Assignment: Design Patterns

Paul Traxler – 11-26-18

Github project link: https://github.com/ptrax/Assn6-DesignPatterns

1. **Design Patterns**

In this assignment I used 4 design patterns: Singleton, Builder, Decorator, and Mediator. Below is a brief description of these patterns and how they were used in my project.

* 1. **Singleton**

The Singleton pattern is one that allows only one copy of an object to be in existence. This was used for the Apiary class as per the project requirements. An apiary is created by getting the current Apiary instance, and if an instance doesn’t exist one is created. With the Apiary class I fulfilled the requirement of being able to have a single apiary with multiple hives in it.

* 1. **Builder**

The builder pattern was used for the Beehive class. What this pattern does is allows you to specify certain required parameters to be used when creating your object and then create add on methods to set up other optional parameters, finished off by a build method that actually creates the object in question. It makes it easier to be able to get the necessary things in there and pick and choose what else you want, and make sure you don’t forget anything. For my Beehive, if an optional parameter isn’t used a default value is used. With the Beehive class I fulfilled several requirements; the beehive can’t have other beehives, it is comprised of a network of rooms, and it has rooms for spawning bees.

* 1. **Decorator**

The decorator pattern allows you to create an interface for the basic aspects of your desired object and then create decorator classes to add on to this object down the road. I used this pattern for my rooms, because it allowed me to define basic room behavior that all rooms shared, and then decorate the rooms with specific functionality based on if they were spawn, rest, or food rooms. With the rooms I fulfilled the requirement of the beehive having rooms.

* 1. **Mediator.**

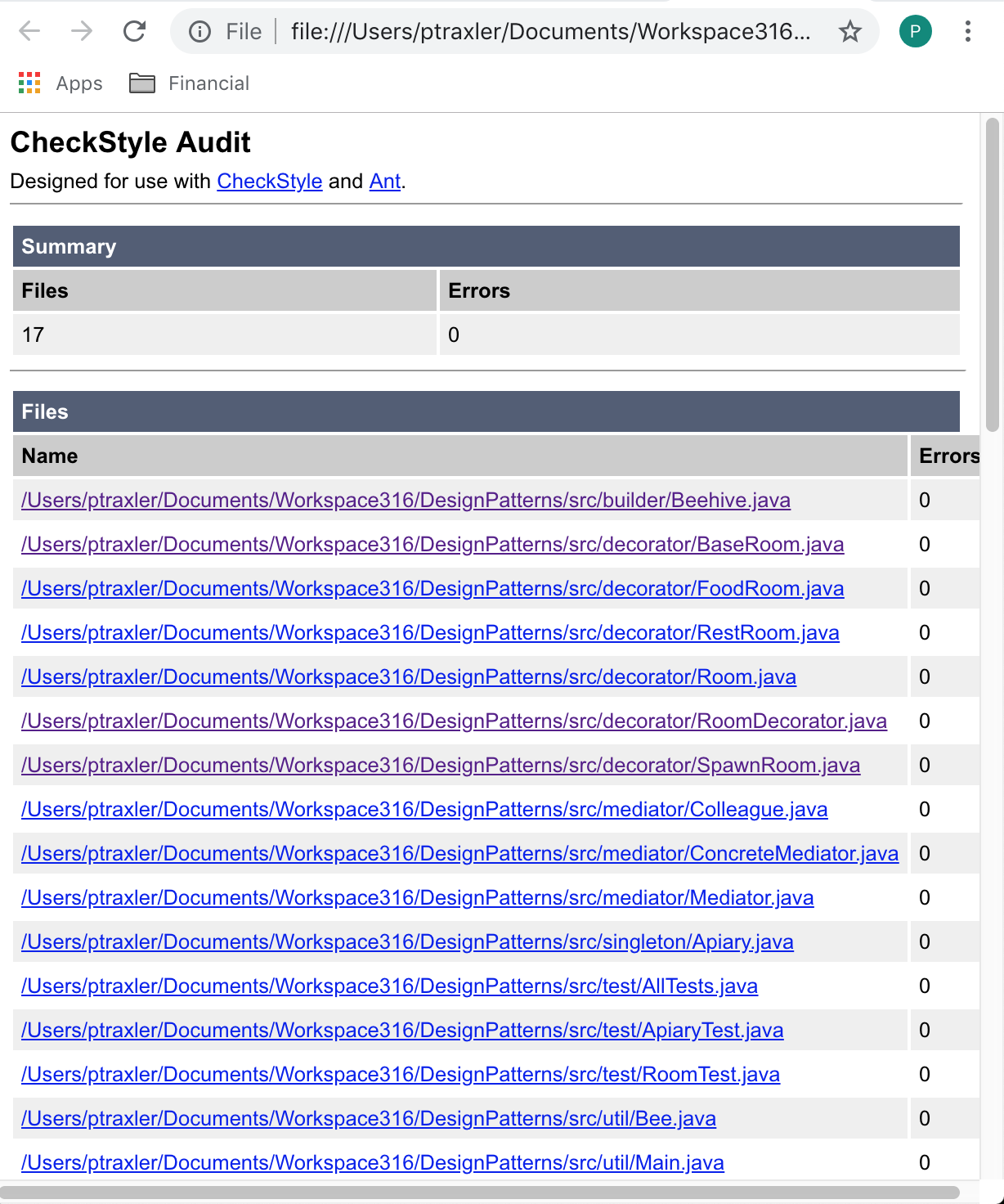
The mediator class is used for communication between objects. The central mediator has the ability to receive messages from Colleague classes (everything is a colleague in my program, apiary, beehive, rooms, etc.) and act on that information. As I have it the messages just print to the console saying that they were received, but more time would allow you to route messages from the central mediator to other objects and make decisions based on the messages. With the Mediator I fulfilled the requirement of communication between objects and gave it the ability to operate on a tick, but didn’t implement the tick.

