CMPT 381 Assignment 5: Grouping, Copying, Undo

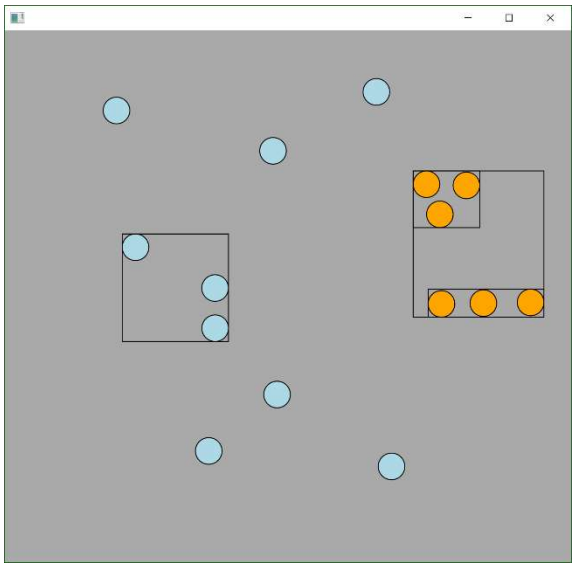
Due: Friday, April 6, 11:55pm

Overview

In this assignment, you will work with fundamental interactions in visual workspaces – grouping, cut/copy/paste, and undo/redo. You will use JavaFX to extend the Blob demo developed in class and will add several interactive capabilities to the system. You will also build on your MVC knowledge of models that contain grouped hierarchical objects.

Part 1: Grouping and Ungrouping

Using JavaFX, add to the Blob demo application developed in class (code available on the moodle site) so that the system can handle grouping of objects. The demo system already supports multiple selection using control-click and rubber-band rectangles, and you will adapt the system to enable grouping of a multiple selection. This will require you to convert the model to work with Groupable items (which can be either a Blob or a Group) instead of Blob items.



***Interaction requirements*:**

- (Basics of object creation and multiple selection are built in to the provided BlobDemo application)
- When several blobs have been selected, the user can press the 'g' key to group them
- The view displays a group by showing its bounding box in addition to the blobs themselves (for all groups, at all times)
- When the user selects any of the blobs in a group, the entire group is shown as selected (in orange)
- When the user drags any of the blobs in a group, the entire group moves
- Groups are hierarchical: they can contain other groups as well as blobs (e.g., if the user clicks on two groups and presses 'g', they become a new group containing the two original groups). See figure at right.
- Selecting a group and pressing the 'u' key ungroups the blobs to the next level (e.g., in the example above, selecting the new group and pressing 'u' would then show only the two original selected groups)

***Software requirements*:**

- Create a Groupable interface that will define the API for both groups and individual blobs
- Adapt the Blob class to implement the Groupable interface
- Create a Group class that also implements Groupable, and that stores a collection of Groupable items as its children
- Adapt the model class to store a collection of Groupable items instead of a collection of Blob items
- Adapt the interaction model class to use Groupable items in the selection, instead of Blob items
- Adapt the view and controller classes to use the new model structures
- An example set of method definitions for the Groupable interface:

```
boolean hasChildren();
ArrayList<Groupable> getChildren();
boolean contains(double x, double y);
boolean isContained(double x1, double y1, double x2, double y2);
void draw(GraphicsContext gc, boolean selected);
void move(double dx, double dy);
```

```
double getLeft();   // bounding box of the group
double getRight();
double getTop();
double getBottom();
void setZ(int newZ);  // Z-order for the group
int getZ();
```

Resources for part 1:

- Starting application: BlobDemo.zip in the Examples directory of the course moodle

Part 2: Cut/Copy/Paste

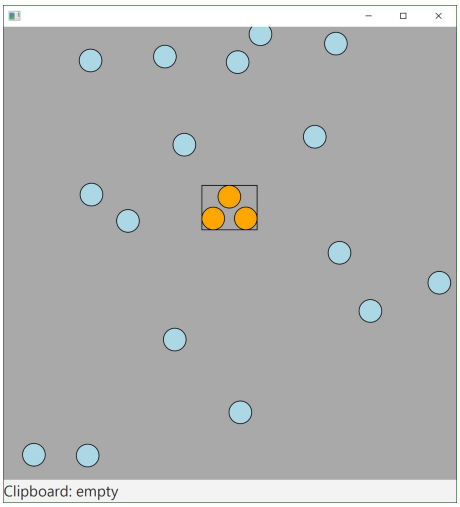
Add functionality to your application that implements a clipboard, with three operations: Cut (control-x), Copy (control-c), and Paste (control-v). Your extended application should meet the following requirements:

