FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

SEMESTER 1, 2020/2021

WIF3007 DESIGN PATTERNS

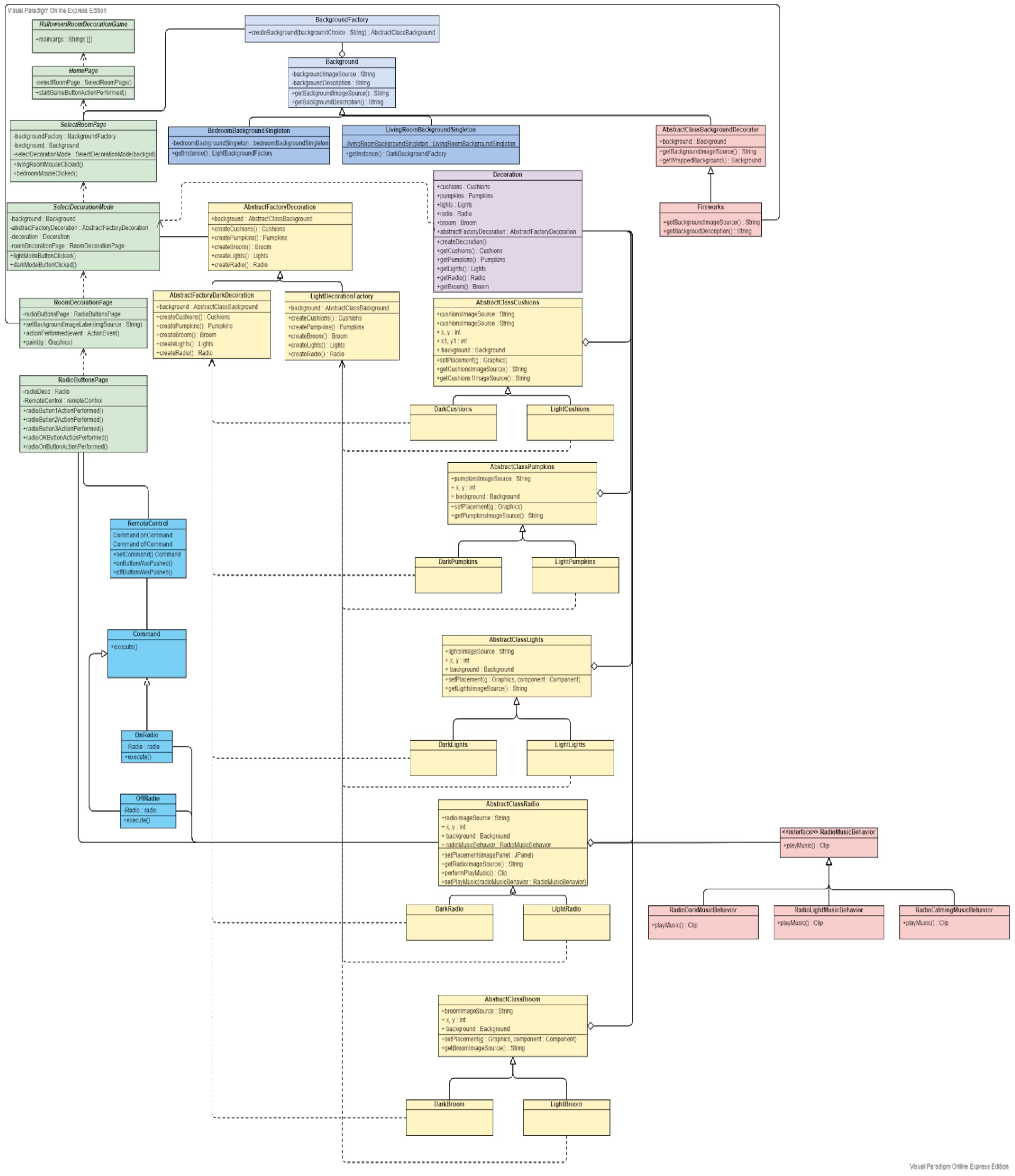
ASSIGNMENT

WONG SHEOK YAN ALICIA	17185416/1
JASMIN BT RAPIAN	17159737/1
LIOW JING WEN	17119965/1
HARITH ARIFFIN BIN KAMAL ARIFFIN	17137026/1
ALIYA FATIHAH BINTI MOHAMED SIDEK	17080861/1

LECTURER: DR. SU MOON TING

Table of Contents

1. Overall UML Class Diagram



Visual Paradigm Online Express Edition
HalloweenRoomDecorationGame
+main(args : Strings [])
HomePage
-selectRoomPage : SelectRoomPage()
+startGameButtonActionPerformed()
SelectRoomPage
-backgroundFactory : BackgroundFactory
-background : Background
-selectDecorationMode : SelectDecorationMode(backgrd)
+livingRoomMouseClicked()
+bedroomMouseClicked()
SelectDecorationMode
-background : Background
-abstractFactoryDecoration : AbstractFactoryDecoration
-decoration : Decoration
-roomDecorationPage : RoomDecorationPage
+lightModeButtonClicked()
+darkModeButtonClicked()
RoomDecorationPage
-radioButtonsPage : RadioButtonsPage
+setBackgroundImageLabel(imgSource : String)
+actionPerformed(event : ActionEvent)
+paint(g : Graphics)
RadioButtonsPage
-radioDeco : Radio
-RemoteControl : remoteControl
+radioButton1ActionPerformed()
+radioButton2ActionPerformed()
+radioButton3ActionPerformed()
+radioOKButtonActionPerformed()
+radioOnButtonActionPerformed()
RemoteControl
Command onCommand
Command offCommand
+setCommand() Command
+onButtonWasPushed()
+offButtonWasPushed()
Command
+execute()
OnRadio
- Radio : radio
+execute()
OffRadio
-Radio : radio
+execute()
BackgroundFactory
+createBackground(backgroundChoice : String) : AbstractClassBackground
Background
-backgroundImageSource : String
-backgroundDescription : String
+getBackgroundImageSource() : String
+getBackgroundDescription() : String
BedroomBackgroundSingleton
-bedroomBackgroundSingleton : bedroomBackgroundSingleton
+getInstance() : LightBackgroundFactory
LivingRoomBackgroundSingleton
-livingRoomBackgroundSingleton : LivingRoomBackgroundSingleton
+getInstance() : DarkBackgroundFactory
AbstractClassBackgroundDecorator
+background : Background
+getBackgroundImageSource() : String
+getWrappedBackground() : Background
Fireworks
+getBackgroundImageSource() : String
+getBackgroudDescription() : String
AbstractFactoryDecoration
+background : AbstractClassBackground
+createCushions() : Cushions
+createPumpkins() : Pumpkins
+createBroom() : Broom
+createLights() : Lights
+createRadio() : Radio
AbstractFactoryDarkDecoration
+background : AbstractClassBackground
+createCushions() : Cushions
+createPumpkins() : Pumpkins
+createBroom() : Broom
+createLights() : Lights
+createRadio() : Radio
LightDecorationFactory
+background : AbstractClassBackground
+createCushions() : Cushions
+createPumpkins() : Pumpkins
+createBroom() : Broom
+createLights() : Lights
+createRadio() : Radio
Decoration
+cushions : Cushions
+pumpkins : Pumpkins
+lights : Lights
+radio : Radio
+broom : Broom
+abstractFactoryDecoration : AbstractFactoryDecoration
+createDecoration()
+getCushions() : Cushions
+getPumpkins() : Pumpkins
+getLights() : Lights
+getRadio() : Radio
+getBroom() : Broom
AbstractClassCushions
+cushionsImageSource : String
+cushionsImageSource : String
+ x, y : int
+ x1, y1 : int
+ background : Background
+setPlacement(g : Graphics)
+getCushionsImageSource() : String
+getCushions1ImageSource() : String
DarkCushions
LightCushions
AbstractClassPumpkins
+pumpkinsImageSource : String
+ x, y : int
+ background : Background
+setPlacement(g : Graphics)
+getPumpkinsImageSource() : String
DarkPumpkins
LightPumpkins
AbstractClassLights
+lightsImageSource : String
+ x, y : int
+ background : Background
+setPlacement(g : Graphics, component : Component)
+getLightsImageSource() : String
DarkLights
LightLights
AbstractClassRadio
+radioImageSource : String
+ x, y : int
+ background : Background
+ radioMusicBehavior : RadioMusicBehavior
+setPlacement(imagePanel : JPanel)
+getRadioImageSource() : String
+performPlayMusic() : Clip
+setPlayMusic(radioMusicBehavior : RadioMusicBehavior)
DarkRadio
LightRadio
<<interface>> RadioMusicBehavior
+playMusic() : Clip
RadioDarkMusicBehavior
+playMusic() : Clip
RadioLightMusicBehavior
+playMusic() : Clip
RadioCalmingMusicBehavior
+playMusic() : Clip
AbstractClassBroom
+broomImageSource : String
+ x, y : int
+ background : Background
+setPlacement(g : Graphics, component : Component)
+getBroomImageSource() : String
DarkBroom
LightBroom
Visual Paradigm Online Express Edition