1. **Running the program**

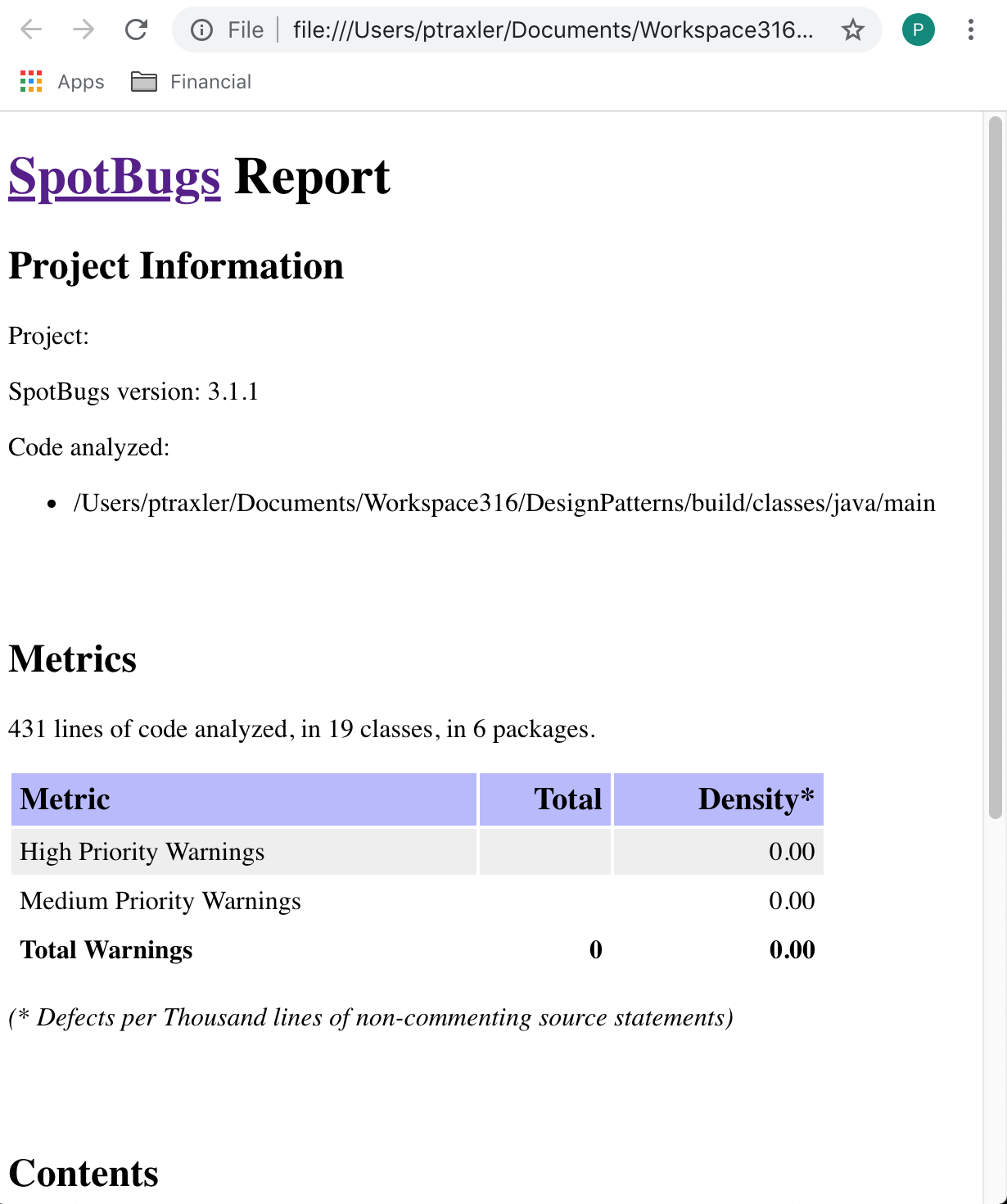
To run the program you have two options. First, you can run it through a gradle run command. You could also do gradle build and then navigate to the build/lib folder and run the JAR file that is created. The console will then print the results of the demonstration of the patterns.