***Interaction requirements*:**

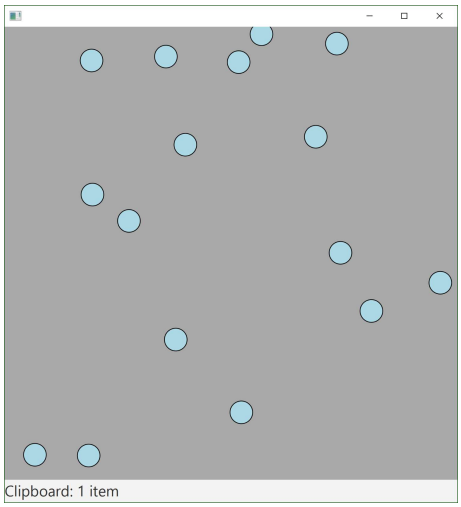
- When the user presses Control-c, any selected blobs/groups are copied to a custom clipboard (see below for details)
- When the user presses Control-x, the selection is removed from the current model and is placed on the custom clipboard
- When the user presses Control-v, any blobs/groups on the custom clipboard are pasted into the model (and shown in the view)
- Pressing Control-v again pastes additional copies of the objects (at the location where they were copied/cut)
- When cut/copied and then pasted, any existing group structure in the selection is preserved.

***Software requirements*:**

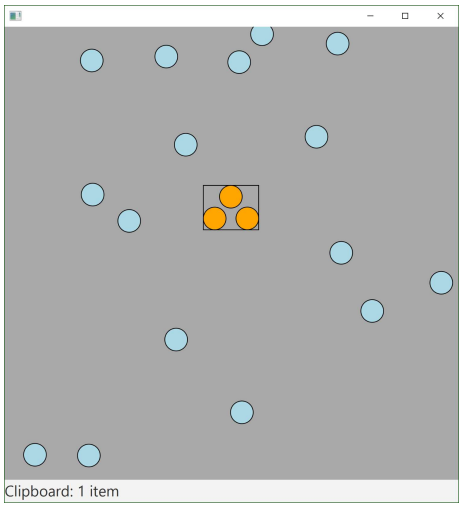
- Add code to the controller class to accomplish the new interaction requirements described above
- Create a new class BlobClipboard to store items that have been cut or copied
- The clipboard object should be created and manipulated in the interaction model
- When copying to the clipboard and when pasting, you will need to make a *deep copy* of the objects (as described in lectures), because you may be pasting them multiple times, and each paste needs to add different objects to the model
- Do **not** use the Java system Clipboard classes. Implement your own custom class as described above.



