CS542 – Design Patterns and Object-Oriented Analysis
Dr. Fan

Group 5
Samuel Mueller
Akhil Sriram
Rishi Ramrakhyani

PhysX: Project Report

Introduction

Description:

PhysX is java application that runs a text-based physics simulation where users are able to create up to ten 3-dimensional shapes (Sphere, Cube, Cylinder). Users are able to determine the size (volume, height, radius, etc.), mass, and location (x,y,z) of these shapes. Once users have created their shapes, they are able to begin a simulation where they can specify the gravity and friction forces that will effect object movement. Once the environment for the simulation has been determined, the user is able to push, pull, lift, and drop selected objects. After each action the user will see the distance traveled by the object they selected and the objects new position. If any collisions occur between objects, the user will be notified and given the same information for the other effected objects.

Project Background:

Members of this group wanted to become more familiar with design patterns and their implementation – while still working on a project that was not very common (e.g. vending software, or a video game). In an attempt to make a generic and reusable project, the decision landed on a physics engine that could be used as the basis of other projects in the future.

Key Features:

Key Features are: Navigating Between Menus, Setting Shape Limit, Creating Shapes, Specifying Shape Properties, Listing All Shapes, Erasing Shapes, Specifying Environment Properties, Pushing Shapes, Pulling Shapes, Lifting Shapes, Dropping Shapes, Handling Shape Collisions, Printing Results, Undoing Actions, Redoing Actions, Saving Simulation Files, and Loading Simulation Files.

Potential Users:

As stated in the Project Background, PhysX was designed to be generic in order to increase reusability. This project can be used as the basis for any software that requires the modeling of forces and object interaction, e.g. Video Game Engines, Teaching Software, etc. As is, the software can also be used to demonstrate basic fundamental physics lessons (the functions of friction, acceleration, velocity, gravity, etc.)

System Design

Functional Requirements:

● The User shall be able to create an environment with specified parameters (e.g. friction, gravity)

● The User shall be able to create objects with specified parameters (e.g. shape, height, radius, mass, position)

● The User shall be able to apply a specified quantity of force to a selected object in a specified direction

● The System shall calculate and display the results of this initial movement (e.g. distance traveled, stopping point)

● The System shall calculate and display the results of any collisions predicated on the initial movement (e.g. what collisions take place, how much force, distance traveled)

● The User shall be able to undo or redo any application of force to a selected object

● The System shall record the state of all objects before any user-action takes place

● The System shall restore the state of the selected object and objects that moved due to collision upon an undo/redo action from the user.

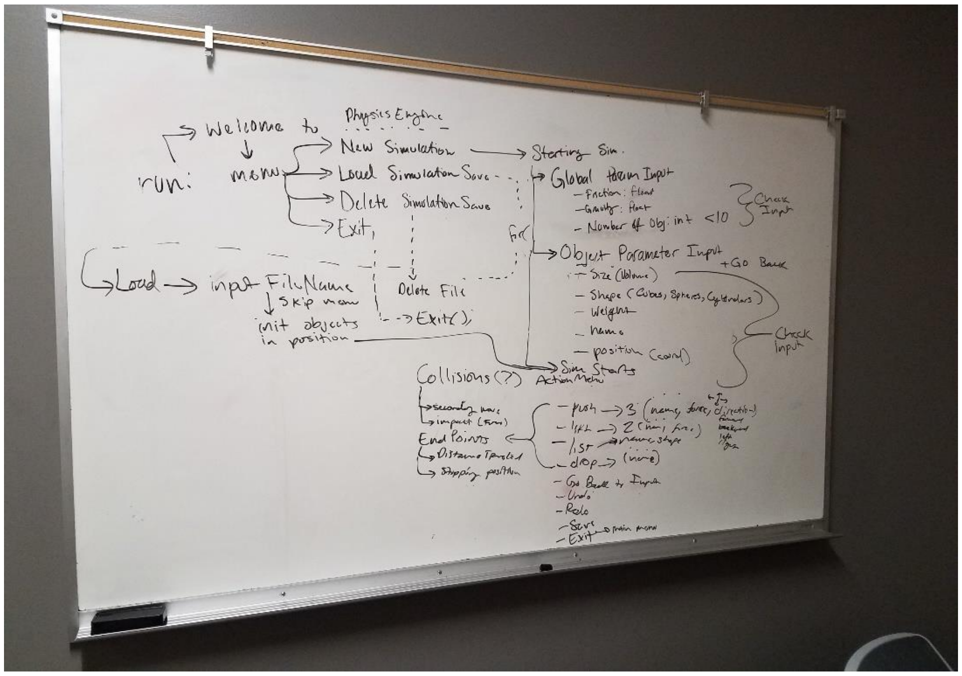
● User shall be able to save the current state of all objects to an external ‘save’ file

● User shall be able to determine the name of their external ‘save’ file when saving

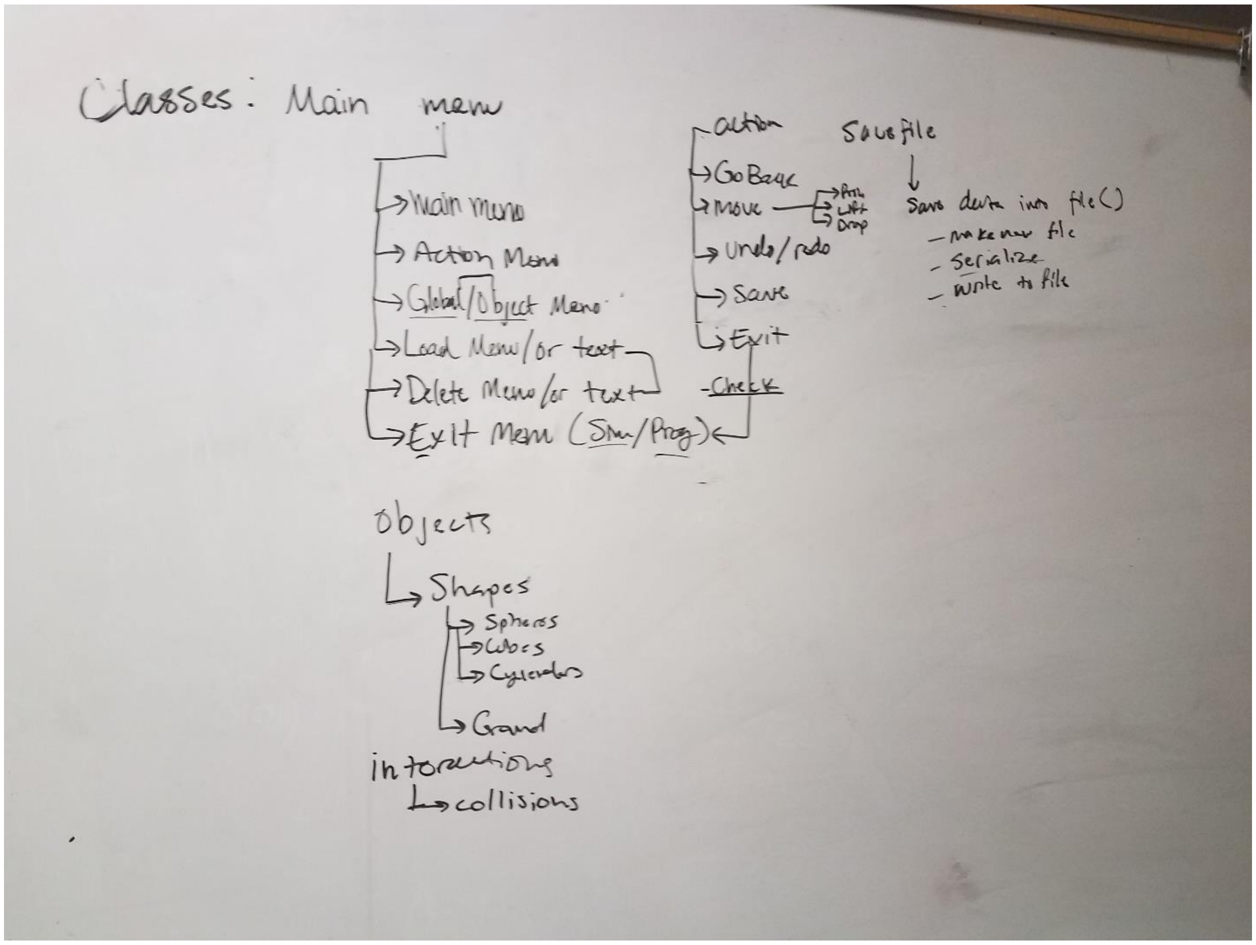
● User shall be able to load the state of all objects from a ‘save’ file when running the software

