Yiming Wang SE450 Final Project:

Asteroid

**Part 1. Discussion of the solution**

In this project, we should build an Asteroid game by Java. Some part of the required classes are given, such as shape factory, keyboard related classes, some shape models, part of ShapeDisplay class.

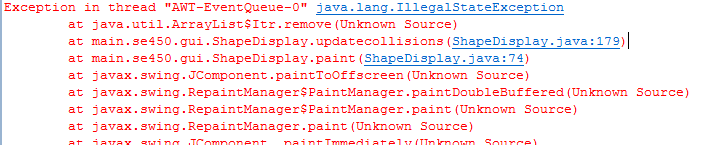
Basically, we need to make this game should be able to be played. The player ship should be able to move, fire, hyper space, protected by shield. For shapes, if it is hit, it should split into size down-graded shapes (if the hit shape is in size of small, it should disappear).

Steps I finished the project:

1. Start with ConfigurationParser, Configuration and ConfigurationManager, so the data in configuration.json (contains set-up data such as playership size, shot size) can be read.
2. Make the player ship can be draw out in the center of the “space”.
3. Create observables (fire, forwardThrust, HyperSpace), key / player ship would be observer. So the player ship can move and fire and hyper space.
4. Create Shield, which is a circle surround player ship and would disappear in certain time.
5. Create shot object. The start place of each shot would depends on where the player ship is (fire() handle this), and would disappear in certain time (). The shot move would be same idea with shape move.
6. Create singleton to store shapes, shots.
7. Create lineCollection for each shapes, playership. It is lines surrounding for collision test.
8. Create ChildrenFactory, which would make child shapes depends on input. If input is large square, children number from configuration is 2, ChildrenFactory would make 2 medium size of new square.
9. Create score observable, FinalProject is observer.
10. In ShapeDisplay class, call Collide.java, check kinds of collision (base on the lineCollection), :
    1. If shots collide with shape, and shapes size is small, then the shape disappeared (removed from shapeList). If the shape is not small, create new down-graded shapes. Each time of collision, score updated base on the shape size.
    2. If playership collides with shape with Shield on. Then, nothing happened to playership. Shapes re-bounded (X,Y become –X, -Y).

If playership collides with shape without shield on. Then, nothing happened to shape. Score set back to 0. The player ship will be re-placed to the center of the “space”.

In this part, it always come exception:



After times of test, I think this issue is due to the situation that more than one shot hit same shape in same time. I could not figure out the reason since I use ListIterator, and it is save to remove. Prof. Freund told me that it is because I use nested loop, the inner loop has delete the shape once. So, when the function try to delete the shape second time, exception would be throw out. Under the help of Prof. Freund, I create a flag to make sure each shape would not be deleted more than once. Also, for each new child shapes, I put it into a temp – ArrayList, then add the ArrayList to ShapeList after the nested loop finished. The problem solved.

1. Add the sound to each explosion action and keyboard input.

**Part 2. Discussion of patterns**

I used 3 kinds of pattern in this project: Factory, Singleton and Observer.

**Factory:**

There is 2 factory in this project:

1. ShapeFactory was given. This factory handles the input data from shape.json. It would automately create shapes by its type.

For example:

"type": "Circle", "left": 950, "top": 350, "right": 970, "bottom": 370, "x": 3, "y": 5, "rotation": 1, "color": -65536, "borders": "PassThru", "size": "large", "score": 10, "multiplier": 2, "children": 2

The type is Circle, then circle would be created. So, we do not need to worry about Shape has several different option like: Triangle, square, circle, ship.

The JSONFileShapeListFactory class called this factory after it read the json data. Then, the shapes created from this factory would be send to ShapeList.

1. ChildrenShapeFactory. This factory handles child shape creation. Since there are 4 different kinds of shapes: triangle, square, circle, ship. Each of them could have different size: medium, large. Different size would go with different size in display and different score. So, one factory to handle all this issue one time, is a great choice.

For example:

If a large size of square is input.

The children number is 2.  
The score is 10.

The multiplier is 2.

Then, 2 new medium size of square would be created. The display size would be 0.7 scale of the original square. The score of new medium square would be 2\*10 = 20. Multiplier would be still 2. Children number of new medium square would be 2.

**Singleton:**

There are several singletons in this project: ConfigurationManager , DisplayManager , ShapeLineCollectionList , ShapeList , ShotList , SoundManager .

Singleton create an object while making sure that only single object gets created. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class.

All of these are for one reason: save certain kind of data in only one place. Make sure we can always easy to get/update the ***right*** data. (if several different object, probably not)

Take ConfigurationManager.java and ShapeLineCollectionList.java as example:

1. ConfigurationManager. Since we only have one Configuration.json (only one line of record in this file). We would store the data in one Configuration object. This one Configuration object would be call and get data.

So, we need to make sure:

* 1. Only one Configuration in this project.
  2. We can easily to call it.

Singleton is the best choice. We can just use ConfigurationManager.get ConfigurationManager().getConfiguration() to reach and call the data we need store in that single Configuration object.

This singleton is called in many classes. Such as: PlayerShip (read playership size, location)

1. ShapeList. Same idea. We need to store all the shapes in one object, so it would be easy to call. Also make sure there is only that one object is used to store data. So, a singleton pattern is good choice.

