Assignment 3 – Design Patterns

Due Date: December 4th / 2022 at 23:59

Follow all of the instructions in this document exactly. Failure to follow instructions will result in a zero on this assignment.

How to submit:

- Each of you has a private git repo that you have invited the markers and Rob to as collaborators.
- Create a folder labeled "A3" at the root of your private repo, all of this assignment's source files must be at the ROOT of this folder (e.g. A3/Main.java, A3/GameBoard.java, etc…)
- Do NOT RAR or zip your files, they must be loose **as-is** in the A3 folder in your private git repo.
- Do NOT commit .class or any other compiled files to your A3 folder, there should be nothing but .java files in your A3 folder. Do not commit any files you used for testing, any readme files, any environment or IDE files or **anything but .java files or commands.txt**.

Requirements:

1. All code you write for this assignment must be written in Java
2. You do not require unit tests for this assignment
3. You must keep the supplied Main.java that implements public static void main() and takes one command-line argument.
4. Your code must compile and run via the following commands executed in **your A3 folder**:
   a. javac *.java
   b. java Main <path to input file>
5. Note the above command-line argument <path to input file>, this will be passed to your program by the TAs, and your program will process the file as described below. It will not be wrapped in <>'s obviously.
6. **If your code does not compile, you will receive zero on this assignment.**

Objective:

I am your boss. I just hired you and I'm ready to test you out on this new game our company is writing in Java. I read on your resume that you learned about design patterns when you were at Dalhousie. That means that you and I now speak a common language, the language of design patterns. I can tell you things like "I want you to implement that feature by using Mementos" and you should know what I'm talking about right? Right.

I have written the shell of a game that implements the old missile defence game from the good old days of video games. I want you to start adding to it. I want you to implement the mechanisms that allow for asteroids to impact the ground and destroy buildings they hit.

The code I wrote is available in the A3 folder of the course repository:
https://git.cs.dal.ca/rhawkey/csci5308

I used the following patterns to implement the game so far:

Pattern	**Classes Involved**	**Purpose**
Singleton		
	GameBoard	Central place where the game logic exists, processes the command file, manages the factory, holds the board composite
Abstract Factory		
	IAsteroidGameFactory	Interface of the Abstract Factory that instantiates all the game classes.
	AsteroidGameFactory	Concrete factory class that does the work.
Composite		
	BoardComponent	Abstract component class implements base Composite functionality
	Square	Composite that contains BoardComponent children, organized as a two dimensional grid on the board
	Building	A component that can belong to a composite, a Square can have many buildings. Buildings get hit by asteroids and destroyed after 2 hits.
	Asteroid	A component that belongs to a square, it starts at a certain height and falls one level at a time, when it hits 0 it impacts the square and does 1 damage to all buildings on the square
Command		

	Command	Abstract Command interface, takes a receiver object to execute the command on, and string arguments for the command.
	InitializeBoardCommand	Command that does the work of setting up the board.
	SpawnAsteroidCommand	Command that spawns an asteroid
	StartGameCommand	Command that starts the game, once the game is started when the building count hits 0 the game is over.
	TickCommand	Command that executes a variable number of "turns" of the game, each turn asteroids fall one level and potentially impact the ground.
State		
	IState	Abstract State pattern interface.
	SetupState	Returns that the game is not over while being setup.
	GameState	Returns that the game is over when the building count is 0.

The main program takes as input a text file name. In the text file are commands for the game to execute. There is one command on each line in the file. **You do not need to parse the file, that work is already done.** You **do** need to add to the factory though to make some new commands that you need to support.

Commands can have arguments, arguments are separated from the command by spaces, **you don't need to parse those either, they will be passed to any new Command objects you create assuming you connect them to the factory properly**.

The syntax for commands is as follows (arguments are defined in <>'s but <>'s do not appear in the file):

INIT <width of game board> <height of game board>

- Sets up the game board to be <width> Squares across, <height> high. Width is the X axis, height is the Y axis.

SPAWN_BUILDING <X index> <Y index>

- Creates a Building component and adds it to the Square composite at position X,Y on the board.

SPAWN_ASTEROID <X index> <Y index> <height>

- Creates an asteroid component and adds it to the Square composite at position X,Y on the board with the given initial height.

SPAWN_SHIELD <X index> <Y index>
- Creates a shield Decorator, and attempts to decorate the Square object at position X,Y (if there is one there).

START_GAME
- Changes the game state to GameState, meaning the game is over when the building count hits 0.

TICK <Number of ticks to execute>
- Executes one turn of the game, causing Asteroids to fall one level and potentially impact their Square. When impact happens all buildings / shields are reduced in health.

You must do the following:
1. Read all of the existing code, make sure you understand all the of the existing usage of design patterns and how everything fits together.
2. Finish creating the missing command SPAWN_BUILDING:
   a. Create a SpawnBuildingCommand class, SpawnAsteroidCommand is a good starting point
   b. Connect it to the MakeCommand method in the factory
   c. When Building objects are created the GameBoard.IncrementBuildingCount() method must be called.
3. Finish creating the missing command SPAWN_SHIELD:
   a. Write a Shield class that has 2 health.
   b. Use the Decorator pattern so that you can decorate Square objects with the Shield class (replace them in the composite hierarchy in their grid position with your decorator object).
   c. When the Shield's health drops to 0 it should no longer decorate its Square. The Square should be returned to its position in the composite hierarchy.
   d. When Shields are decorating Squares, asteroid impacts in that Square do not hurt the buildings in the Square, the Shield blocks the Asteroid while its health lasts.
   e. Write a SpawnShieldCommand class, attach it to the MakeCommand method in the factory. This should do the work of decorating the correct Square object with your Shield decorator.
4. Use the Observer design pattern to implement a system to allow Asteroids to notify the board (Composite hierarchy) that an Asteroid has impacted the Square that is the parent of the Asteroid:
   a. Define a subject interface that BoardComponent objects can attach themselves to.
   b. Define a concrete subject class for Asteroid impacts.

c. Put an instance of the concrete subject class somewhere that all members of the board composite hierarchy can get to so they can attach themselves to the subject and get updated when the subject object is notified of an asteroid impact.
d. Implement the Update() in the Building class and Shield classes so they take 1 point of damage if they are impacted by an asteroid. If a Shield hits 0 health it is removed as a decorator from the Square object it is decorating. If a Building hits 0 health it is destroyed, and removed from its parent Square object. Make sure you call GameBoard.DecrementBuildingCount() when this happens.

Marking Rubric:

- Code does not compile = Automatic 0 on entire assignment.
- Program correctly functions and correctly processes commands as defined in this document: 20%
- Proper use of design patterns as instructed in this document: 60%
- Code readability / understandability: 20%
- If you have anything that is not a .java file or commands.txt file in your A3 folder: -5%
- Late assignments not accepted