Design Process:

With the above goals in mind, the team was able to draft an idea of what the user-experience should look like:



With an idea of how the user-experience should be designed, we were able to identify key components of our software:

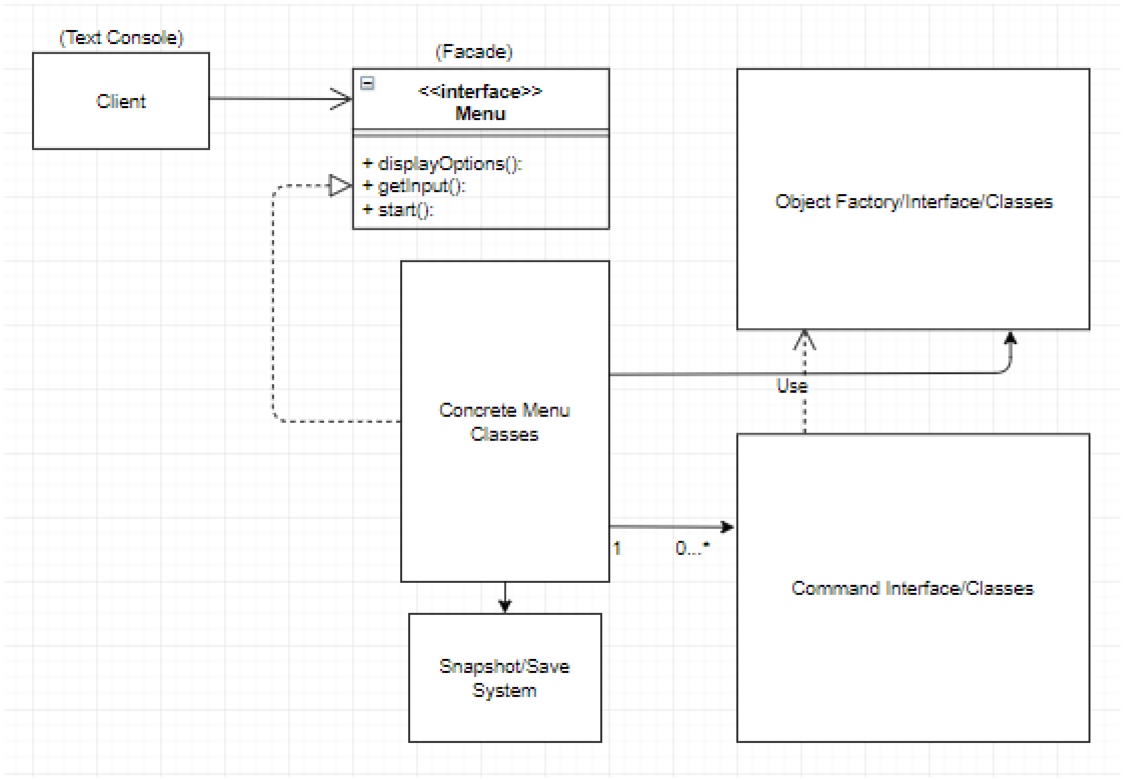


From this point, we built a software skeleton – creating empty classes and discussing how they might interact with each other. After these discussions we were able to create our interfaces. The interfaces made it clear what information would be passed between different components of the system – meaning we were now able to divide the work between members of the group.

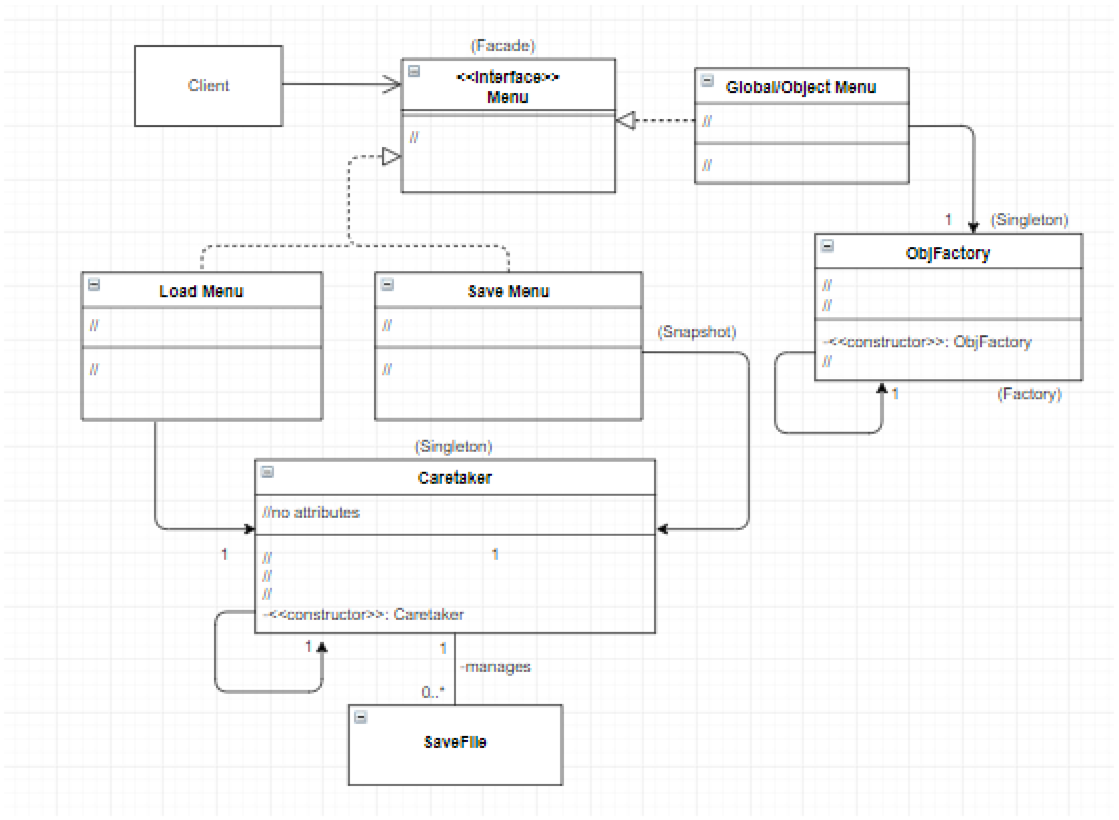
Design Patterns:

The final iteration of the software included the following five design patterns.