This singleton is called in ShapeDisplay class (check collide, then remove shapes and add children shapes), FinalProject class (start the game, create original Shapes and put into ShapeList)

**Observer:**

Since this is a game, all the element display in the window is moving. And different action would be call by the keyboard. So, we need to keep the data of each object (shapes, shot, player ship) updated in time. Observer pattern is design to handle this kind of situation.

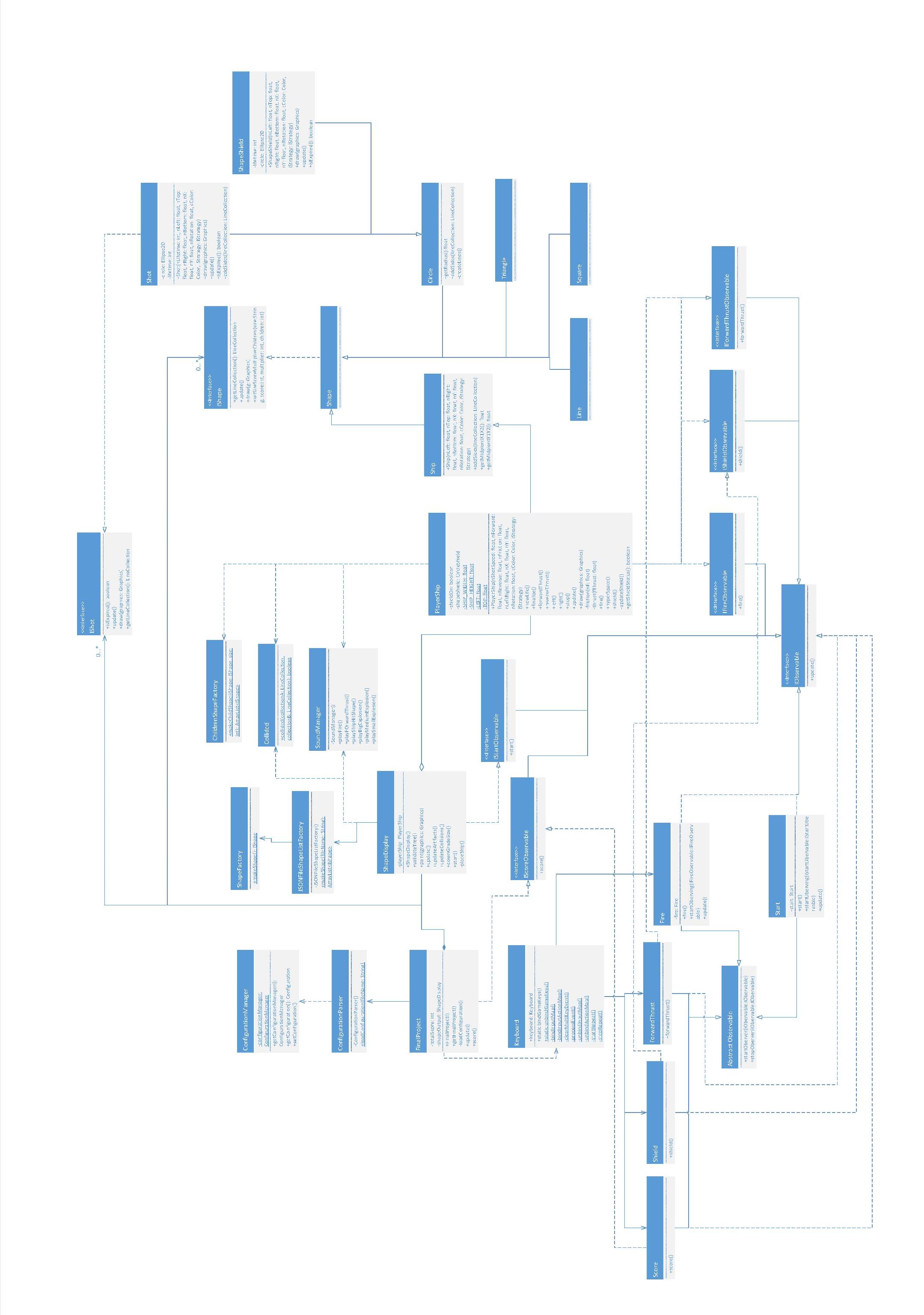
There are several action using observer pattern, including: forward thrust, backward thrust, left turn, right turn, fire, hyperspace, score, shield etc.

Use forward thrust of playership as example:

1. Up arrow key controls playership to move forward.
2. An interface IForwardThrustObservable with forwardThrust() abstract function created, this is the observer of forwardThrust event, it would perform forwardThrust() when notified.
3. ForwardThrust class is created to define how to manage the registered IForwardThrustObservable objects (observers) and how to notify observers when forwardThrust events occur. When forwardThrust () is called, i.e. changes are made, the observer would be updated.
4. Keyboard class binds up arrow key with statement: ForwardThrust.forwardThrust(). When up arrow key pressed, ForwardThrust.forwardThrust() would be called and observer would be notified of the change.
5. Playership class is registered to be an observer of forwardThrust event by implementing IForwardThrustObservable, and define playership action by overriding forwardThrust () to forward moving.

**Part3. UML Class diagram**

Please check UML.png for more detail.



**Part4. Javadocs**

Attached 3 javadocs screen shot from Eclipse: ChildrenShapeFactory, ShapeDisplay, PlayerShip.

For the Javadocs of all classes, please check from Eclipse.