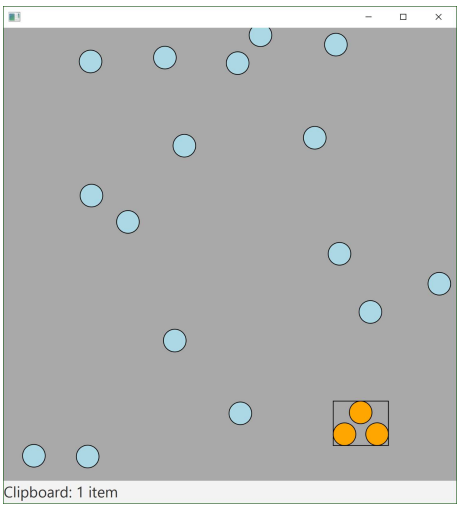
After selecting and grouping



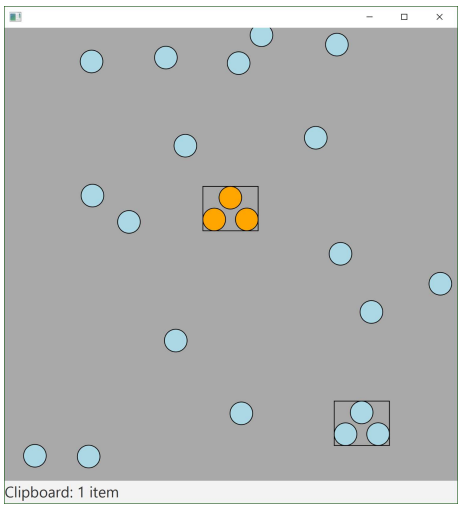
After cutting with control-x



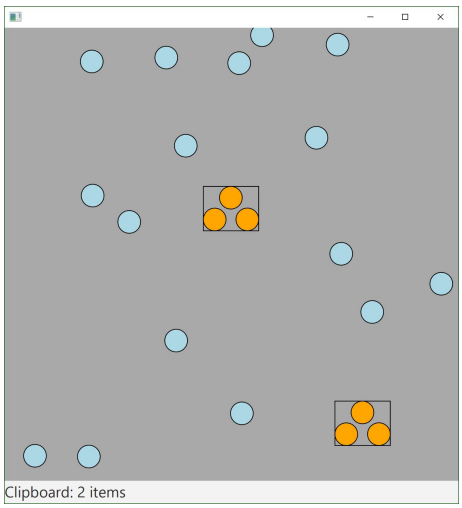
After pasting with control-v



After dragging pasted group



After pasting again with control-v



After selecting and copying with control-c

Resources for part 2:

- Details on cut/copy/paste operations: Olsen text, chapter 15
- Tutorial on Java copying: https://dzone.com/articles/java-copy-shallow-vs-deep-in-which-you-will-swim

Part 3: Undo/Redo

In the third part of the assignment, you will extend your system to add the ability to Undo and Redo certain actions in the visual workspace. You will use the Backup Undo approach described in lectures and the text, using the Command pattern.

Additional interface requirements:

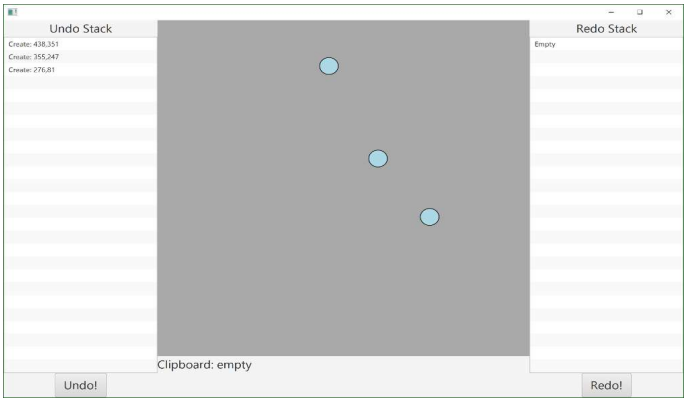
- Add panels to the left and right of the main workspace
- Each panel contains a Label, a ListView, and a Button
- The left panel shows the current state of the undo stack, and the right panel the state of the redo stack
- Pressing the "Undo!" button executes one undo; pressing the "Redo!" button executes one redo

Additional interaction requirements:

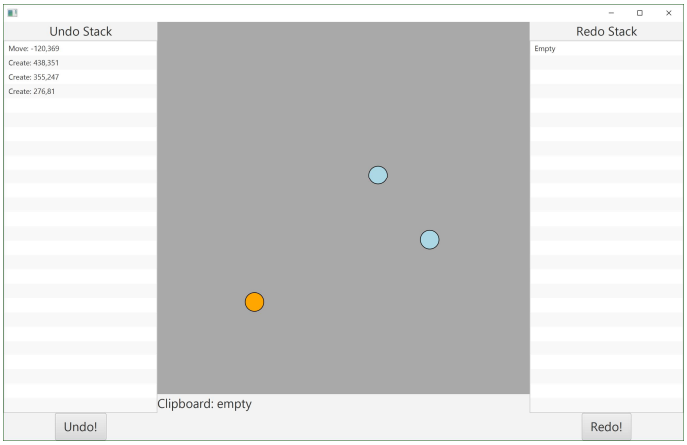
- All blob creation, and all blob/group move actions result in the creation of a new BlobCommand object that encapsulates how to undo and redo that action
- Only create and move actions are undoable/redoable. Selection and grouping actions, and cut/copy/paste actions are *not* undoable/redoable (and it is not required that the selection state match the state of the workspace when a particular action was first carried out).
- Whenever a BlobCommand object is created it is pushed onto the undo stack (and the ListView is updated)
- Whenever the "Undo!" button is pressed, the undo stack is popped and the top BlobCommand's undo() method is executed. In addition, the BlobCommand is then pushed onto the redo stack.
- When the "Redo!" button is pressed, the redo stack is popped and the top BoxCommand's doIt() method is executed. In addition, the BoxCommand is then pushed onto the undo stack.
- Note: the markers will only test basic versions of undo and redo, because interleaving creates and moves with grouping actions and cut/copy/paste actions can lead to odd behaviour.