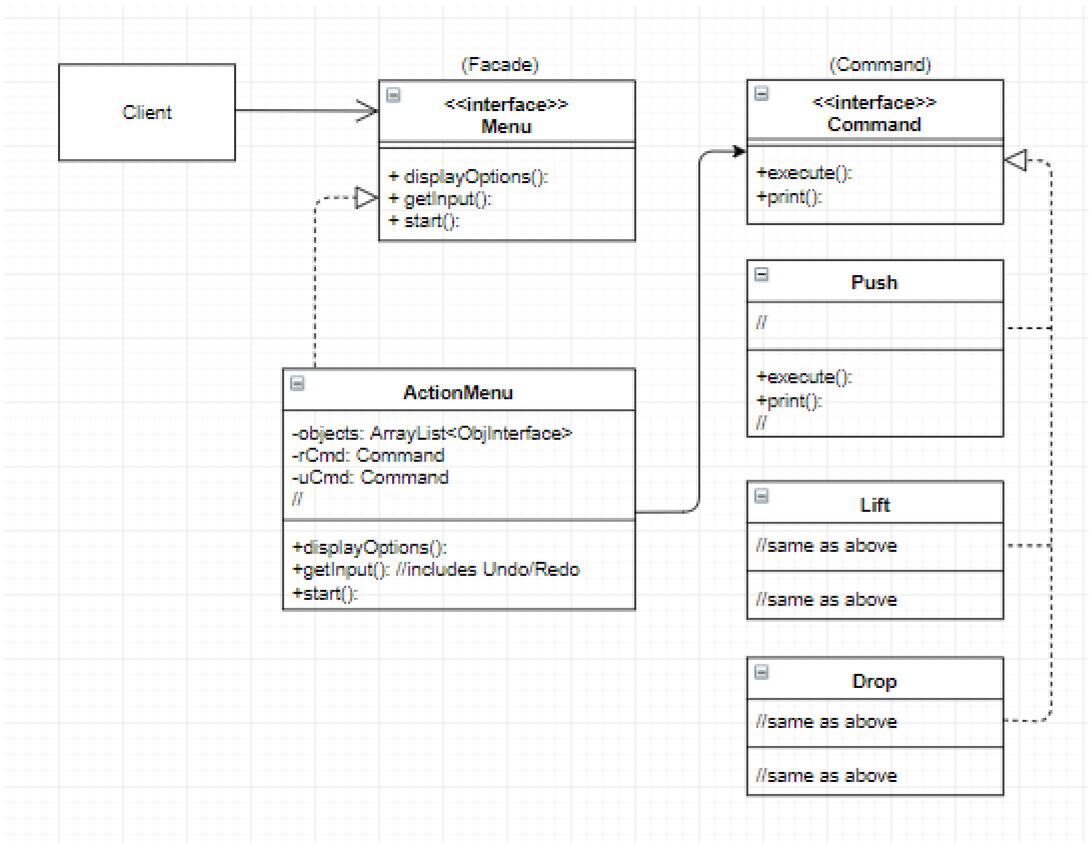
1) Façade – This design pattern included a class that would abstract the system from the user. The client would call the menu functions through the façade interface and the system would return the results the same way.



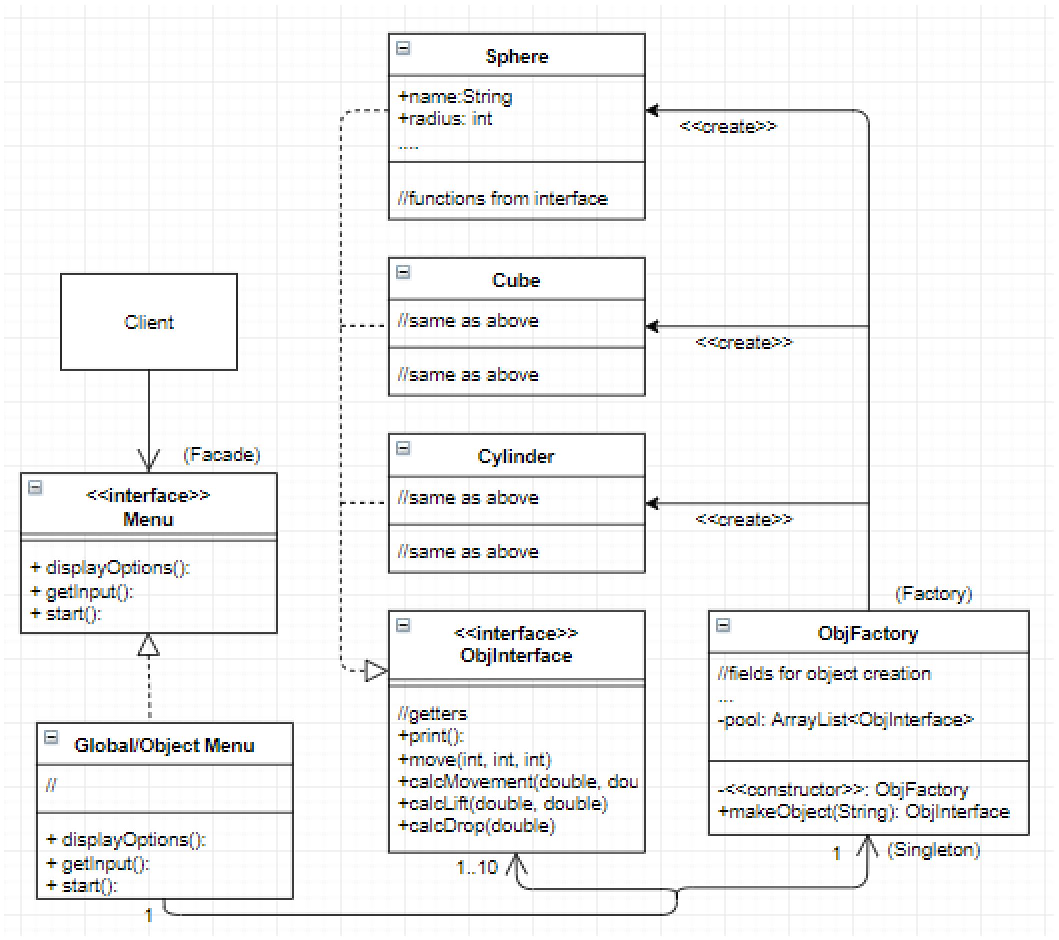
2) Singleton – This design pattern included two classes. One was the ObjectFactory, a class which was computationally expensive to create and which our software only required a single instance of. The second singleton class in our design was the CareTaker class in the save system. This class, while less expensive than the Factory, was able to read and write into save files. To avoid any possible issues with corruption from concurrent reads/writes – the group elected to make the CareTaker a singleton as well.



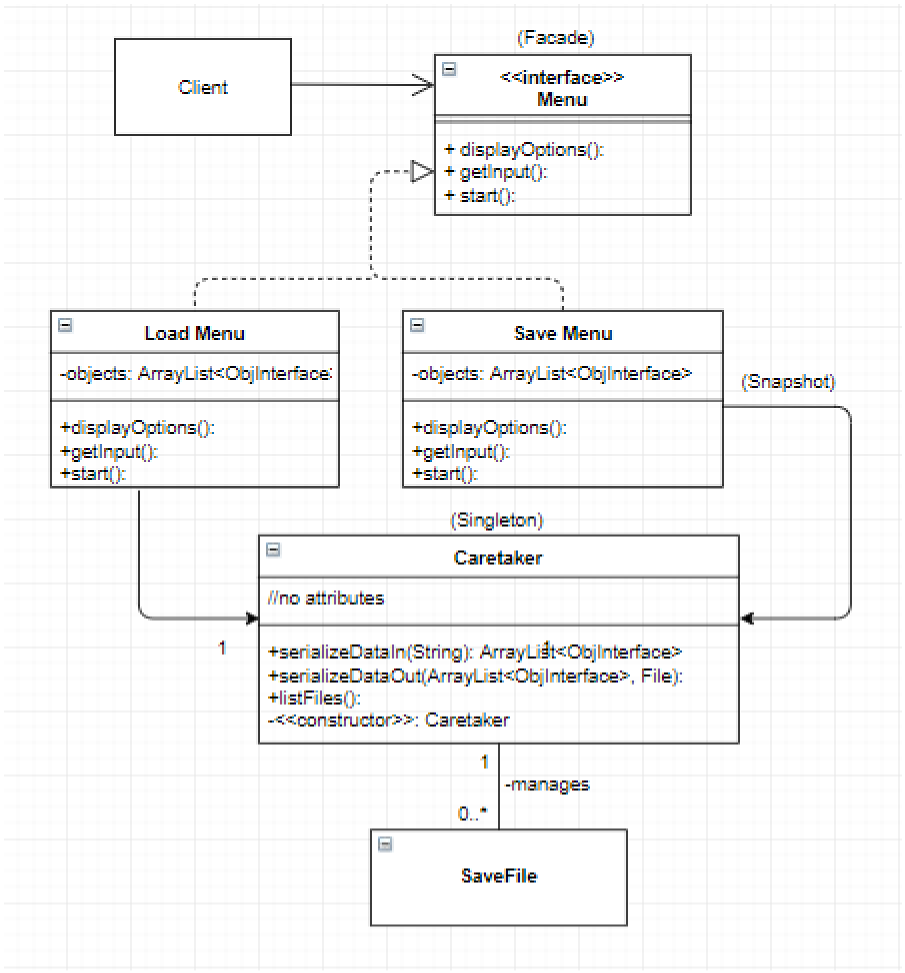
3) Command – This design pattern included five classes (menu, commands, and an interface) and enabled the system to keep track of user actions as first class objects. Specifically, when a user (through the façade) requested that an object be pushed, lifted, or dropped – the system was able to execute that specific command and track it for undo/redo purposes (along with the state of the objects it affected)