2. Design Patterns Applied

2.1. Simple Factory

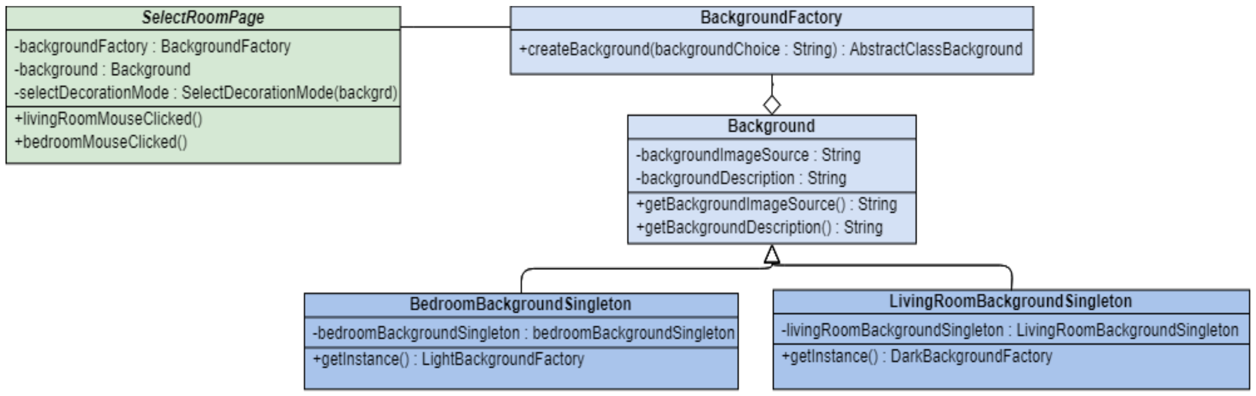

SelectRoomPage
-backgroundFactory : BackgroundFactory
-background : Background
-selectDecorationMode : SelectDecorationMode(backgrd)
+livingRoomMouseClicked()
+bedroomMouseClicked()
BackgroundFactory
+createBackground(backgroundChoice : String) : AbstractClassBackground
Background
-backgroundImageSource : String
-backgroundDescription : String
+getBackgroundImageSource() : String
+getBackgroundDescription() : String
BedroomBackgroundSingleton
-bedroomBackgroundSingleton : bedroomBackgroundSingleton
+getInstance() : LightBackgroundFactory
LivingRoomBackgroundSingleton
-livingRoomBackgroundSingleton : LivingRoomBackgroundSingleton
+getInstance() : DarkBackgroundFactory

Figure 2.1.1 UML Class Diagram for Simple Factory

The figure above shows a part of the UML Class diagram that depicts the Simple Factory method. Simple factory is used in our system for the selection of backgrounds. The `BackgroundFactory` referring to appendix 5.1.1 is an implementation of simple factory that refers to concrete background classes. There are two concrete background classes that represent the bedroom and living room respectively. These two classes extend the abstract `Background` class. By using simple factory pattern, the creation details of the background is abstracted from the client class (`SelectRoomPage`). The client class only needs to know that by calling the `BackgroundFactory` class, a background object is returned.

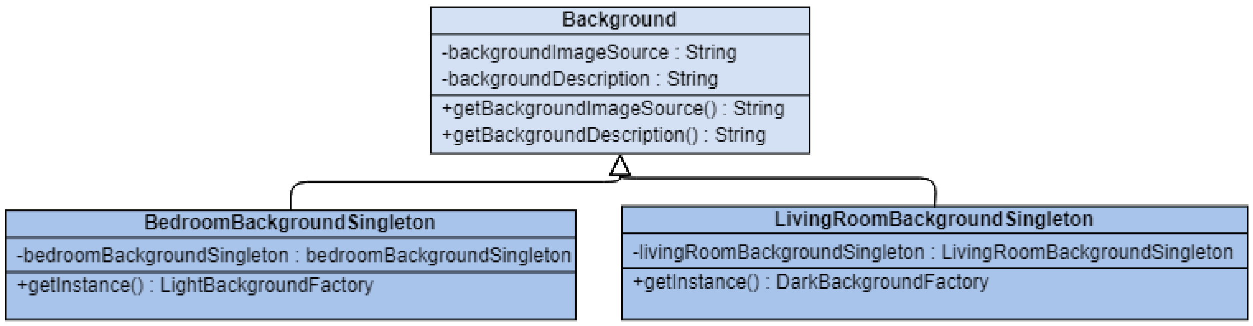
The `createBackground`() method will return a Background object which is either the living room or the bedroom, depending on the user input.The Background class is an abstract class which can be found in appendix 5.1.2. The `getBackgroundImageSource`() returns the image for the background. The `getBackgroundDescription`() is an abstract method that will return the background description.

The `LivingRoomBackgroundSingleton` class (Appendix 5.1.3) is a concrete class that extends the Background class and overrides the `getBackgroundDescription`() method to return ‘livingroom’.

The `BedroomBackgroundSingleton` class (Appendix 5.1.4) is a concrete class that extends the Background class and overrides the `getBackgroundDescription`() method to return ‘bedroom’.

Figure 3.2 under Screen Captures of GUI displays the page for room selection.The figure is the GUI for the select room page that is the client class that will call the `BackgroundFactory` class. The GUI displays the two options for the room which is either the living room or the bedroom. When the user selects the room intended, the client class will call the specific method for the room in the SelectRoomPage class that will call `createBackground`() method from `BackgroundFactory` class. You may refer to the corresponding codes in Appendix 5.1.5. Figure 3.4 and 3.6 from Screen Captures of GUI are the respective display for the bedroom and living room page.

2.2. Singleton Design Pattern



Background
-backgroundImageSource : String
-backgroundDescription : String
+getBackgroundImageSource() : String
+getBackgroundDescription() : String
BedroomBackgroundSingleton
-bedroomBackgroundSingleton : bedroomBackgroundSingleton
+getInstance() : LightBackgroundFactory
LivingRoomBackgroundSingleton
-livingRoomBackgroundSingleton : LivingRoomBackgroundSingleton
+getInstance() : DarkBackgroundFactory

Figure 2.2.1 UML Class Diagram for Singleton Design Pattern

Figure 2.2.1 displays the class diagram for the Singleton design pattern.There are two classes that apply the design pattern which are the `LivingRoomBackgroundSingleton` and the `BedroomBackgroundSingleton` (Appendix 5.1.3 and 5.1.4 respectively). The singleton pattern is implemented to ensure that only one instance of the background is created. The background object is a shared resource for the other objects associated with decorating the background. It is used to ensure that all the decorations are added to the same instance of the background.

Both of the classes `LivingRoomBackgroundSingleton` and `BedroomBackgroundSingleton` implement singleton design pattern.The constructors are declared as private to ensure that only Singleton can instantiate the class. The `getInstance`() method is the only way to instantiate the class and return an instance of it if it exists.

2.3. Abstract Factory Design Pattern

Figure 2.3.1 illustrates the corresponding part of the UML Class Diagram for Abstract Factory Design Pattern.

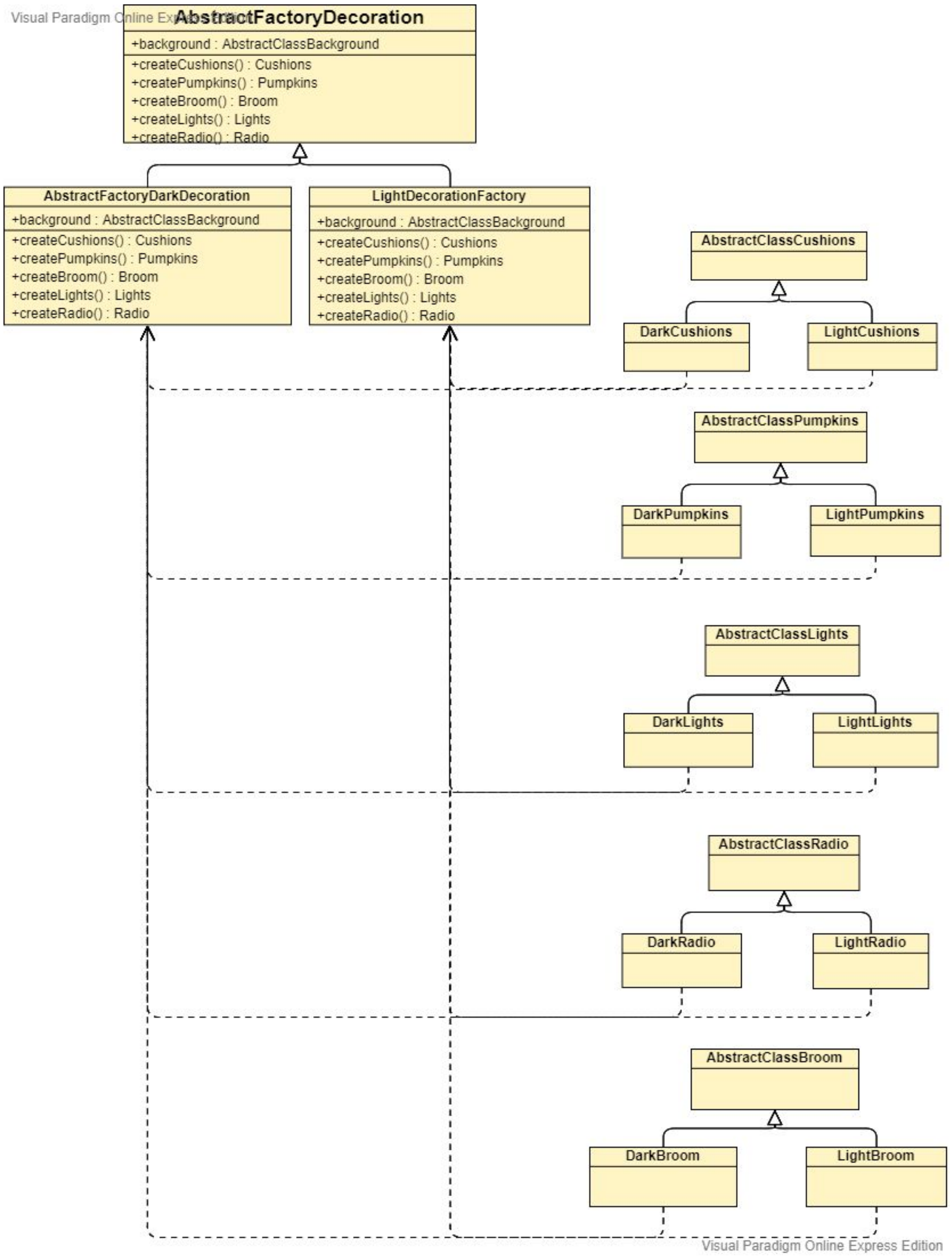


Figure 2.3.1 UML Class Diagram for Abstract Factory Design Pattern

To make our decorating application more interesting, the theme of decoration can be selected by the user based on his or her aesthetic preferences. We offer a selection of dark-themed decorations and light-themed decorations for the users to adorn their room. This is when Abstract Factory Design Pattern comes into the picture as it provides an interface for creating families of related or dependent objects without specifying their concrete classes. With Abstract Factory Design Pattern, families of Halloween decorations can be produced for dark-themed and light-themed decorations respectively, through the concrete factories that extend from an Abstract Factory interface.

