Most Difficult Part(s) of the Assignment

There were two components to this assignment that I found the most difficult. They included:

- Understanding Dependency Inversion. It was not stated in the document write up on how students should use Dependency Inversion. I was confused as to how I should implement the interface in my design, and how to inherit subclasses and superclasses.
- Creating a makefile for a Java program packaged in another program. I had a very difficult time making a java file that catered to Java programs that are packaged in a folder. My shell could not find my main class, and thus, all my testing was done in the Netbeans IDE. Futher iterations need to be done to ensure makefile (or another script) functionality..

Status

-Missing makefile (unsure if current Makefile will run). Everything else is completed besides this.

Code

Important Statistics

Important statistics about my code include:

1. How I implemented the LampInterface. I overode the switchOn and switchOff methods so TableLamp and Button objects could use them.

```
package lamps;
public interface LampInterface {
    public void switchOn();
    public void switchOff();

}
```

2. Button inherits Lightbulb and implements the LampInterface. By doing so, it turns on a Lightbub when it is turned on:

```
package buttons;
import lightbulbs.Lightbulb;
import lamps.*;
public class Button extends Lightbulb implements LampInterface
{

     public Button()
     {

     }

     public void switchOn()
     {

          System.out.println("Button switched to ON");
                on();
     }


     public void switchOff()
     {
          /*stdout="Button switched to OFF";
```

```
            System.out.println("Button switched to OFF");
                  off();
      }

}
```

3. TableLamp overrides the switchOn() and switchOff() methods:

```
package lamps;
import lightbulbs.Lightbulb;
public class TableLamp extends Lightbulb implements LampInterface
{
      String stdout="nothing";
      public TableLamp()
      {

      }

    @Override
      public void switchOn()
      {
            System.out.println("TableLamp switched to ON");
                System.out.println("Button switched to ON");
                on();
      }

      @Override
      public void switchOff()
      {
            System.out.println("TableLamp switched to OFF");
                System.out.println("Button switched to OFF");
                off();
      }


}
```

4. PushdownButton is utilized through a boolean variable. The boolean changes once the PushButton() method is called. If push is true, the button is turned on (and so is the lamp). Otherwise, the button is turned off (and so is the lamp).

```
package buttons;
import lightbulbs.Lightbulb;

public class PushdownButton extends Lightbulb
{
      boolean push=true;
      public void PushdownButton()
      {

      }

      public void PushButton()
      {
            if(push)
            {
                  System.out.println("Push Button switched to ON");
                        on();
                  push=false;
            }
            else
```

```
            {
                  System.out.println("Push Button switched to OFF");
                        off();
                  push=true;
            }


      }

}
```

Unit Test Code

All unit tests were tested with assertEquals. Examples are shown below. The only unit test that stray from this is when the pushbuttondownoff test was tested. The prinlin statement had to be reset in order for the stream to know the object was called twice.

Example 1:

```
@Test
    public void lightbulbOnTest()
    {
        Lightbulb lb=new Lightbulb();
        lb.on();
        assertEquals("Lightbulb on\n", println.toString());
    }
```

Example 2:

```
@Test
    public void buttonOffLightbulbOffTest()
    {
        Button bu=new Button();
        bu.off();
        assertEquals("Lightbulb off\n", println.toString());
    }
```

Example 3:

```
@Test
    public void pushDownButtonOffTest()
    {
        PushdownButton pd=new PushdownButton();
        pd.PushButton();
        println.reset();
        pd.PushButton();
        assertEquals("Push Button switched to OFF\n" + "Lightbulb off\n",
println.toString());
    }
```

Unit Test Coverage

Unit Test Coverage was utilized by using JaCoCo, a NetBeans plugin. JaCoCo uses jCoverage. The report indicates all methods besides the constructors were tested. A picture of the coverage is also shown below.

JaCoCoverage analysis of project "CS_445_HW_3" (powered by JaCoCo from EclEmma)

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
lamps		96%		n/a	1	5	1	30	1	5	0	2
buttons		98%		100%	1	7	1	19	1	6	0	2
lightbulbs		100%		n/a	0	3	0	6	0	3	0	1
Total	4 of 128	97%	0 of 2	100%	2	15	2	55	2	14	0	5

Cyclomatic Complexity

Excluding Unit Tests, Test Runner, and LampInterface

Edges

Button: 2
Lightbulb: 2
TableLamp: 2
PushdownButton: 2
TableLampClient: 3

Nodes

Button: 1
Lightbulb: 1
TableLamp: 1
Nodes: 2
Nodes: 1
TableLampClient: 1

Exit Points

Button: 1
Lightbulb: 1
TableLamp: 1
Nodes: 1
TableLampClient: 1

Total

Edges: 11
Nodes: 7
Exit Points: 5

Edges-Nodes+2*(Exit Points)
11-7+10=14

Does not exceed 20