4) Factory – This design pattern included five classes (the concrete products, the product interface, and the factory). By using the Factory pattern, we were able to abstract the instantiation of products away from the user and centralize the implementation logic (allowing us to more easily limit the number of objects created). In addition, this pattern allowed users to choose what shapes they wanted to create during runtime (dynamically).



5) Snapshot – This design pattern included 3 classes (Load/Save menu and the CareTaker) and facilitated two functional requirements for our software. The first- undo/redo functionality. When the program is running, the system would call the CareTaker to take a snapshot of the current simulation and save it to a temporary file. If the user elected to undo a command, the snapshot was simply restored (thus undoing the command). If the user elected to redo the command, the most recent state (after the command had taken place) was reapplied. In addition to undo/redo functionality, the Snapshot (momento) design pattern allowed us to create save files for each simulation that could be loaded into the software by the user (allowing for the sharing of simulation files or saving the user setup time for repeated experiments).

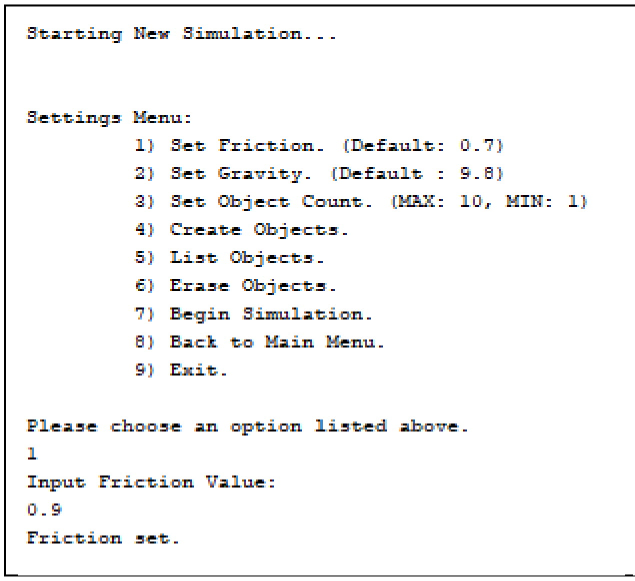


The combination of these design patterns (full system design) can be seen in the presentation slides from class.

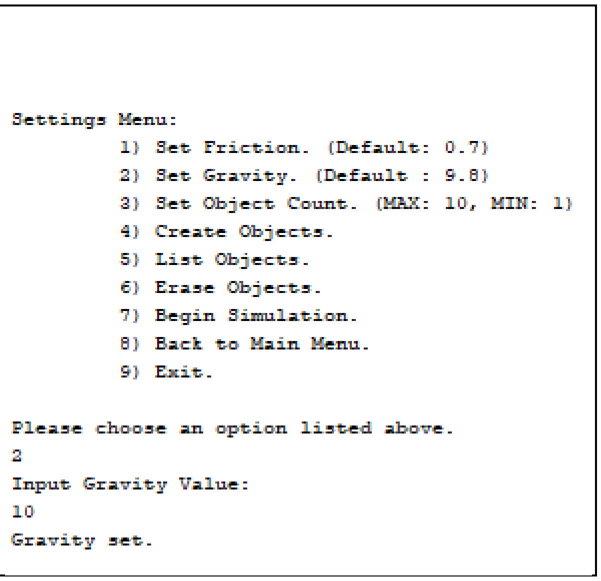
System Implementation: (source code available with submission)

Every feature listed in the functional requirements has been implemented. In short the user can:

Set environment parameters (GlobalMenu.java):

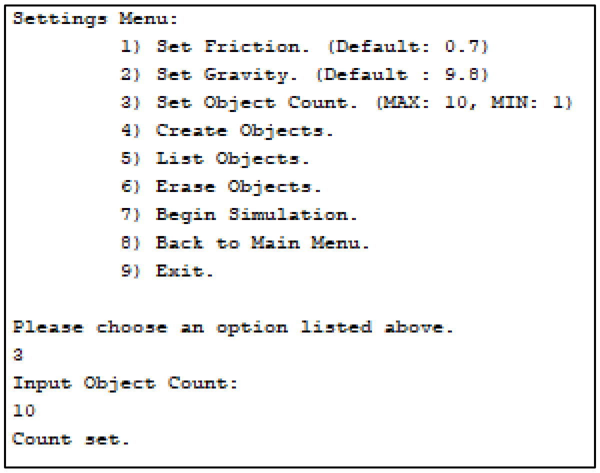


```
Starting New Simulation...


Settings Menu:
         1) Set Friction. (Default: 0.7)
         2) Set Gravity. (Default : 9.8)
         3) Set Object Count. (MAX: 10, MIN: 1)
         4) Create Objects.
         5) List Objects.
         6) Erase Objects.
         7) Begin Simulation.
         8) Back to Main Menu.
         9) Exit.

Please choose an option listed above.
1
Input Friction Value:
0.9
Friction set.
```



```
Settings Menu:
         1) Set Friction. (Default: 0.7)
         2) Set Gravity. (Default : 9.8)
         3) Set Object Count. (MAX: 10, MIN: 1)
         4) Create Objects.
         5) List Objects.
         6) Erase Objects.
         7) Begin Simulation.
         8) Back to Main Menu.
         9) Exit.

Please choose an option listed above.
2
Input Gravity Value:
10
Gravity set.
```