***Additional software requirements*:**

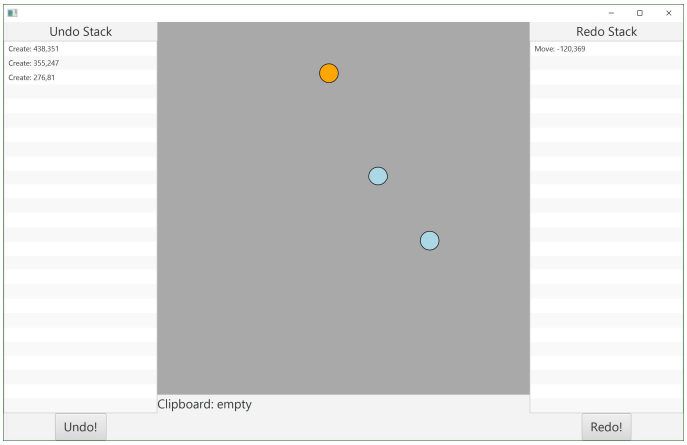
- Implement the BlobCommand interface (following the example in the text)
- Implement classes MoveCommand and CreateCommand that implement this interface
- Each of these command classes should include a toString() method so that you can put a string representation of the command into the ListView's list model
- In your InteractionModel, add stack data structures as needed for the undo and redo stacks (you may use the old Stack class, or the more modern Deque class)
- You may treat the Application class as the home for the ListViews (you should add methods to the Application class to update the ListViews when the stacks change)



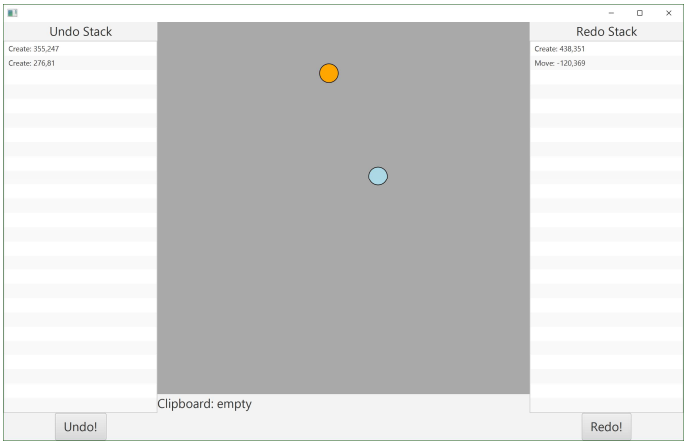
After creating three blobs



After moving one blob



After undoing the move operation



After undoing the final create operation

This assignment is to be completed individually; each student will hand in an assignment.

What and Where to hand in

- JavaFX: a zip file of your NetBeans project folder for the system. If a part of the system is unfinished, you may hand in multiple versions (e.g., part 2 working, part 3 not) to receive partial marks.
- A readme.txt file that indicates exactly what the marker needs to do to run your code. (Systems for 381 should never require the marker to install external libraries).
- Hand in your all files (project zip and one readme.txt) to the link on the course Moodle.

Evaluation

Marks will be given for producing a system that meets the requirements above, and compiles and runs without errors. Note that no late assignments will be allowed, and no extensions will be given, without medical reasons.