1. **Reports**

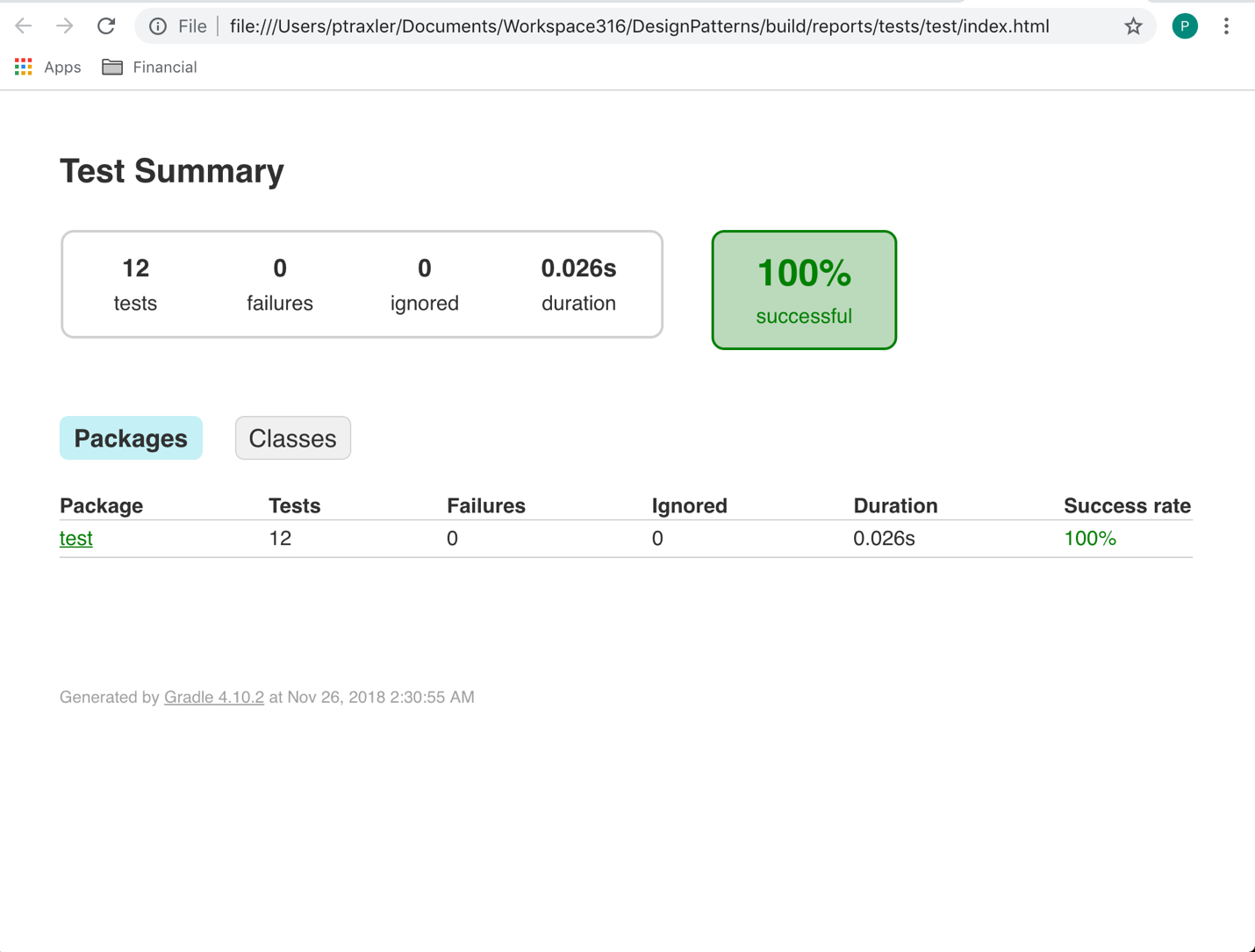
**Checkstyle**

****

**SpotBugs**

****

**JUnit Tests**

****