) and [CS15 JavaFX Guide](#) before you start. Refer to these lectures and additional resources as you work on your cartoon. **Make sure you understand the lecture slides and the JavaFX Shapes Documentation before you begin working on the project!** Feel free to refer to CS15 JavaFX Guide if you need more examples, or need some guidance in adding bells & whistles to your cartoon. The JavaFX Guide provides all of the explanations of the classes, methods, and common pitfalls that you need to know for Cartoon. **Reading these documents will save you a significant amount of time when coding and debugging.**

Once you have completed it, the JavaFX lab will also cover many of the concepts and code that can jumpstart your work on this project, so we would recommend completing that lab before you start coding for Cartoon.

**Keyboard Interaction:**

You may be wondering how you will make your cartoon respond to user input. The answer is keyboard interaction! There are a few ways to do keyboard interaction in Java, though we suggest the method below. (Feel free to explore [Javadoc](#) for other ways to go about it, as long as they work well and are readable).
The method we recommend makes use of the interface `javafx.event.EventHandler`. You need to define a private inner class which implements this interface and decide what you want the `handle(Event e)` method to do.

To get information on which key is being pressed, first your `handle(Event e)` method should take in a `javafx.scene.input.KeyEvent`. Then, you need to call method `getCode()` on the `KeyEvent` which will report which keyboard key (`KeyCode`) has been pressed. Read more about KeyEvents [here](#) (jump to the section on “KeyBoard Input”).

For a complete list of the types of `KeyCode`s, you can check the [Javadoc](#). Once you know what keyboard key was pressed, you can make a decision (hint, hint... “Math and Making Decisions” lecture slides) on what you want to do when a specific key is pressed.

Let’s define an `EventHandler` to move a ball in space. We’ll write it as a private inner class in the `Game` class:

```java
private class MoveHandler implements EventHandler<KeyEvent> {
  @Override
  public void handle(KeyEvent e) {
    KeyCode keyPressed = e.getCode();

    if (keyPressed == KeyCode.UP) {
      // Code to make the ball move up.
    } else if (keyPressed == KeyCode.DOWN) {
      // Code to make the ball move down.
    }

    e.consume(); //**See explanation below.
  }
}
```

**A few important notes to keep in mind regarding keyboard interaction:**

JavaFX has some built-in functionality for certain `KeyEvent`s. For example, the down arrow key moves the input focus to another node in the scene graph. Depending on your Cartoon implementation, this may mean that pressing down would move the focus out of the `Pane` that holds your cartoon and into another, which can cause all sorts of issues! There are two steps that you should take to fix this:

1. Call the `consume()` method on the `KeyEvent`
One way to make sure the program only executes what you indicate in the `EventHandler` is to “consume” the event, which you can think of as throwing away the event after it has done everything you need it to do. To do this, call the `consume()` method on the `KeyEvent` at the end of the method you use it in. You can read more about this [here](#) (the section under “Consuming of an Event” at the bottom of the page.)

2. Call the `requestFocus()` and `setFocusTraversable(true)` methods on the relevant Panes

   We can also explicitly manage where focus is set. To do this, you’ll need to make sure to call `requestFocus()` and `setFocusTraversable(true)` on the Pane that listens to your KeyEvents. To make sure no other node (in this case, the quit button) grabs focus inadvertently, you can call `setFocusTraversable(false)` on each one. You can read about this method [here](#).

Remember to associate your `EventHandler` with the relevant Pane (hint: `setOnKeyPressed(keyEventHandler)`) or nothing will happen when you press the arrow keys!***

More JavaFX Resources:

You can check out the [Javadocs](#) for loads of JavaFX Documentation -- these are great resources for finding the methods and properties of JavaFX elements! However, the [CS15 JavaFX Guide](#) provides a lot of implementation & ideal design details that the JavaDocs don’t cover in as much depth, so use the JavaDocs to supplement the CS15 JavaFX guide, rather than to replace it.

