Assignment #4

Langlois, Matthew (7731813)

Yaraskavitch, Matthew (6301664)

November 30, 2015

E128.

b) **Adapter pattern**. Create an adapter that masks the operations and changes performed on the *RegularPolygon* objects.

c) **Factory pattern** with a generic class being an *Image* and the specific class being the individual images. When a new image is created, destroy the old instance to free space in memory. I.e. a photo viewing application on a computer that only loads an image into memory and displays it when the image’s path is selected in the viewer application.

d**) Façade.** Create a specific API for these classes to use via a façade pattern. Only expose the minimum level of complexity to simplify things for the other subsystems being implemented.

e) **Observer pattern** to watch the input and react accordingly based upon the type of input. Implementation may require the use of other patterns (i.e. adapters) for each specific type of input.

E135.

a) We have a **façade** pattern (i.e. Hotel is likely a façade that maps its operations to other sub components). We also have a **general hierarchy** because destroying the hotel object would also destroy its quality level. Also, hotel has many rooms within it. **Singleton** pattern (i.e. a specific hotel may only have one instance of itself, since we don’t cover the concept of franchises in this design).

b) The updated version of the UML diagram could be improved using the observer pattern to instantly notify users when a room becomes available. This would be done by making the room observable and then users could be observers, so when the observable room is free then the users are notified that the room is available for booking. Another pattern which can be used is the adapter pattern. An adapter pattern can be used to create a special version of a room for example rooms which may accommodate guests with disabilities. This will allow a special type of room to be created within the hotel system. Lastly, if this system were to become public so random people can query their API then a read only pattern will need to be implemented so that people will be unable to modify the rooms from the external API.

E152.

c) See **E152\_Sequence\_Diagram.pdf** for the scanned version of the sequence diagram.

E169.

a) See ***E169\_State\_Machine.ump*** for raw code and ***E169\_State\_Machine.pdf*** for rendered image. Also note, both have been rendered via Umple.

**PART 2**

a) See ***Survey\_Class\_Diagram.ump*** for raw code and ***Survey\_Class\_Diagram.pdf*** for rendered image. Also note, both have been rendered via Umple.

b) See ***Survey\_State\_Diagram.ump*** for raw code and ***Survey\_State\_Diagram.pdf*** for rendered image. Also note, both have been rendered via Umple.

c) The class diagram developed was based purely upon the specifications given in the question. In order to improve the design, multiple improvements could be made. First, the survey system itself should not have direct access to both the master and slave node servers. This detail should be hidden behind a **façade** pattern. In this way, the survey system would only be communicating to the data servers through some sort of well-defined API. Within the façade, the backup and retrieval of data would be masked away from the application. This way, the same server system could be used to store data for many types of applications concurrently, as opposed to just survey instances.

Secondly, the façade could be improved by implementing a **singleton** pattern. Here, a “ServerConnector” singleton could be created that would handle parallel connections to the exterior server façade that manages the master and node servers. This way, the entire application backend could be simplified to only show the bare minimum complexity to the developers implementing the user facing portion of the server system.