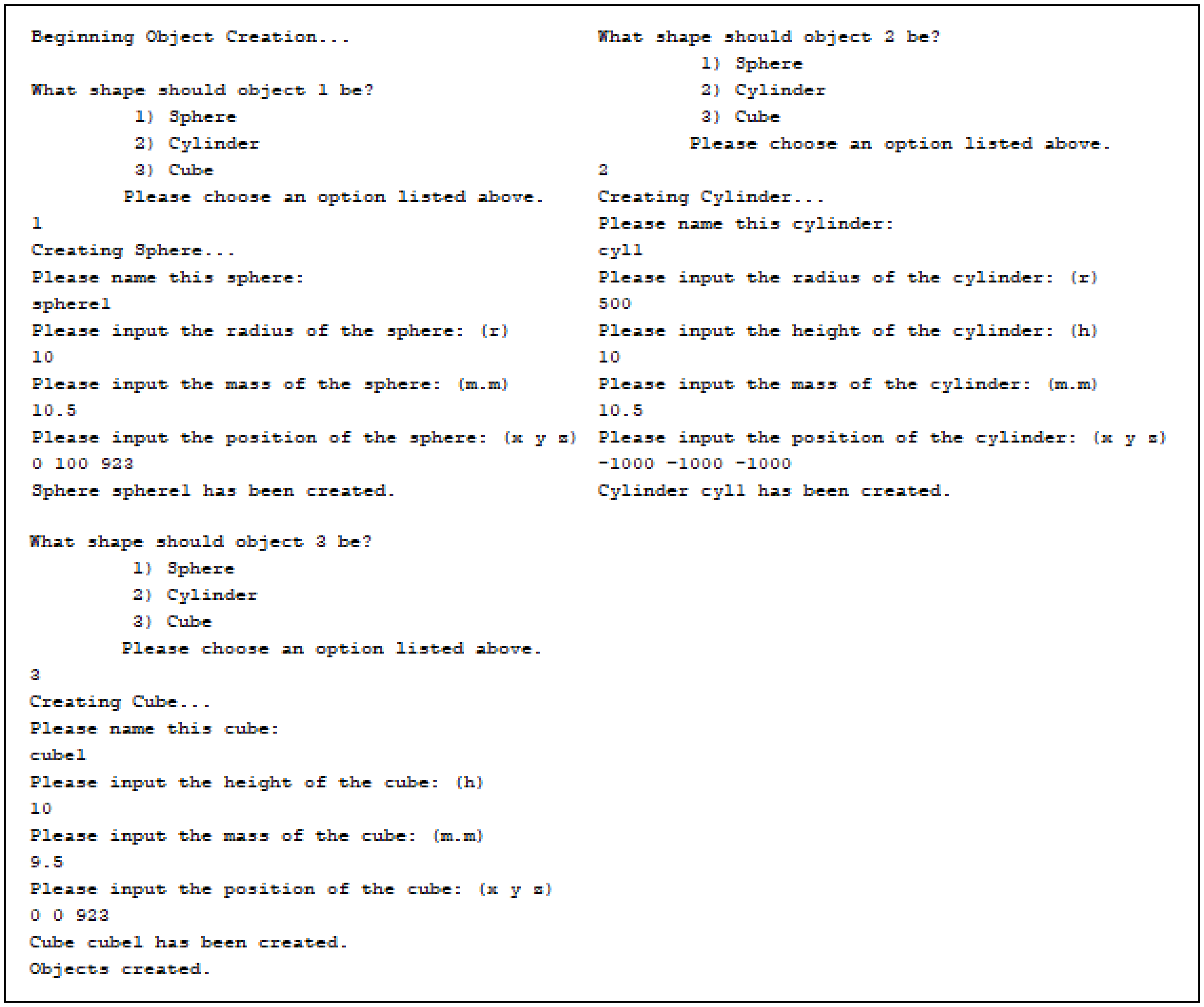
Specify a number of objects to create (GlobalMenu.java):



```
Settings Menu:
         1) Set Friction. (Default: 0.7)
         2) Set Gravity. (Default : 9.8)
         3) Set Object Count. (MAX: 10, MIN: 1)
         4) Create Objects.
         5) List Objects.
         6) Erase Objects.
         7) Begin Simulation.
         8) Back to Main Menu.
         9) Exit.

Please choose an option listed above.
3
Input Object Count:
10
Count set.
```

(continued)

Create those objects with specific parameters(GlobalMenu.java):

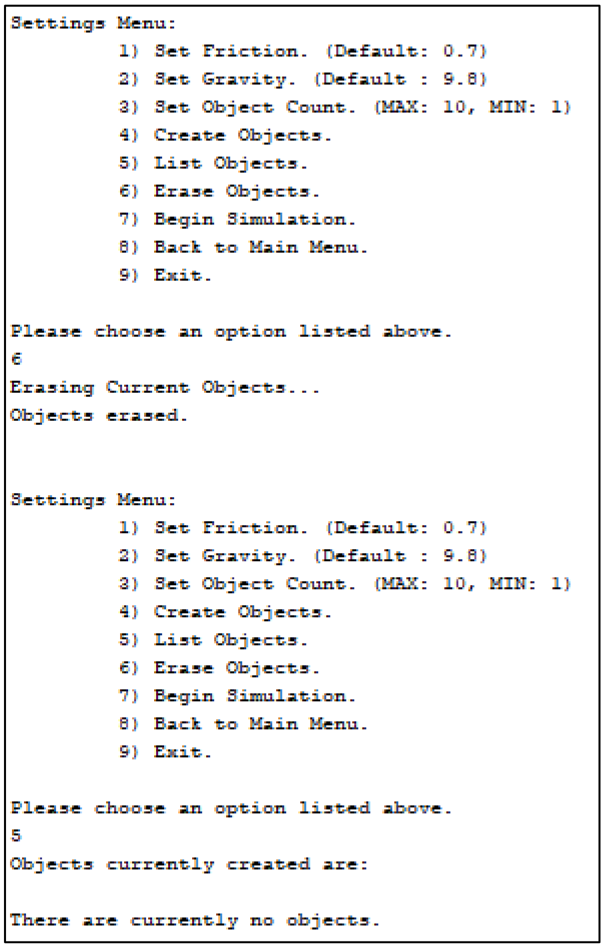


```
Beginning Object Creation...

What shape should object 1 be?
         1) Sphere
         2) Cylinder
         3) Cube
        Please choose an option listed above.
1
Creating Sphere...
Please name this sphere:
sphere1
Please input the radius of the sphere: (r)
10
Please input the mass of the sphere: (m.m)
10.5
Please input the position of the sphere: (x y z)
0 100 923
Sphere sphere1 has been created.
```

```
What shape should object 2 be?
         1) Sphere
         2) Cylinder
         3) Cube
        Please choose an option listed above.
2
Creating Cylinder...
Please name this cylinder:
cyl1
Please input the radius of the cylinder: (r)
500
Please input the height of the cylinder: (h)
10
Please input the mass of the cylinder: (m.m)
10.5
Please input the position of the cylinder: (x y z)
-1000 -1000 -1000
Cylinder cyl1 has been created.
```

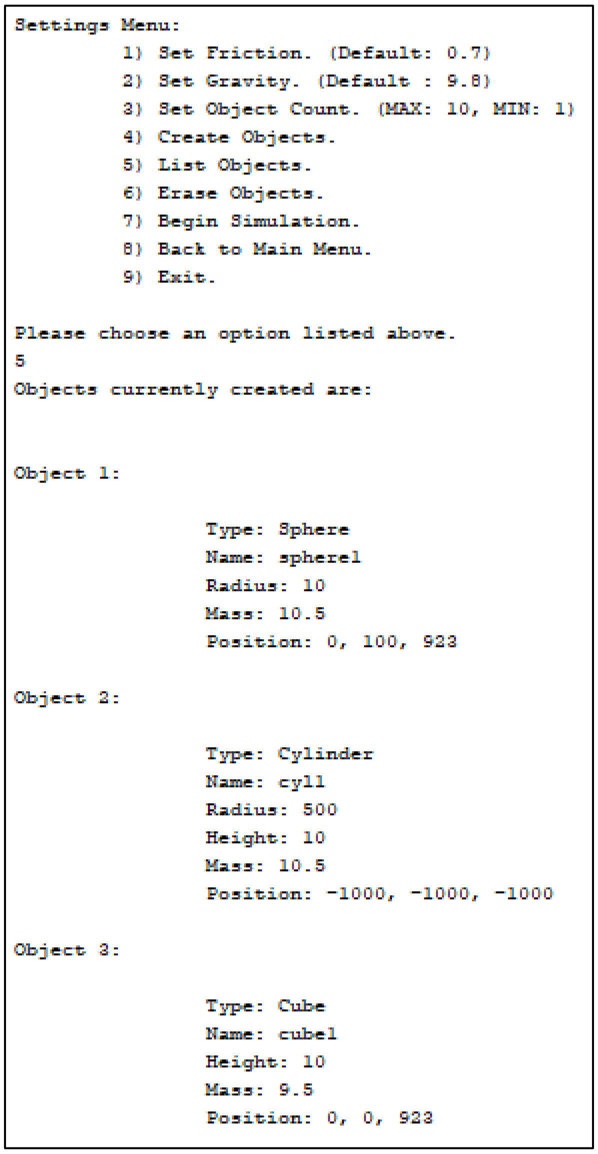
```
What shape should object 3 be?
         1) Sphere
         2) Cylinder
         3) Cube
        Please choose an option listed above.
3
Creating Cube...
Please name this cube:
cube1
Please input the height of the cube: (h)
10
Please input the mass of the cube: (m.m)
9.5
Please input the position of the cube: (x y z)
0 0 923
Cube cube1 has been created.
Objects created.
```