In terms of maintainability, the loose coupling between the factories that create the decorations and the client codes allows us to implement a variety of factories that can produce decorations for other themes in the future. It also eases substitution for different factories to produce decorations of different themes (or families) because the actual decoration products are decoupled from the client codes.

In our application, an Abstract Factory, `AbstractFactoryDecoration` defines the required method for the creation of decorations, as shown in Appendix 5.2.1.

```
public abstract Cushions createCushions();
public abstract Pumpkins createPumpkins();
public abstract Lights createLights();
public abstract Radio createRadio();
public abstract Broom createBroom();
```

There are two concrete factories that extend from the Abstract Factory class, namely `AbstractFactoryDarkDecoration` and `AbstractFactoryLightDecoration` to create a family of decorations for dark theme and light theme respectively. In the concrete factories, the abstract methods to create decorations will be overridden to return relevant decoration products, as shown in Appendix 5.2.2 and 5.2.3 respectively.

Each of the decoration products inherit the background and x, y coordinates for placement in the background from their respective parent class, along with the method for setting the decoration placement in the background. Appendix 5.2.13 displays the `Pumpkins` parent class as an example. In the children decoration classes, the image sources will be defined. Appendix 5.2.14 shows the `DarkPumpkins` child class as an example.

After the user runs the application, a page to select a room will be displayed and after the choice has been made, a page to select the theme of decorations will be displayed, as shown in Figure 3.3.

The respective factories to produce decorations for the selected theme will be initialized after the choice is made. Appendix 5.2.16 shows the initialization of `new AbstractFactoryLightDecoration(background)` and Appendix 5.2.17 shows the initialization of `new AbstractFactoryDarkDecoration(background)` after the `lightModeButtonMouseClicked()` or `darkModeButtonMouseClicked()` is triggered in `SelectDecorationMode` page respectively.

2.4. Strategy Design Pattern

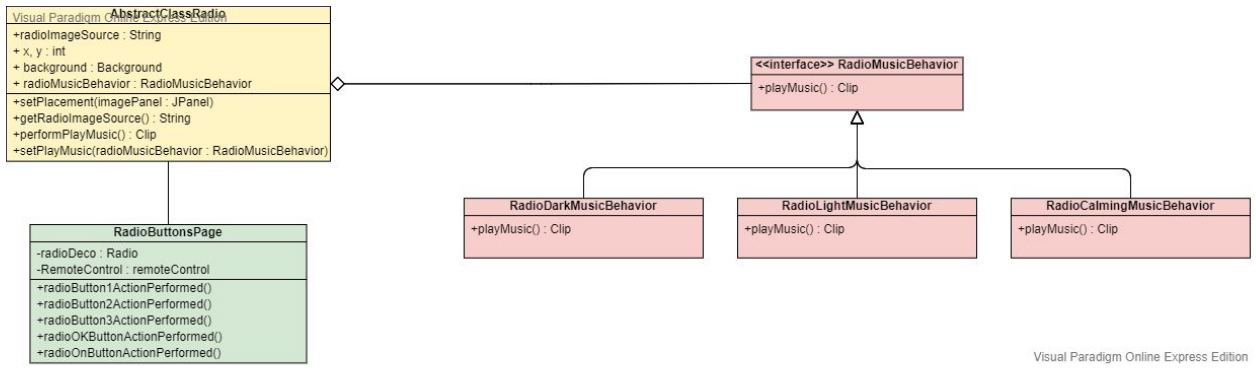

AbstractClassRadio
+radioImageSource : String
+ x, y : int
+ background : Background
+ radioMusicBehavior : RadioMusicBehavior
+setPlacement(imagePanel : JPanel)
+getRadioImageSource() : String
+performPlayMusic() : Clip
+setPlayMusic(radioMusicBehavior : RadioMusicBehavior)
<<interface>> RadioMusicBehavior
+playMusic() : Clip
RadioDarkMusicBehavior
+playMusic() : Clip
RadioLightMusicBehavior
+playMusic() : Clip
RadioCalmingMusicBehavior
+playMusic() : Clip
RadioButtonsPage
-radioDeco : Radio
-RemoteControl : remoteControl
+radioButton1ActionPerformed()
+radioButton2ActionPerformed()
+radioButton3ActionPerformed()
+radioOKButtonActionPerformed()
+radioOnButtonActionPerformed()
Visual Paradigm Online Express Edition

Figure 2.4.1 UML Class Diagram for Strategy Design Pattern

From the figure above (Figure 2.4.1), the strategy design pattern (classes in red) is implemented to the AbstractClassRadio by allowing the user to choose the type of background music that they wished. There are three types of music provided to the users, hence, the radio has three behaviours which are, RadioDarkMusicBehavior referring to Appendix 5.4.1, RadioLightMusicBehavior referring to Appendix 5.4.2 and RadioCalmingBehavior referring to Appendix 5.4.1.

To integrate with the current system, the behavior of the music also corresponds to the room theme with a setter and getter method called setPlayMusic() and performPlayMusic() in Radio class referring to Appendix 5.4.5.

```
public void setPlayMusic(RadioMusicBehavior radioMusicBehavior){
        this.radioMusicBehavior = radioMusicBehavior;
    }

public Clip performPlayMusic(){
        this.clip = this.radioMusicBehavior.playMusic();
        return this.clip;
    }
```

By default in the dark theme that extends Radio class, RadioDarkMusicBehavior will be executed referring to Appendix 5.4.8 and in light theme that is also extending Radio class the RadioLightMusicBehavior will be executed referring to Appendix 3.4.7.

```
  public DarkRadio(Background background) {
        super(background);
        RadioMusicBehavior darkMusic = new
RadioDarkMusicBehavior();
```

```
        setPlayMusic(darkMusic);   }

  public LightRadio(Background background) {
        super(background);
        RadioMusicBehavior lightMusic = new
RadioLightMusicBehavior();
        setPlayMusic(lightMusic);}
```

In strategy design pattern, we could choose the algorithm during runtime by exploiting polymorphism by programming it to an interface, as in this system, the interface is RadioMusicBehavior referring to Appendix 5.4.5. This interface is implemented to the three behaviour classes of the radio.

```
public interface RadioMusicBehavior {
    public Clip playMusic();
}

public class RadioDarkMusicBehavior implements
RadioMusicBehavior
public class RadioCalmingMusicBehavior implements
RadioMusicBehavior
public class RadioLightMusicBehavior implements
RadioMusicBehavior
```

Then the three of the radio's music behavior will be instantiated as different buttons on the Radio, referring to Appendix 5.4.6 and will call the different radio's music behavior when the button that is labelled as 1, 2 or 3 is clicked as shown in Figure 3.7.

```
private void
radioButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        radioDeco.clip.stop();
        radioDeco.setPlayMusic(new RadioDarkMusicBehavior());
        radioDeco.performPlayMusic();
    }
    private void
radioButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        radioDeco.clip.stop();
        radioDeco.setPlayMusic(new RadioLightMusicBehavior());
        radioDeco.performPlayMusic();
    }
    private void
radioButton3ActionPerformed(java.awt.event.ActionEvent evt) {
        radioDeco.clip.stop();
        radioDeco.setPlayMusic(new RadioCalmingMusicBehavior());
        radioDeco.performPlayMusic();
    }
```

Strategy design pattern was chosen to be implemented for changing radio’s music because it will be easier to add new music or remove existing music, thus the extensibility of the Radio object in terms of music options it offers is increased. With every addition using a strategy design pattern instead of using a big nested if/else to call the music file, cyclomatic complexity can be avoided.