For even more resources, you can check out this [tutorial](#) that Oracle provides on getting used to JavaFX. “Hello World, JavaFX Style” and “Form Design in JavaFX” will help you learn the basic layout design. Also, you can take a look at “Animated Shapes and Visual Effects” to learn how `Timeline` is used, but you do not need to get as fancy in your Cartoon project.

Bells and Whistles (Extra Credit)

If you want to add extra credit to your assignment, take a look at the snazzy demos for inspiration. Additionally, here are some possible ideas, but you may certainly come up with your own (though see the footnote on adding audio first).³

³ Note: Because many of the department computers do not have the necessary libraries to play audio, we cannot give extra credit for it and it thus can’t be used for Bells & Whistles.
● Include other JavaFX elements, like `javafx.scene.control.Slider`, `javafx.scene.control.ColorPicker`, `javafx.scene.control.Spinner`, or some other element to interact with your Cartoon in even more ways. The possibilities are endless!
● Find a way to have mouse AND keyboard interactions!
● Add images! Read more here.
  ○ Note: if you are adding an image, use "file:./Cartoon/<image_name>" so that it runs properly when we are grading.
● Make your Cartoon out of 3D shapes. You can read about how to use them here.
● Make fancy transitions! There are many transition animation effects provided in JavaFX such as `FadeTransition`, `ParallelTransition`, `PathTransition`, `RotateTransition` and so on. Read more here.
● Add some cool effects to your shapes/text! See the Javadoc for possible effects.
● Make your shape move at different speeds, by adding in an acceleration variable! The movement must be based on a Timeline, but the changes in speed can happen based the Timeline or based on another factor (e.g. keyboard input). Manually setting (hardcoding) speeds will not count.

Remember that you should make sure that you have a fully functional program before working on extra credit. Remember, from the Course Missive:

“Extra credit is only to be done after the original assignment has been fully completed - if you have not met the requirements, you will not receive extra credit. Extra credit may not redefine the original assignment. You are entitled to one “free” late pass during the semester. The late pass allows you to turn in one program by the late date without penalty, though you will still not be eligible for extra credit on the assignment.”

**README**

Since designs for Cartoon are unique, it is going to be extra important that you write a clear README that explains how to interact with your Cartoon, and states the design choices that you made. Your visual changes and inputs should all be explained in the README. Remember, the TAs look over your README before grading Cartoon, so make them happy and explain all significant design choices.

You are expected to create your own README file. Please refer to the README guide for information on what information your README should contain and how you should format it.

**Handing In Your Code**
In order to hand in your code, run the handin script at the top of this handout. Make sure you have your containment diagram either in PDF format or in your README. This is required to help the TA grade your code. The script will list all of the files you are about to hand in and will prompt you to confirm. Once you’ve confirmed, you will receive an email stating that the handin was successful.

As usual, your program must be fully commented. This means we are expecting to see header comments for every class and on any methods you write, but you do not need to comment saying something like “//here I am declaring a private instance variable of type BorderPane” – we trust you know what you’re doing. See the Style Guide for more information.

---

Minimum Functionality Requirements

MF Policy Summary: In order to pass CS15, you will have to meet minimum functionality requirements for all projects. If you don’t meet them the first time around, you may hand the project in again until you succeed, but you will keep your original grade. MF requirements are not the same as the requirements for full credit on the project. You should attempt the full requirements on every project to keep pace with the course material. An ‘A’ project would meet all of the requirements enumerated in the assignment specification section of the handout and have good design and code style.

To meet minimum functionality for Cartoon:

- A text label that changes text based on the movement or location of your composite shape in the Cartoon. A label that changes when you press a key or a label that changes with the Timeline are both acceptable.
- A composite shape that moves graphically based on a Timeline. (Note: the shape must be logically composite in the code, like the alien in the JavaFX lectures).
- An element in your Cartoon visually changes based on a KeyEvent, with the key inputs explained in your README.