CSCI 6626 Object Oriented Principles and Practices

Documentation for Sudoku Game:

My feel about the project : Proper design, saves time and produces best results.

Classes Present:

Board
Cluster
Can view
Canvas
Diagonal Board
Frame
Game
Grid Char
My Stack
Mixed Char
Square
State
Sixy Board
Traditional Board
Tools
Viewer

Exception Classes:

Game Error:
Wrong game type, Wrong board input, Extra Junk input, Wrong row or column, Empty Stack

Stream Error:
Wrong file, No file, Bad file, Blank file

Main method:

Takes a filename from command line input. Catches all stream and game logic exceptions
If game logic exception occurs it asks for a new file in the console. Instantiates game class and jumps to run method.

Game Class:

Execution starts at the game constructor which has file name as parameter, reads the contents of the file and creates a board object by supplying proper arguments for the board constructor. Based on the contents of the file appropriate board is constructed. Control from here is passed to the board class. After this, game's run method is called where a menu choice is given to the user which has options to play the game. This is the point where the entire game logic interacts with GUI.

Board:
Execution starts from the concerned board constructor (I mean whether Sixy, Diagonal or Traditional). Using ctor (Constructor initializer) its super class's constructor is called to do necessary job.

The base board class creates N*N square objects and initializes their state them by using the ifstream object passed as a parameter. Creates clusters and also performs Shoop operation. Inherits can view class and implements it's pure virtual functions. Creates frame by interacting with frame class. Restores state of the board by assigning current states of the squares in the board.

Square:
Creates state object. Calls cluster Shoop on each cluster for a particular square. Delegates concerned operations to state.

State:
Marks the square when required and updates possibility list of a square.

Cluster:
Creates N square pointers and initializes them. Calls turn off on each of N squares with the value in that square. This value comes from square class when Shoop cluster method is called. Which is a method in cluster class.

Frame:
Creates an N*N state objects. Initializes them with state objects from Board class. Uses copy constructor. No problem because state has no dynamic parts. Have options to send frame objects back when required, serialize and realize. Serialize method takes an output stream object as a parameter and writes to a file (all the N*N state objects). Realize method takes an open stream object as parameter and reads the states of all N*N squares from file and creates a new frame.

My Stack:

It is an adapter class which adapts a vector standard template and overrides the methods present in it. We use this data structure to store frame objects. It initializes the template with a frame pointer.

Exceptions:
It has 2 main classes, one is game logic exception class and other is stream exception class. There are 5 child classes which inherit from game logic class and 4 child classes which inherit from stream exception class. These child classes override print method from parent class.

What I have learned through the project and lectures are ?

- Let the class have ultimate access to its data members.
- To communicate with other other classes, delegate
- To provide restricted access use const and necessary modifiers (Public, Private, Protected).
- Do not use global variables. Singleton design pattern or static const variables are remedy
- I learned that we can define our own operators in c++ and by this we can print an object as if we print a variable, and we can do many more
- Handle dynamically created members yourself.
- If I feel that I need to repeat the code, I will analyze on why to repeat, if there are variants exists I will use inheritance and polymorphism, if necessary. Else, I will use functions.
- I will use comments that are precise.
- The important principle that I learned is that I will first design and then code, rather than code, then make mistakes and redesign
- Even though recursion works at all places, it should be used only at certain places where it is necessary.
- Make variable names meaningful
- I have implemented the adapter design pattern, which is used to blend required interface with required functions taken from a template.

Student Name : Sreenikhil Kollu.