2.5. Decorator Design Pattern

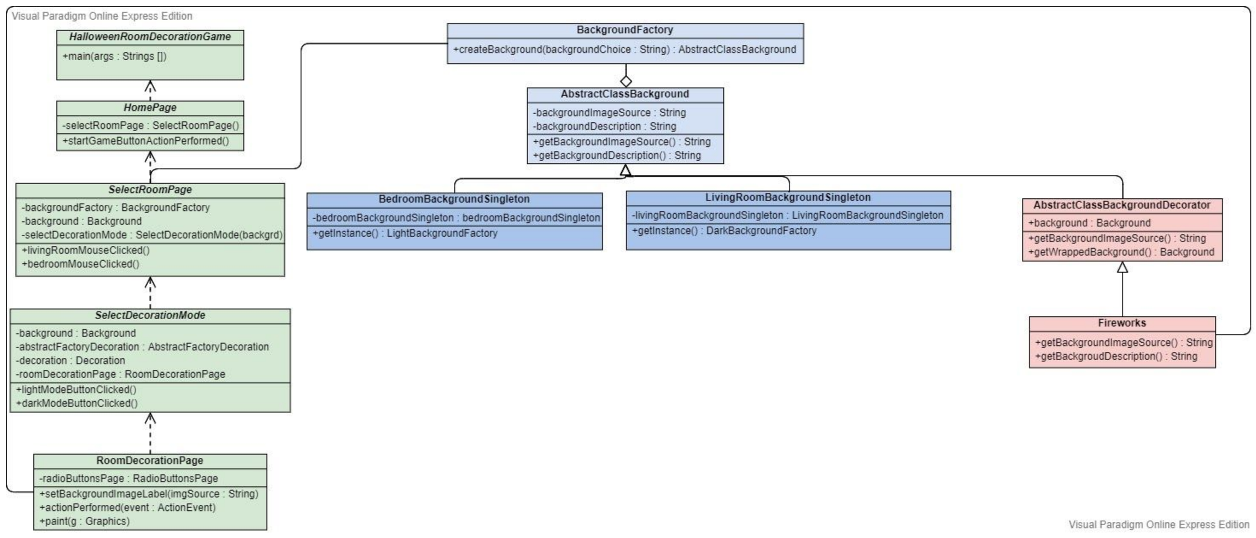

Visual Paradigm Online Express Edition
HalloweenRoomDecorationGame
+main(args : Strings [])
HomePage
-selectRoomPage : SelectRoomPage()
+startGameButtonActionPerformed()
SelectRoomPage
-backgroundFactory : BackgroundFactory
-background : Background
-selectDecorationMode : SelectDecorationMode(backgrd)
+livingRoomMouseClicked()
+bedroomMouseClicked()
SelectDecorationMode
-background : Background
-abstractFactoryDecoration : AbstractFactoryDecoration
-decoration : Decoration
-roomDecorationPage : RoomDecorationPage
+lightModeButtonClicked()
+darkModeButtonClicked()
RoomDecorationPage
-radioButtonsPage : RadioButtonsPage
+setBackgroundImageLabel(imgSource : String)
+actionPerformed(event : ActionEvent)
+paint(g : Graphics)
BackgroundFactory
+createBackground(backgroundChoice : String) : AbstractClassBackground
AbstractClassBackground
-backgroundImageSource : String
-backgroundDescription : String
+getBackgroundImageSource() : String
+getBackgroundDescription() : String
BedroomBackgroundSingleton
-bedroomBackgroundSingleton : bedroomBackgroundSingleton
+getInstance() : LightBackgroundFactory
LivingRoomBackgroundSingleton
-livingRoomBackgroundSingleton : LivingRoomBackgroundSingleton
+getInstance() : DarkBackgroundFactory
AbstractClassBackgroundDecorator
+background : Background
+getBackgroundImageSource() : String
+getWrappedBackground() : Background
Fireworks
+getBackgroundImageSource() : String
+getBackgroudDescription() : String
Visual Paradigm Online Express Edition

Figure 2.5.1 UML Class Diagram For Decorator Design Pattern

The pink classes in Figure 2.5.1 shows `AbstractClassBackgroundDecorator` and `Fireworks` that correspond to the Decorator design pattern. We use Decorator design pattern, a structural pattern here to wrap up the fireworks decoration. Decorator pattern allows us to add new decoration to our current background room without changing its structure. This pattern creates a decorator class where it will wrap the background class and provide additional fireworks decoration into the room, keeping the background class methods intact.

Appendix 5.5.2 shows that `AbstractClassBackgroundDecorator` will be extending `AbstractClassBackground` in Appendix 5.5.1 to inherit `getBackgroundImageSource()` where it will be overridden by `Fireworks` later on. `Fireworks` will inherit the `getBackgroundImageSource()` by extending from `AbstractClassBackgroundDecorator`. This concrete class is in charge of wrapping the fireworks onto the background image and returning the wrapped image. A counter will be passed into the parameter as the class is constructed in order to keep track with the placement of the fireworks each time it is called. We combine the firework image with the chosen room background together with the help from these library classes from Java:

```
import java.awt.AlphaComposite;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
```

The `Fireworks` class then returns the image source path to the caller method in `RoomDecorationPage` as shown in Appendix 5.5.4 below.

In `RoomDecorationPage`, a `fireworkAddButton JButton` is used to listen to any action when the user clicks the add button. When it is clicked, the counter, `fireworkCounterInt` will start to keep track of how many fireworks should be wrapped. For our project we set the limit to three, where a maximum of three fireworks can be added to the background room:

```
private static final int FIREWORKS_COUNTER = 3;
```

The concrete class of `Fireworks` will be instantiated here when the user clicked the add button. When the `getBackgroundImageSource()` is called from `Fireworks`, it will return the path name of the new image source that has been wrapped with a firework image. We will then set the current background to a new background with fireworks being wrapped together as one by calling the method:

```
setBackgroundImageLabel(imgSrc);
```

And if the user clicked more than the limit we set, we will show on the GUI that it is the max amount of fireworks. We can also unwrap all the fireworks when the user clicks on the `fireworkButton JButton`. It will call `getWrappedBackground()` where it corresponds with `fireworkCounterInt` to get back the returned original image of the background. This background is then passed to `setBackgroundImageLabel()` to be displayed in Figure 3.7 under Screen Captures of GUI of Application.

2.6. Facade Design Pattern

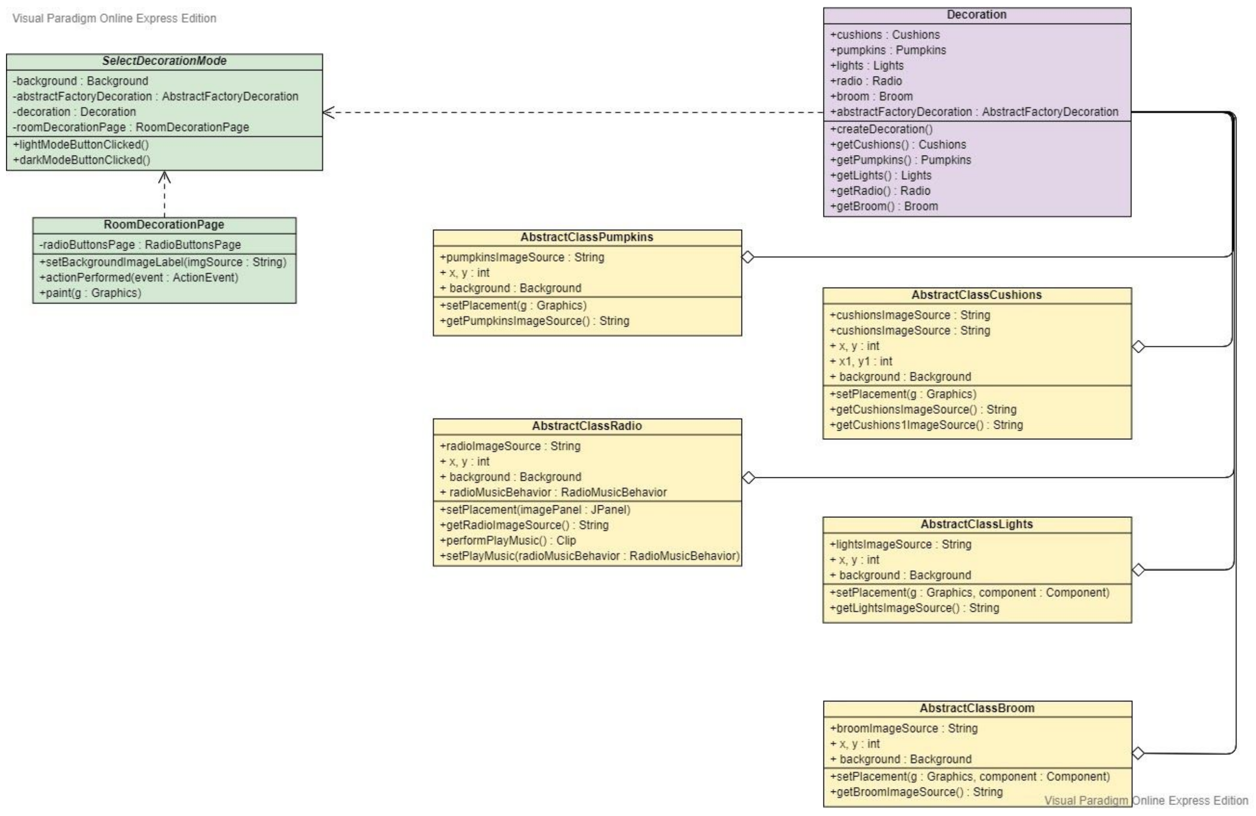

Visual Paradigm Online Express Edition
SelectDecorationMode
-background : Background
-abstractFactoryDecoration : AbstractFactoryDecoration
-decoration : Decoration
-roomDecorationPage : RoomDecorationPage
+lightModeButtonClicked()
+darkModeButtonClicked()
RoomDecorationPage
-radioButtonsPage : RadioButtonsPage
+setBackgroundImageLabel(imgSource : String)
+actionPerformed(event : ActionEvent)
+paint(g : Graphics)
Decoration
+cushions : Cushions
+pumpkins : Pumpkins
+lights : Lights
+radio : Radio
+broom : Broom
+abstractFactoryDecoration : AbstractFactoryDecoration
+createDecoration()
+getCushions() : Cushions
+getPumpkins() : Pumpkins
+getLights() : Lights
+getRadio() : Radio
+getBroom() : Broom
AbstractClassPumpkins
+pumpkinsImageSource : String
+ x, y : int
+ background : Background
+setPlacement(g : Graphics)
+getPumpkinsImageSource() : String
AbstractClassCushions
+cushionsImageSource : String
+cushionsImageSource : String
+ x, y : int
+ x1, y1 : int
+ background : Background
+setPlacement(g : Graphics)
+getCushionsImageSource() : String
+getCushions1ImageSource() : String
AbstractClassRadio
+radioImageSource : String
+ x, y : int
+ background : Background
+ radioMusicBehavior : RadioMusicBehavior
+setPlacement(imagePanel : JPanel)
+getRadioImageSource() : String
+performPlayMusic() : Clip
+setPlayMusic(radioMusicBehavior : RadioMusicBehavior)
AbstractClassLights
+lightsImageSource : String
+ x, y : int
+ background : Background
+setPlacement(g : Graphics, component : Component)
+getLightsImageSource() : String
AbstractClassBroom
+broomImageSource : String
+ x, y : int
+ background : Background
+setPlacement(g : Graphics, component : Component)
+getBroomImageSource() : String
Visual Paradigm Online Express Edition