Erase objects(GlobalMenu.java):



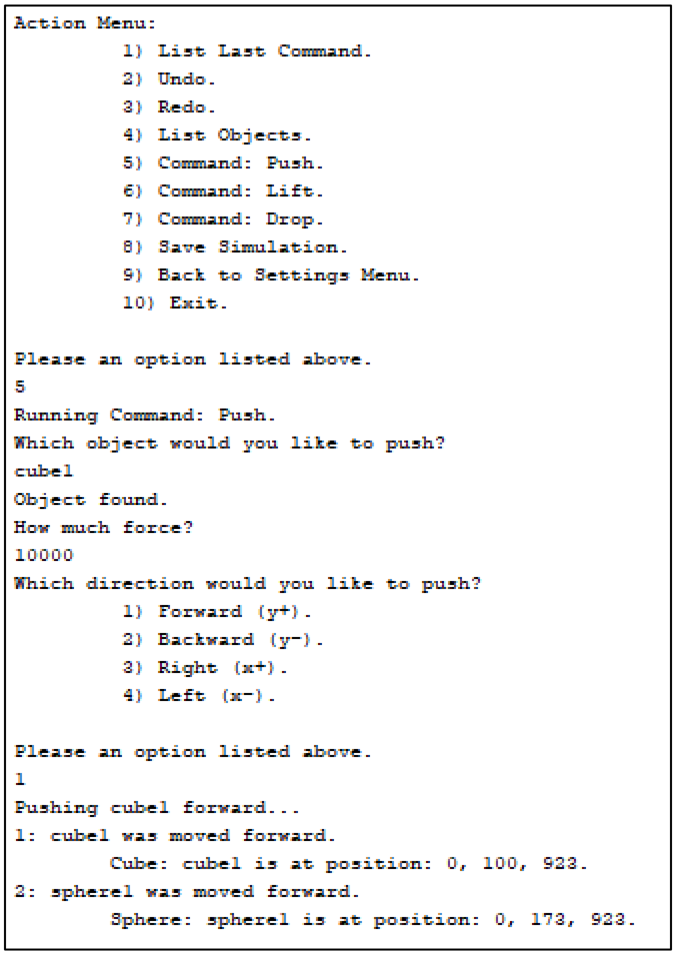
```
Settings Menu:
         1) Set Friction. (Default: 0.7)
         2) Set Gravity. (Default : 9.8)
         3) Set Object Count. (MAX: 10, MIN: 1)
         4) Create Objects.
         5) List Objects.
         6) Erase Objects.
         7) Begin Simulation.
         8) Back to Main Menu.
         9) Exit.

Please choose an option listed above.
6
Erasing Current Objects...
Objects erased.


Settings Menu:
         1) Set Friction. (Default: 0.7)
         2) Set Gravity. (Default : 9.8)
         3) Set Object Count. (MAX: 10, MIN: 1)
         4) Create Objects.
         5) List Objects.
         6) Erase Objects.
         7) Begin Simulation.
         8) Back to Main Menu.
         9) Exit.

Please choose an option listed above.
5
Objects currently created are:

There are currently no objects.
```

List created objects(GlobalMenu.java/ActionMenu.java):

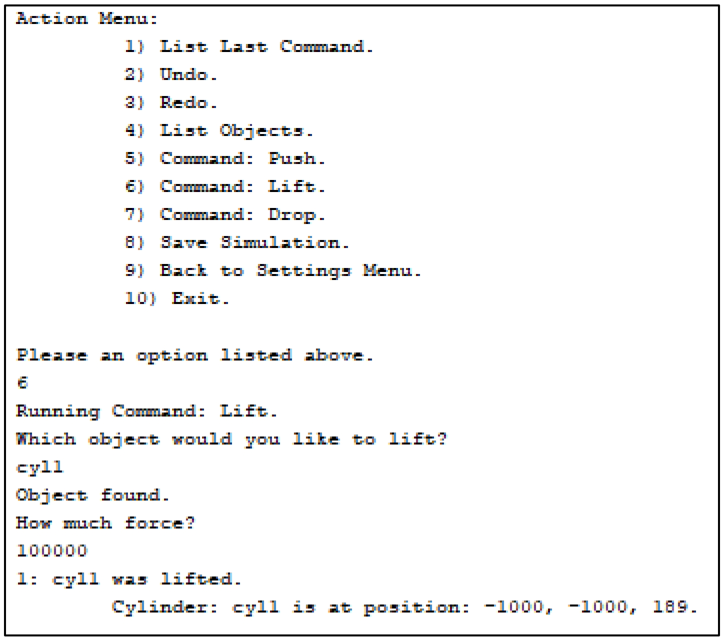


```
Settings Menu:
	1) Set Friction. (Default: 0.7)
	2) Set Gravity. (Default : 9.8)
	3) Set Object Count. (MAX: 10, MIN: 1)
	4) Create Objects.
	5) List Objects.
	6) Erase Objects.
	7) Begin Simulation.
	8) Back to Main Menu.
	9) Exit.

Please choose an option listed above.
5
Objects currently created are:


Object 1:

		Type: Sphere
		Name: sphere1
		Radius: 10
		Mass: 10.5
		Position: 0, 100, 923

Object 2:

		Type: Cylinder
		Name: cyl1
		Radius: 500
		Height: 10
		Mass: 10.5
		Position: -1000, -1000, -1000

Object 3:

		Type: Cube
		Name: cube1
		Height: 10
		Mass: 9.5
		Position: 0, 0, 923
```

Push objects(ActionMenu.java):

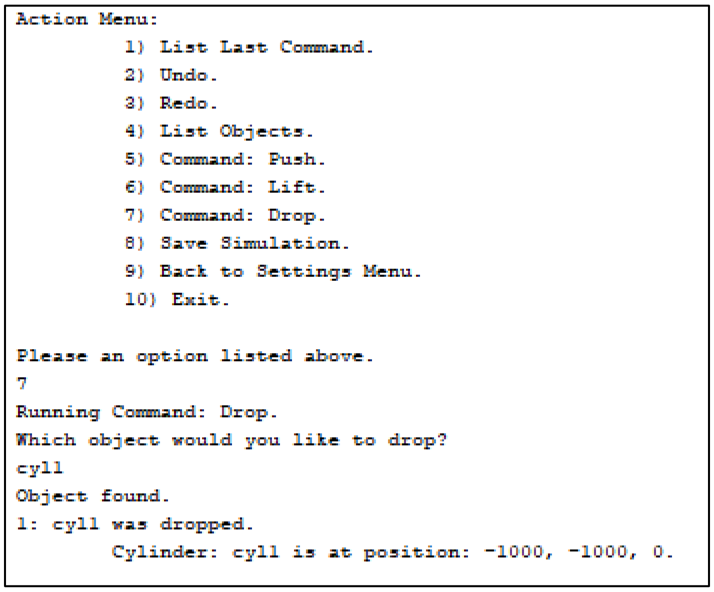

```
Action Menu:
         1) List Last Command.
         2) Undo.
         3) Redo.
         4) List Objects.
         5) Command: Push.
         6) Command: Lift.
         7) Command: Drop.
         8) Save Simulation.
         9) Back to Settings Menu.
         10) Exit.

Please an option listed above.
5
Running Command: Push.
Which object would you like to push?
cube1
Object found.
How much force?
10000
Which direction would you like to push?
         1) Forward (y+).
         2) Backward (y-).
         3) Right (x+).
         4) Left (x-).

Please an option listed above.
1
Pushing cube1 forward...
1: cube1 was moved forward.
        Cube: cube1 is at position: 0, 100, 923.
2: sphere1 was moved forward.
        Sphere: sphere1 is at position: 0, 173, 923.
```

