Cartoon

Help Slides

Design Check-in\(^1\) **Mini-Assignment** Due: Thursday, October 13 at 2 PM

TA Check-ins: Thursday, October 13 - Saturday, October 15

If one of your design discussion TAs hasn't already reached out by email to schedule a check-in, please contact the HTAs by noon on Wednesday (10/12) at the latest to coordinate scheduling

**Early Handin:** Tuesday, October 18th, 11:59 pm

**On-Time Handin:** Thursday, October 20th, 11:59 pm

**Late Due Handin:** Friday, October 21st, 10:00 pm

To run the demo: `cs015_runDemo Cartoon`

*Demos may not work well over ssh! Try FastX or the Sunlab.*

To run a snazzy demo\(^2\): `cs015_runSnazzyDemo Cartoon`

To install: `cs015_install Cartoon`

To handin: `cs015_handin Cartoon`

---

**Silly Premise**

- New Concepts Covered
- Assignment Specifications
- Programming Tips
- JavaFX
- Bells and Whistles (Extra Credit)
- Handing In Your Code

---

**Silly Premise**

It is a rainy rainy day and Sally and her brother cannot go outside to play. Just when they thought they would have to sit there forever, the Cat in the Hat showed up with a brand new endeavour. “Look at me! Look at me! Look at me now! It’s fun to have fun but you have to know how. I can hold up the cup and the milk and the cake! I can hold up the the cup and the milk and the cake! I can hold up these books and the fish on a rake!”

“Boring,” said Sally. “I want to watch TV,” said her brother.

---

\(^1\) Because of the individual nature of this project, you will have a 20 minute check-in with one of your design discussion TAs to go over strategies to best design your individual Cartoon. You should have received more information about these by email -- please contact the HTAs if you have not.

\(^2\) 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.
So the Cat in the Hat, disappointed and floored, asked the CS15 students to get going with their Umlet boards. He requested a Cartoon so sweet and so glorious that Sally and her brother would be 112.365% victorious.

**Collaboration Policy Reminder**

From the collaboration policy:

Collaboration on project design is not allowed, except for during the design discussion for this project under the supervision of your discussion leader TAs. Otherwise, **no collaboration is allowed on project-specific details.** You are not allowed to discuss the classes you will be using in your project, what methods you will be writing, inheritance hierarchies, the design discussion mini-assignment, or any other design components of the program. You may not discuss the implementation and debugging of code for projects with anyone except the course staff.

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

- 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 in each case, any examples used must be from the lectures or your own creativity – you may not discuss how even broad design concepts like containment/association pertain to a specific 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
- Changing speed (linear acceleration)

**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 once 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 use of a top-level PaneOrganizer, Cartoon class (see note further on and Help Slides), constants, and helper methods as needed.

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

- A composite shape, as discussed above
  - Some of the classes you could use are Circle, Rectangle, and Polygon. For example, you can have a composite shape of 3 Circles and a Rectangle.
  - Use at least two different types of Shape, e.g. a Rectangle and a Circle, when creating your composite shape.
- A Pane that contains your composite shape
- 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 KeyHandler (or equivalent) class that implements EventHandler<KeyEvent> and that 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 some button on the keyboard (completely up to you) causes some visible change in the cartoon
- A javafx.animation.Timeline that animates your Cartoon
- A composite shape that can move at 20 or more distinct speeds (Hint: think a speed variable that is updated)
  - 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)
  - This can be the same composite shape that meets the first requirement
- A javafx.scene.control.Label with text that changes based on something in your Cartoon (for example, based on the changing location of your 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

By creating a GUI we hope that you will gain a solid grasp of JavaFX. However, we don't want to squash your creativity by dictating the exact GUI layout. Nevertheless, if you are lacking inspiration, you may emulate the layout of the demo. In any case, be sure to explain why you arranged your GUI the way that you did in your comments.
Programming Tips

JavaFX
This is the first of the projects using JavaFX. JavaFX is designed to be used for a wide variety of applications and thus has many tools that you can use for this and later projects.

Because you'll be doing this in 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 TASafeHouse. 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 moving your shapes across the screen. Instead, PaneOrganizer can delegate most of the responsibility for animation and user input to another class- your Cartoon class! See the Help Slides 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.

20+ Distincts Speeds (Acceleration)
There are a few ways of doing this. Before thinking about how to change speeds, make sure you are comfortable with how to get an object to move at a constant speed (see the Alien example in lecture). Here are a few things to consider for changing the speed of an object:

1. If there is a variable that keeps track of a current speed, which class would contain this variable? How would we access it?
   a. In previous labs and projects, how have we kept track of a current color?
2. Where would we want to tell the composite shape to change speeds?
   a. How would we update/increment the current speed variable for the next iteration?
3. What does “speed” mean in regards to moving the object? (Hint: think “change in pixels”)
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 Javadocs 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 KeyCodes, you can check the JavaDocs. Once you know what keyboard key was pressed, you can make a decision (hint, hint... “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)³

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):

- 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!
- Make use of an interface in your design in a meaningful way
- Find a way to have mouse AND keyboard interactions!
- Add images! Read more here.
- 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 Javadocs for possible effects.

³ Note: Because many of the department computers do not have the necessary libraries to play audio, we do not recommend trying to add audio and cannot give extra credit for it. If you do add sound, as always, make sure your program still runs on the department computers without runtime exceptions before turning in.
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] to be done only after the original assignment is completed in full. You must include full documentation of your program’s additional functionality in the program header. In other words, if everything is working right, extensions will give your grade a boost, but if you did not do part of the assignment, the extensions will be worth nothing. This also means that extra credit will not be assessed for your project if you hand in a project late. This is not arbitrary; most extensions, while requiring extra work, do not teach new concepts.”

Handing In Your Code

In order to hand in your code, run the handin script at the top of this handout. 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 something in the Cartoon.
- 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).
- At least one of the following:
  - An element in your Cartoon visually changes based on a KeyEvent
  - A composite shape that moves at 20 or more different speeds.