Figure 2.6.1 UML Class Diagram For Facade Design Pattern

The classes shown above corresponds with Facade design pattern. We implemented Facade design pattern because it helps us hide the complexities of our system and provides an interface to the `SelectDecorationMode`, client of the system. This design pattern is categorised under structural pattern because this pattern adds an interface to an existing system to hide its complexities. It also avoids tight coupling between clients and subsystems. This pattern involves a single class, `DecorationFacade` which provides simplified methods required by our client, `SelectDecorationMode`.

The client class, `SelectDecorationMode` will be instantiating the Facade, `DecorationFacade` with `abstractFactoryDecoration` as the component. This class then pass the facade class as a component when instantiating `RoomDecorationPage`:

```
    decoration = new DecorationFacade(abstractFactoryDecoration);
roomDecorationPage = new RoomDecorationPage(background, decoration);
```

In `RoomDecorationPage`, the method `createDecoration()` is called to instantiate all the subsystem classes in `DecorationFacade`.

Appendix 5.6.3 shows how `DecorationFacade` class implements the Facade Pattern, we created a class that simplifies and unifies a set of more complex classes such as `cushions`, `pumpkins`, `lights`, `radio` and `broom`. In the `createDecoration()` method, we will instantiate each decoration using the abstract class, `AbstractFactoryDecoration`. The `DecorationFacade` class also has methods to return the decorations when being called by `RoomDecorationPage.`

2.7. Command Design Pattern

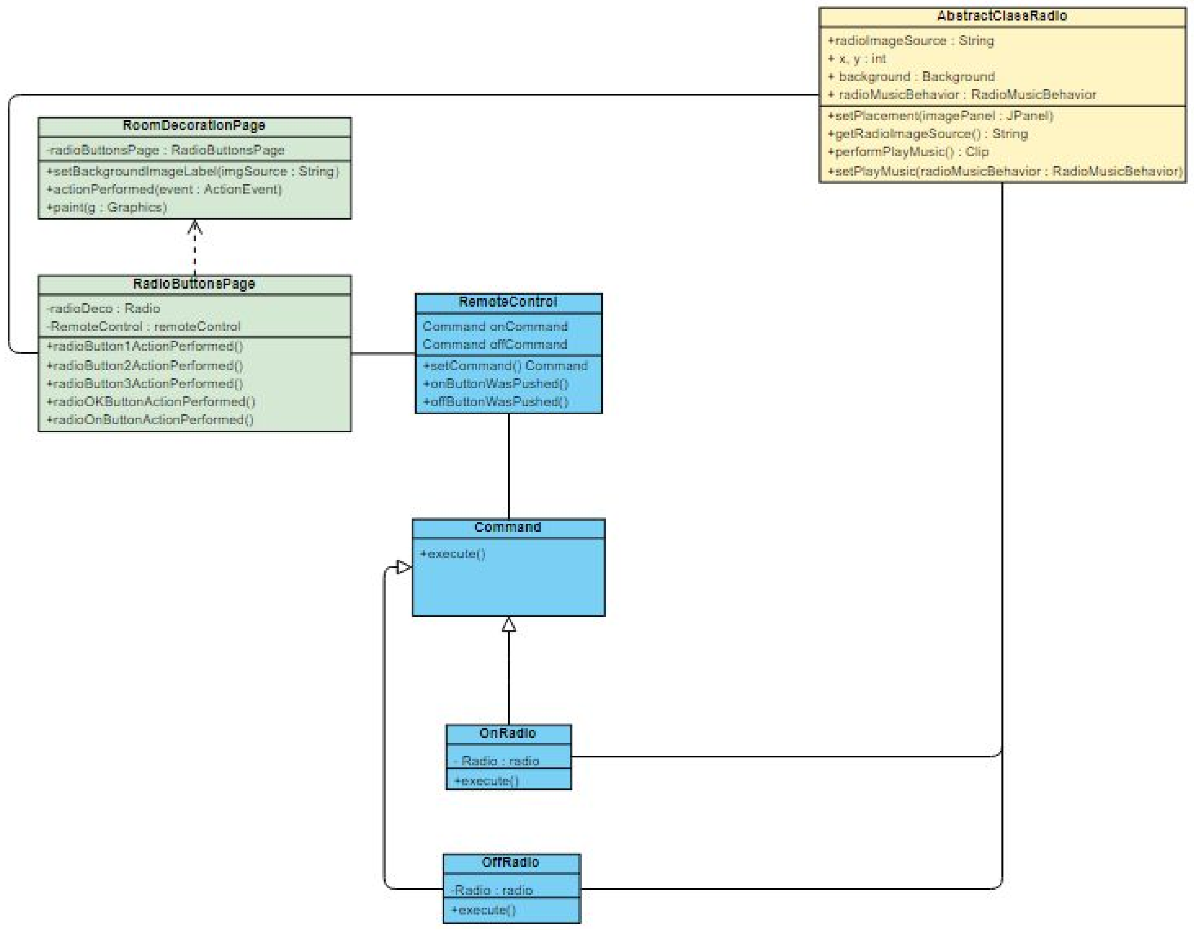

AbstractClassRadio
+radioImageSource : String
+ x, y : int
+ background : Background
+ radioMusicBehavior : RadioMusicBehavior
+setPlacement(imagePanel : JPanel)
+getRadioImageSource() : String
+performPlayMusic() : Clip
+setPlayMusic(radioMusicBehavior : RadioMusicBehavior)
RoomDecorationPage
-radioButtonsPage : RadioButtonsPage
+setBackgroundImageLabel(imgSource : String)
+actionPerformed(event : ActionEvent)
+paint(g : Graphics)
RadioButtonsPage
-radioDeco : Radio
-RemoteControl : remoteControl
+radioButton1ActionPerformed()
+radioButton2ActionPerformed()
+radioButton3ActionPerformed()
+radioOKButtonActionPerformed()
+radioOnButtonActionPerformed()
RemoteControl
Command onCommand
Command offCommand
+setCommand() Command
+onButtonWasPushed()
+offButtonWasPushed()
Command
+execute()
OnRadio
- Radio : radio
+execute()
OffRadio
-Radio : radio
+execute()

Figure 2.6.1: UML Class Diagram for Command Design Pattern

The figure shown above is the list of classes involved when implementing the Command design pattern. The classes involved is `RemoteControl`, `Command`, `OnRadio`, `OffRadio` and `AbstractClassRadio`, `RadioButtonsPage` and, `RoomDecorationPage`. We use the Command design pattern to make the radio able to be turned on and off. The Command design pattern is used in this feature because we have the choice of turning the radio on or off, which have the same concept as the undo and redo function. We just tweak the functionality and make it suitable for the radio feature. The Command design pattern decouples the classes that invoke the operations from the classes that know how to perform the operation. This design pattern helps to simplify the code, which is really helpful.

The `Command` is the interface class (Appendix 5.7.1), where it acts as the command for this design the function of this class is to execute the process. The method `execute()` is defined in the interface and it must be used by the concrete command class. The method is a medium for the `Receiver`. This class will be implemented by the concrete command class, the `OnRadio` and `OffRadio`.
`OnRadio` and `OffRadio` is the concrete command class (Appendix 5.7.2 & 5.7.3) that will implements the interface `Command`. We can all it as a sub-command, to ease the flow of the code and easily differentiate type of request. Concrete command can be seen as a request made by the programmer, that depends on the interface class for it to be executed. The class will override the `execute()` function in the `Command` interface to specify the request. Such that `OnRadio` will request for the code to play the music while `OffRadio` will request to stop the music.

`RemoteControl` (Appendix 5.7.4) is a type of invoker class, based on the Command design pattern. This class will identify the type of the concrete command which will execute which command, based on the request. However, it will not implement the code directly as it will rely on the interface `Command` class to perform the request, meaning that it is independent from how the request is performed. In this case, the `RemoteControl` will execute the command of `OnRadio` and `OffRadio` requested through the `Receiver`.

The class `Radio` (Appendix 5.7.5) is the receiver in this project. The receiver is where the set of commands is performed. This is the final destination where all the code input is going to be used. This class is loosely coupled from the invoker class `(RemoteControl)`, which means the invoker class does not have the knowledge of the receiver, but is still able to invoke a command for the receiver to execute. The `Radio` holds all the requests from the `OnRadio` and `OffRadio` of the object and knows how to perform it, which it will be called by the `Invoker` class.