Lift objects(ActionMenu.java):



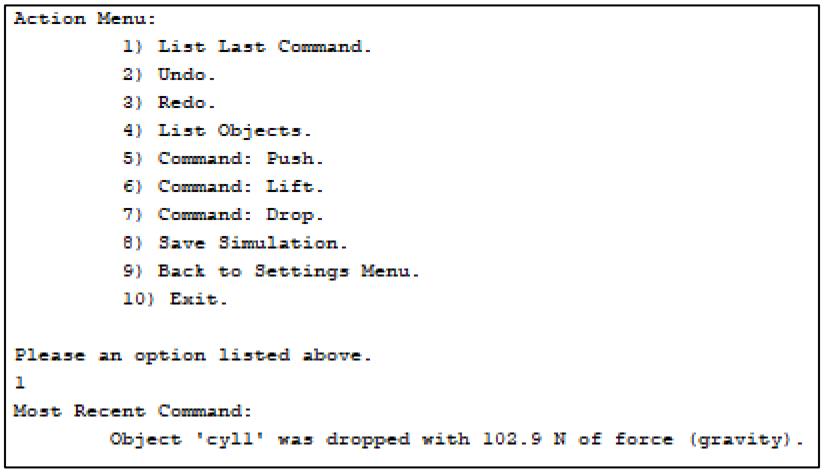
```
Action Menu:
         1) List Last Command.
         2) Undo.
         3) Redo.
         4) List Objects.
         5) Command: Push.
         6) Command: Lift.
         7) Command: Drop.
         8) Save Simulation.
         9) Back to Settings Menu.
         10) Exit.

Please an option listed above.
6
Running Command: Lift.
Which object would you like to lift?
cyll
Object found.
How much force?
100000
1: cyll was lifted.
        Cylinder: cyll is at position: -1000, -1000, 189.
```

Drop objects(ActionMenu.java):



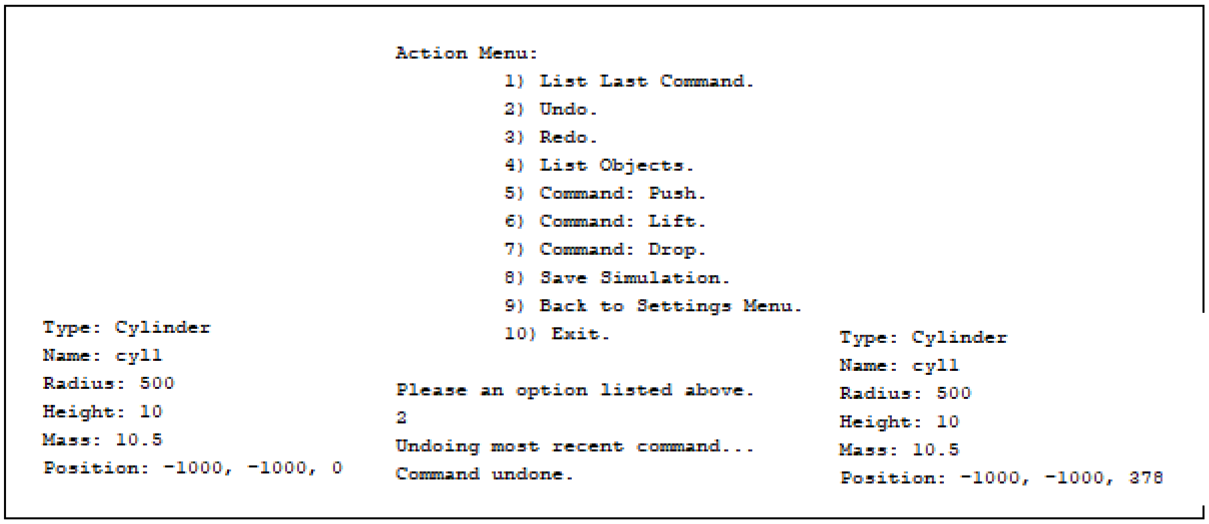
```
Action Menu:
         1) List Last Command.
         2) Undo.
         3) Redo.
         4) List Objects.
         5) Command: Push.
         6) Command: Lift.
         7) Command: Drop.
         8) Save Simulation.
         9) Back to Settings Menu.
         10) Exit.

Please an option listed above.
7
Running Command: Drop.
Which object would you like to drop?
cyll
Object found.
1: cyll was dropped.
        Cylinder: cyll is at position: -1000, -1000, 0.
```

View MRU command(ActionMenu.java):



```
Action Menu:
	1) List Last Command.
	2) Undo.
	3) Redo.
	4) List Objects.
	5) Command: Push.
	6) Command: Lift.
	7) Command: Drop.
	8) Save Simulation.
	9) Back to Settings Menu.
	10) Exit.

Please an option listed above.
1
Most Recent Command:
	Object 'cyll' was dropped with 102.9 N of force (gravity).
```

Undo commands(ActionMenu.java):

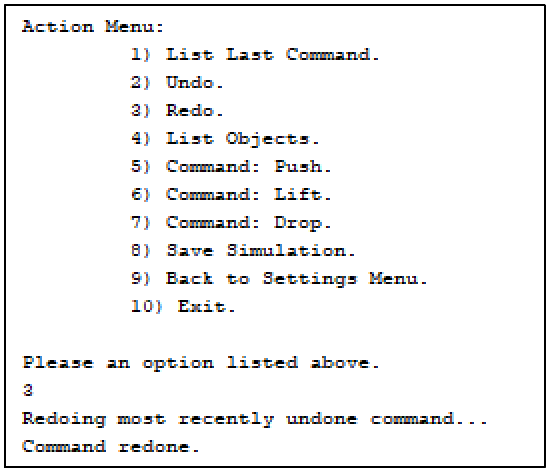


```
Type: Cylinder
Name: cyll
Radius: 500
Height: 10
Mass: 10.5
Position: -1000, -1000, 0
```