The `RadioButtonsPage` (Appendix 5.7.6) is the client, where we will want to see the result of the process. Two buttons were made as there are ‘On’ and ‘Off’ button, which each button will execute a different command, because of their different requests. The `OnRadio` request is being put in the `OnRadioButtonActionPerformed()` where if pressed, it will play the music. The `OffRadio` request is being put in the `OffRadioButtonActionPerformed()` where if pressed, the radio will stop playing the music.

3. Screen Captures of GUI of Application

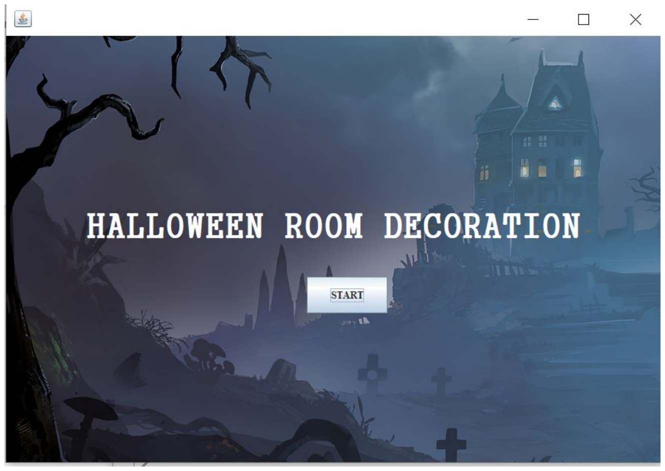


Figure 3.1 Home Page for Halloween Room Decoration Application

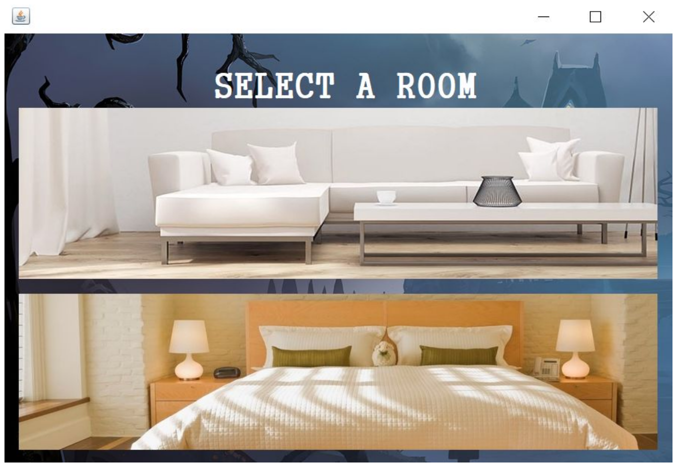


Figure 3.2 Select a Room Page

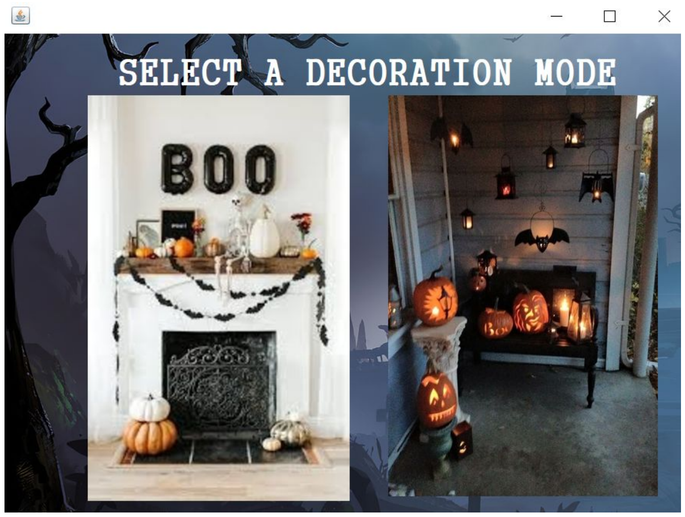


Figure 3.3 Select a Decoration Mode Page

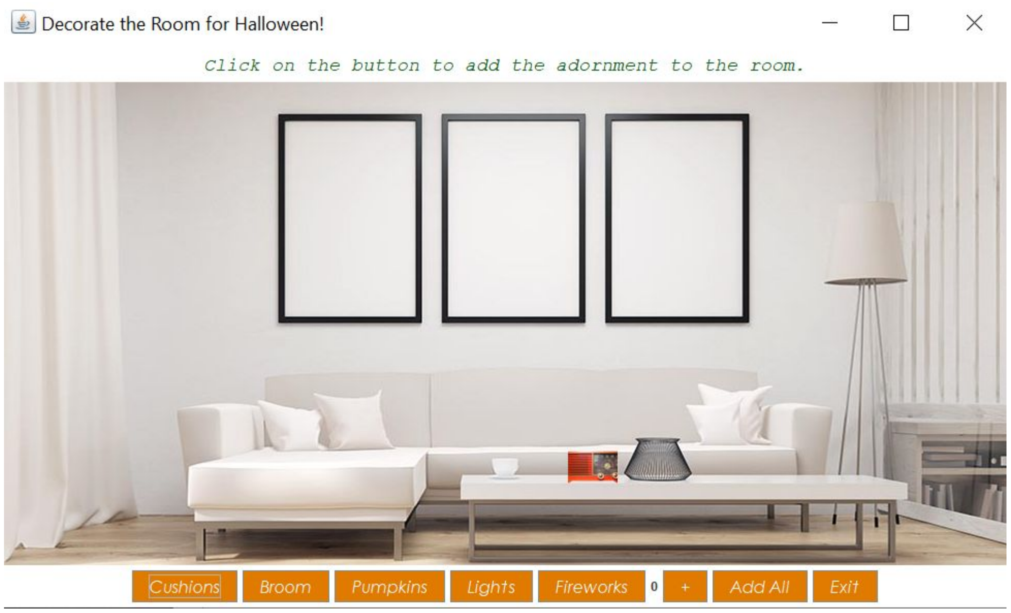

Decorate the Room for Halloween!
Click on the button to add the adornment to the room.
Cushions
Broom
Pumpkins
Lights
Fireworks
0
+
Add All
Exit

Figure 3.4 A Room With Living Room Background for Decorating

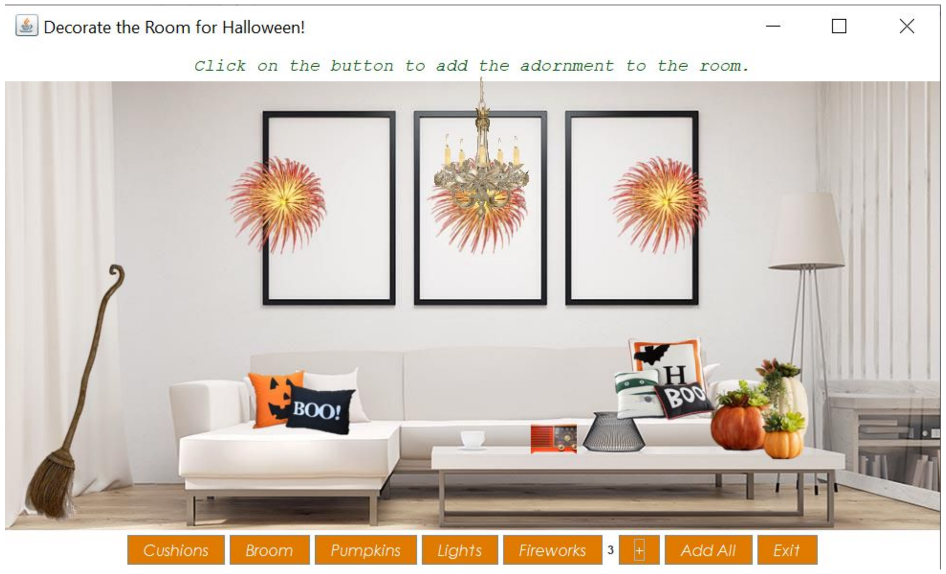

Decorate the Room for Halloween!
Click on the button to add the adornment to the room.
Cushions
Broom
Pumpkins
Lights
Fireworks
3
+
Add All
Exit

Figure 3.5 A Room With Living Room Background After Adding All Light-Themed Decorations and Fireworks

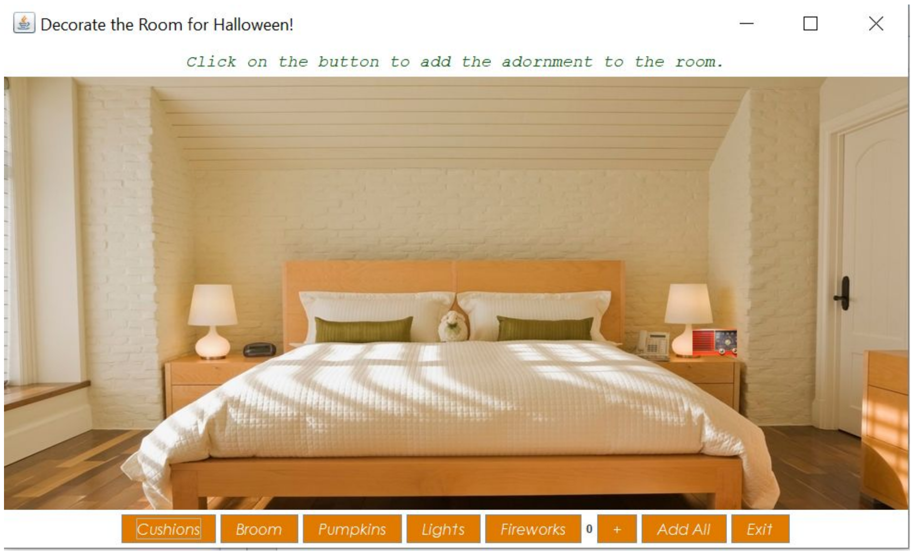

Decorate the Room for Halloween!
Click on the button to add the adornment to the room.
Cushions
Broom
Pumpkins
Lights
Fireworks
0
+
Add All
Exit

Figure 3.6 A Room With Bedroom Background for Decorating

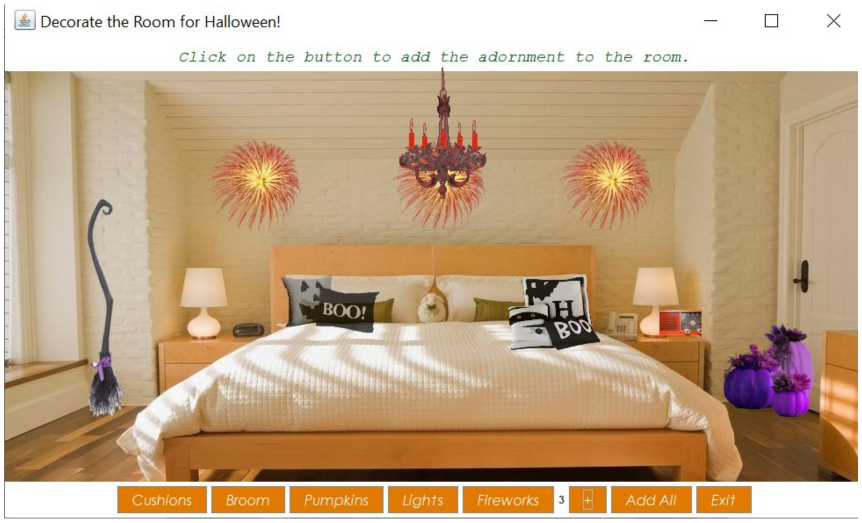

Decorate the Room for Halloween!
Click on the button to add the adornment to the room.
Cushions
Broom
Pumpkins
Lights
Fireworks
3
+
Add All
Exit

Figure 3.7 A Room With Bedroom Background After Adding All Dark-Themed Decorations and Fireworks

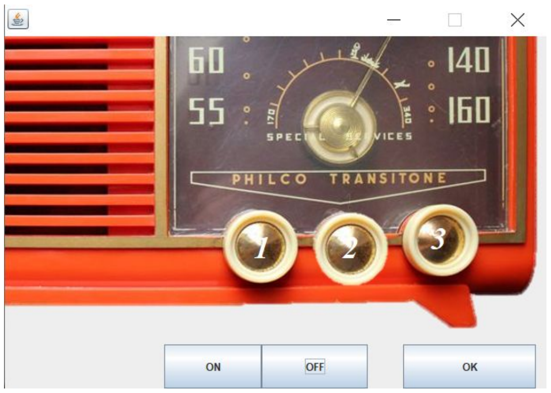


Figure 3.7 Radio Buttons Page After Radio is Clicked

4. Reference List

Freeman, E. (2016). *Head First Design Patterns: A brain-friendly guide* (10th ed.). O'reilly.

Gamma, E., Heml, R., Johnson, R. & Vlissides, J. (1995). *Design Patterns: Elements of reusable object-oriented software*. Addison-Wesley.

Shalloway, A., Trott, J. R. (2004). *Design Patterns Explained: A new perspective on object-oriented design (2nd ed.)*. Addison-Wesley

5. Appendix containing complete source code of the application

5.1. Simple Factory

5.1.1 BackgroundFactory Class

```
public class BackgroundFactory {

   public Background createBackground(String backgroundChoice)
   throws UnsupportedAudioFileException, IOException,
   LineUnavailableException{
           switch(backgroundChoice){
               case "livingroom":
                   return
   LivingRoomBackgroundSingleton.getInstance();
               case "bedroom":
                   return
   BedroomBackgroundSingleton.getInstance();
               default:
                   return null;
           }
       }

     }
```

5.1.2 Background Class

```
public abstract class Background {

    String backgroundImageSource;
    String backgroundDescription;

    public String getBackgroundImageSource(){
        return this.backgroundImageSource;
    }

    public abstract String getBackgroundDescription();

}
```

5.1.3 LivingRoomBackgroundSingleton Class

```
public class LivingRoomBackgroundSingleton extends Background{

   private static LivingRoomBackgroundSingleton
   livingRoomBackgroundSingleton;

   private LivingRoomBackgroundSingleton() throws IOException,
   LineUnavailableException, UnsupportedAudioFileException{
           this.backgroundImageSource =
   "src/s7assignment1halloweenroom4/media/sofalivingroon.jpg";
   //         this.backgroundDescription = "livingroom";
       }

   public static LivingRoomBackgroundSingleton getInstance()
   throws IOException, LineUnavailableException,
   UnsupportedAudioFileException{
           if(livingRoomBackgroundSingleton == null){
               livingRoomBackgroundSingleton = new
   LivingRoomBackgroundSingleton();
           }
           return livingRoomBackgroundSingleton;
       }

       @Override
       public String getBackgroundDescription() {
           return "livingroom";
       }

}
```

5.1.4 BedroomBackgroundSingleton Class

```
public class BedroomBackgroundSingleton extends Background{

  private static BedroomBackgroundSingleton
  bedroomBackgroundSingleton;

      private BedroomBackgroundSingleton() throws IOException,
  LineUnavailableException, UnsupportedAudioFileException{
          this.backgroundImageSource =
  "src/s7assignment1halloweenroom4/media/bedroom.jpg";
  //         this.backgroundDescription = "bedroom";
      }
```

```
	public static BedroomBackgroundSingleton getInstance()
  throws UnsupportedAudioFileException,
  UnsupportedAudioFileException, IOException,
  LineUnavailableException{
		if(bedroomBackgroundSingleton == null){
		    bedroomBackgroundSingleton = new
  BedroomBackgroundSingleton();
		}
		return bedroomBackgroundSingleton;
	}

	@Override
	public String getBackgroundDescription() {
	    return "bedroom";
	}

}
```

5.1.5 SelectRoomPage Class

```
public class SelectRoomPage extends javax.swing.JFrame {

    /**
     * Creates new form SelectRoomPage
     */

    private static BackgroundFactory backgroundFactory = new
BackgroundFactory();
    private static Background background;

    private static SelectDecorationMode selectDecorationMode;

    public SelectRoomPage() {
        initComponents();
        this.setLocationRelativeTo(null);
    }

    /**
     * This method is called from within the constructor to
initialize the form.
     * WARNING: Do NOT modify this code. The content of this
method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated
Code">
    private void initComponents() {

        jLabel2 = new javax.swing.JLabel();
        livingRoomButton = new javax.swing.JLabel();
        bedroomButton = new javax.swing.JLabel();
        jLabel1 = new javax.swing.JLabel();


setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CL
OSE);
        setPreferredSize(new java.awt.Dimension(915, 640));
        getContentPane().setLayout(null);

        jLabel2.setFont(new java.awt.Font("GungsuhChe", 1,
48)); // NOI18N
        jLabel2.setForeground(new java.awt.Color(255, 255,
255));
        jLabel2.setText("SELECT A ROOM");
        getContentPane().add(jLabel2);
        jLabel2.setBounds(280, 30, 360, 80);

        livingRoomButton.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/s7assignment1hal
loweenroom4/media/sofalivingroomHomePage.jpg"))); // NOI18N
        livingRoomButton.setText("jLabel4");
        livingRoomButton.addMouseListener(new
java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent
evt) {
                livingRoomButtonMouseClicked(evt);
            }
        });
        getContentPane().add(livingRoomButton);
        livingRoomButton.setBounds(20, 100, 850, 230);

        bedroomButton.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/s7assignment1hal
loweenroom4/media/bedroomHomePage.jpg"))); // NOI18N
        bedroomButton.setText("jLabel1");
        bedroomButton.addMouseListener(new
java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent
evt) {
```

```
                bedroomButtonMouseClicked(evt);
            }
        });
        getContentPane().add(bedroomButton);
        bedroomButton.setBounds(20, 350, 850, 210);

        jLabel1.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/s7assignment1hal
loweenroom4/media/halloweenHomePage.jpg"))); // NOI18N
        jLabel1.setText("jLabel1");
        getContentPane().add(jLabel1);
        jLabel1.setBounds(0, 0, 900, 590);

        pack();
    }// </editor-fold>

    private void
livingRoomButtonMouseClicked(java.awt.event.MouseEvent evt) {
        // TODO add your handling code here:
        try {
            background =
backgroundFactory.createBackground("livingroom");
        } catch (UnsupportedAudioFileException ex) {

Logger.getLogger(SelectRoomPage.class.getName()).log(Level.SEVE
RE, null, ex);
        } catch (IOException ex) {

Logger.getLogger(SelectRoomPage.class.getName()).log(Level.SEVE
RE, null, ex);
        } catch (LineUnavailableException ex) {

Logger.getLogger(SelectRoomPage.class.getName()).log(Level.SEVE
RE, null, ex);
        }
        this.setVisible(false);
        selectDecorationMode = new
SelectDecorationMode(background);
        selectDecorationMode.setVisible(true);
    }

    private void
bedroomButtonMouseClicked(java.awt.event.MouseEvent evt) {
        // TODO add your handling code here:
        try {
            background =
backgroundFactory.createBackground("bedroom");
        } catch (UnsupportedAudioFileException ex) {

Logger.getLogger(SelectRoomPage.class.getName()).log(Level.SEVE
RE, null, ex);
        } catch (IOException ex) {

Logger.getLogger(SelectRoomPage.class.getName()).log(Level.SEVE
RE, null, ex);
        } catch (LineUnavailableException ex) {

Logger.getLogger(SelectRoomPage.class.getName()).log(Level.SEVE
RE, null, ex);
        }
        this.setVisible(false);
        selectDecorationMode = new
SelectDecorationMode(background);
        selectDecorationMode.setVisible(true);
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and
feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not
available, stay with the default look and feel.
         * For details see
http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/
plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {
```

```
java.util.logging.Logger.getLogger(SelectRoomPage.class.getName
()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(SelectRoomPage.class.getName
()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(SelectRoomPage.class.getName
()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException
ex) {

java.util.logging.Logger.getLogger(SelectRoomPage.class.getName
()).log(java.util.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new SelectRoomPage().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JLabel bedroomButton;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel livingRoomButton;
    // End of variables declaration
}
```