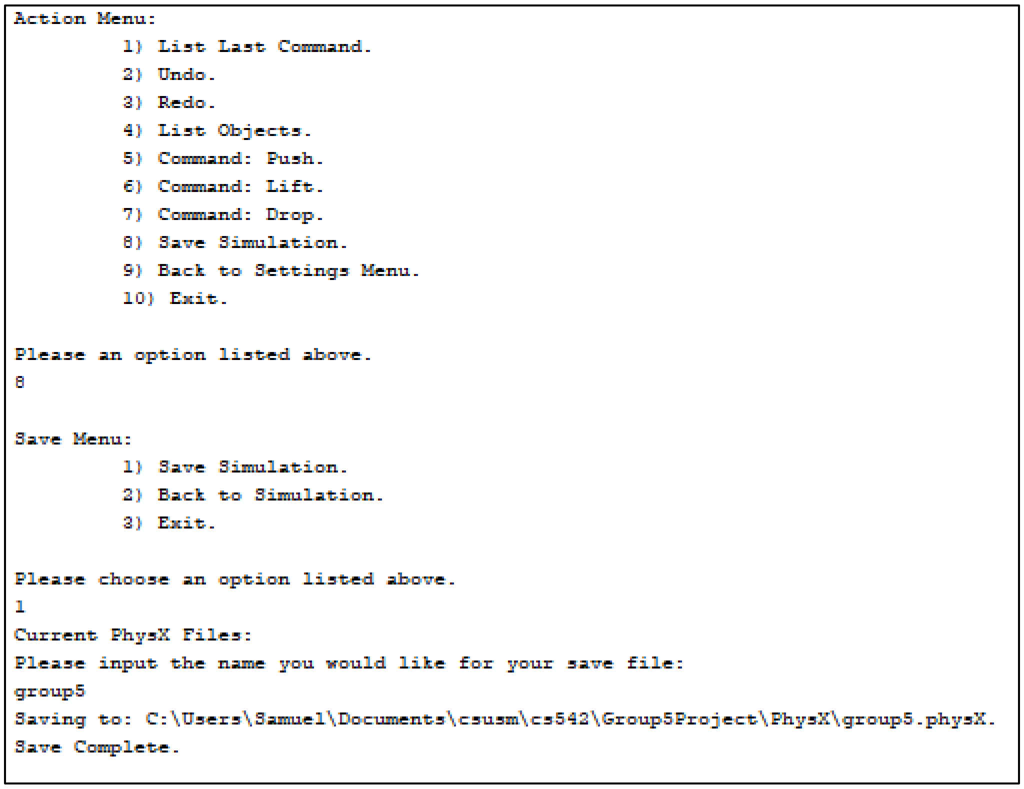
```
Action Menu:
	1) List Last Command.
	2) Undo.
	3) Redo.
	4) List Objects.
	5) Command: Push.
	6) Command: Lift.
	7) Command: Drop.
	8) Save Simulation.
	9) Back to Settings Menu.
	10) Exit.

Please an option listed above.
2
Undoing most recent command...
Command undone.
```

```
Type: Cylinder
Name: cyll
Radius: 500
Height: 10
Mass: 10.5
Position: -1000, -1000, 378
```

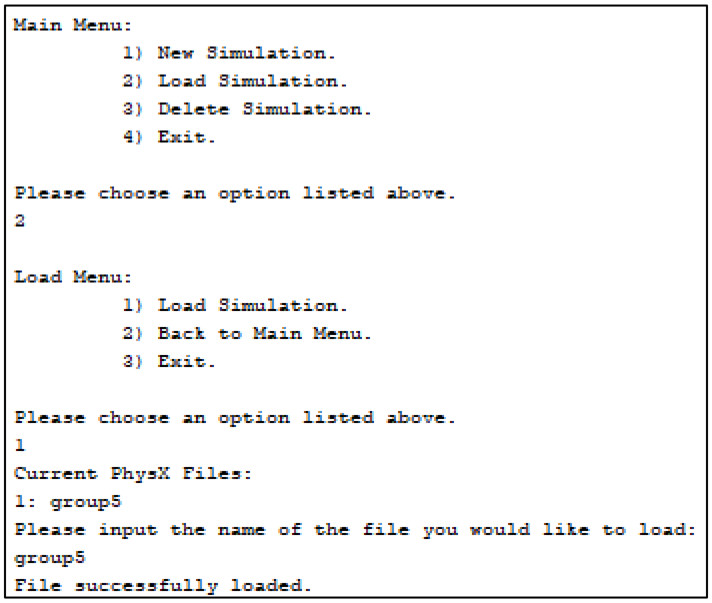
Redo undone commands(ActionMenu.java):


```
Action Menu:
         1) List Last Command.
         2) Undo.
         3) Redo.
         4) List Objects.
         5) Command: Push.
         6) Command: Lift.
         7) Command: Drop.
         8) Save Simulation.
         9) Back to Settings Menu.
         10) Exit.

Please an option listed above.
3
Redoing most recently undone command...
Command redone.
```

Save current state of simulation(ActionMenu.java):

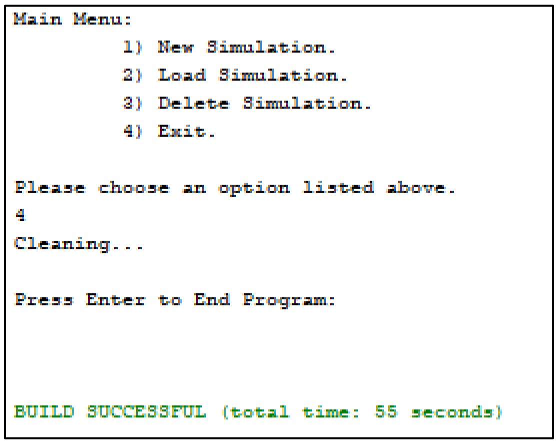

```
Action Menu:
         1) List Last Command.
         2) Undo.
         3) Redo.
         4) List Objects.
         5) Command: Push.
         6) Command: Lift.
         7) Command: Drop.
         8) Save Simulation.
         9) Back to Settings Menu.
         10) Exit.

Please an option listed above.
8

Save Menu:
         1) Save Simulation.
         2) Back to Simulation.
         3) Exit.

Please choose an option listed above.
1
Current PhysX Files:
Please input the name you would like for your save file:
group5
Saving to: C:\Users\Samuel\Documents\csusm\cs542\Group5Project\PhysX\group5.physX.
Save Complete.
```

Load simulation from .physX file(MainMenu.java):



```
Main Menu:
         1) New Simulation.
         2) Load Simulation.
         3) Delete Simulation.
         4) Exit.

Please choose an option listed above.
2

Load Menu:
         1) Load Simulation.
         2) Back to Main Menu.
         3) Exit.

Please choose an option listed above.
1
Current PhysX Files:
1: group5
Please input the name of the file you would like to load:
group5
File successfully loaded.
```

Terminate gracefully(All Menus):



```
Main Menu:
         1) New Simulation.
         2) Load Simulation.
         3) Delete Simulation.
         4) Exit.

Please choose an option listed above.
4
Cleaning...

Press Enter to End Program:



BUILD SUCCESSFUL (total time: 55 seconds)
```

Lessons Learned:

Samuel -

1) It is important to establish expectations and deadlines early in the development process to ensure that everyone is on the same page and work is distributed evenly.

2) Testing regularly for small incremental changes **is a must**. Without consistent and frequent testing (unit), integrating larger pieces of software becomes incredibly difficult (hard to tell if the component is bad or if multiple components are not connected properly).

3) Planning (requirement engineering) is the most important step. Without a clear goal of what you are building and what your milestones are, it is hard to pace your production and impossible to maintain the scope of the project.

Akhil –

1) The term project gave everyone a very good opportunity to get hands on experience to work with the design patterns which we learnt throughout the course and we also got to know when to use which design patterns.

2) Throughout the project we learnt how to cooperate with our teammates and how to fix the meeting hours suitable for everyone.

3) We used the Github Repository to make sure that everyone was aware of others work and also if we were struck somewhere then we could help out each other.

4) The most important thing which I learnt is that whenever we start a project we should know the requirements and the deadlines and work accordingly.

5) We must always have to work on the common parts first and later on we can move to the individual stuff so that no one has to wait for the other.

Rishi –

1) To complete the work with accordance to the schedule and not to delay the process and meet the deadlines.

2) To plan the project during the initial stages and evaluate the circumstance to avoid making changes in the project at the end.

3) Integration of various UMLs and codes into one form and building it with accordance to the goal of the project was the toughest part this helped us to be good developers in Programing language of Java and building of Use case diagrams.