5.2. Abstract Factory Design Pattern

5.2.1 AbstractFactoryDecoration

```
public abstract class AbstractFactoryDecoration {
    Background background;

    public AbstractFactoryDecoration(Background background)
throws IOException, LineUnavailableException,
UnsupportedAudioFileException{
        this.background = background;
    }

    public abstract Cushions createCushions();
    public abstract Pumpkins createPumpkins();
    public abstract Lights createLights();
    public abstract Radio createRadio();
    public abstract Broom createBroom();
}
```

5.2.2 AbstractFactoryDarkDecoration

```
public class AbstractFactoryDarkDecoration extends
AbstractFactoryDecoration{

    public AbstractFactoryDarkDecoration(Background background)
throws IOException, LineUnavailableException,
UnsupportedAudioFileException {
        super(background);
    }

    @Override
    public Cushions createCushions() {
        return new DarkCushions(background);
    }
    @Override
    public Pumpkins createPumpkins() {
        return new DarkPumpkins(background);
    }
    @Override
    public Lights createLights() {
        return new DarkLights(background);
    }
    @Override
    public Radio createRadio() {
        return new DarkRadio(background);
    }
    @Override
    public Broom createBroom() {
        return new DarkBroom(background);
    }

}
```

5.2.3 AbstractFactoryLightDecoration

```
public class AbstractFactoryLightDecoration extends
AbstractFactoryDecoration{

    public AbstractFactoryLightDecoration(Background
background) throws IOException, LineUnavailableException,
UnsupportedAudioFileException {
        super(background);
    }

    @Override
    public Cushions createCushions() {
        return new LightCushions(background);
    }
    @Override
    public Pumpkins createPumpkins() {
        return new LightPumpkins(background);
    }
    @Override
    public Lights createLights() {
        return new LightLights(background);
    }
    @Override
    public Radio createRadio() {
        return new LightRadio(background);
    }
    @Override
    public Broom createBroom() {
        return new LightBroom(background);
    }

}
```

5.2.4 Broom

```
public abstract class Broom {
    String broomImageSource;
    Background background;
    int x, y;
    public Broom(Background background){
    this.background = background;
    }
    public String getBroomImageSource(){
        return this.broomImageSource;
    }
    public void setPlacement(Graphics g){

        if(background.getBackgroundDescription().endsWith("livi
        ngroom")){
            this.x = 30;
            this.y = 270;
        }
        else if
        (background.getBackgroundDescription().endsWith("bedroo
        m")){
            this.x = 30;
            this.y = 220;
        }
        try {
            Image broomImage = ImageIO.read(new
            File(broomImageSource));
            g.drawImage(broomImage, x, y, null);
        } catch (IOException ex) {

            Logger.getLogger(Pumpkins.class.getName()).log(Leve
            l.SEVERE, null, ex);
        }
    }
}
```

5.2.5 DarkBroom

```
public class DarkBroom extends Broom{
    public DarkBroom(Background background) {
        super(background);
        this.broomImageSource =
        "src/s7assignment1halloweenroom4/media/darkBroom.png";
    }
}
```

5.2.6 LightBroom

```
public class LightBroom extends Broom{
    public LightBroom(Background background) {
        super(background);
        this.broomImageSource =
        "src/s7assignment1halloweenroom4/media/lightBroom.png";
    }
}
```

5.2.7 Cushions

```
public abstract class Cushions {
    String cushionsImageSource;
    String cushions1ImageSource;
    int x, y;
    int x1, y1;
    Background background;

    public Cushions(Background background){
        this.background = background;
    }
    public String getCushionsImageSource(){
        return this.cushionsImageSource;
    }
    public String getCushions1ImageSource(){
        return this.cushions1ImageSource;
    }

    public void setPlacement(Graphics g){

        if(background.getBackgroundDescription().endsWith("livi
        ngroom")){
            this.x = 260;
            this.y = 370;
            this.x1 = 645;
            this.y1 = 340;
         }else if
        (background.getBackgroundDescription().endsWith("bedroo
        m")){
            this.x = 320;
            this.y = 300;
            this.x1 = 580;
            this.y1 = 305;
        }
        try {
            Image cushionsImage = ImageIO.read(new
            File(cushionsImageSource));
            g.drawImage(cushionsImage, x, y, null);
            Image cushions1Image = ImageIO.read(new
            File(cushions1ImageSource));
            g.drawImage(cushions1Image, x1, y1, null);
        } catch (IOException ex) {

            Logger.getLogger(Cushions.class.getName()).log(Leve
            l.SEVERE, null, ex);
        }
    }
}
```

5.2.8 DarkCushions

```
public class DarkCushions extends Cushions{
    public DarkCushions(Background background) {
        super(background);
        cushionsImageSource =
        "src/s7assignment1halloweenroom4/media/darkCushions.png
        ";
        cushions1ImageSource =
        "src/s7assignment1halloweenroom4/media/darkCushions1.pn
        g";
    }
}
```

5.2.9 LightCushions

```
public class LightCushions extends Cushions{
    public LightCushions(Background background) {
        super(background);
        cushionsImageSource =
        "src/s7assignment1halloweenroom4/media/lightCushions.pn
        g";
        cushions1ImageSource =
        "src/s7assignment1halloweenroom4/media/lightCushions1.p
        ng";
    }
}
```

5.2.10 Lights

```
public class Lights {
    String lightsImageSource;
    Background background;
    int x, y;

    public Lights(Background background){
        this.background = background;
    }
    public String getLightsImageSource(){
        return this.lightsImageSource;
    }

    public void setPlacement(Graphics g, Component component)
    throws IOException{

        if(background.getBackgroundDescription().endsWith("livi
        ngroom")){
            this.x = 460;
            this.y = 70;
        }
        else if
        (background.getBackgroundDescription().endsWith("bedroo
        m")){
            this.x = 460;
            this.y = 70;
        }
        ImageIcon lightsIcon = new
        ImageIcon(getClass().getResource(lightsImageSource));
        lightsIcon.paintIcon(component, g, x, y);
    }
}
```

5.2.11 DarkLights

```
public class DarkLights extends Lights{
    public DarkLights(Background background) {
        super(background);
        lightsImageSource = "media/darkLights.gif";
    }
}
```

5.2.12 LightLights

```
public class LightLights extends Lights{
    public LightLights(Background background) {
        super(background);
        lightsImageSource = "media/lightLights.gif";
    }
}
```

5.2.13 Pumpkins

```
public abstract class Pumpkins {
    String pumpkinsImageSource;
    Background background;
    int x, y;

    public Pumpkins(Background background){
        this.background = background;
    }
    public String getPumpkinsImageSource(){
        return this.pumpkinsImageSource;
    }

    public void setPlacement(Graphics g){

        if(background.getBackgroundDescription().endsWith("livi
        ngroom")){
            this.x = 730;
            this.y = 355;
        }
        else if
        (background.getBackgroundDescription().endsWith("bedroo
        m")){
            this.x = 810;
            this.y = 355;
        }
        try {
            Image pumpkinsImage = ImageIO.read(new
            File(pumpkinsImageSource));
            g.drawImage(pumpkinsImage, x, y, null);
        } catch (IOException ex) {

            Logger.getLogger(Pumpkins.class.getName()).log(Leve
            l.SEVERE, null, ex);
        }
    }
  }
```

5.2.14 DarkPumpkins

```
public class DarkPumpkins extends Pumpkins{
    public DarkPumpkins(Background background) {
        super(background);
        this.pumpkinsImageSource =
        "src/s7assignment1halloweenroom4/media/darkPumpkins.png
        ";
    }
}
```

5.2.15 LightPumpkins

```
public class LightPumpkins extends Pumpkins{
    public LightPumpkins(Background background) {
        super(background);
        this.pumpkinsImageSource =
        "src/s7assignment1halloweenroom4/media/lightPumpkins.pn
        g";
    }
}
```

5.2.16 lightModeButtonMouseClicked in SelectDecorationMode

```
    private void
lightModeButtonMouseClicked(java.awt.event.MouseEvent evt) {
        try {
            // TODO add your handling code here:
            abstractFactoryDecoration = new
            AbstractFactoryLightDecoration(background);
            decoration = new
            Decoration(abstractFactoryDecoration);
            roomDecorationPage = new
            RoomDecorationPage(background, decoration);
        } catch (IOException ex) {

            Logger.getLogger(SelectDecorationMode.class.getName
            ()).log(Level.SEVERE, null, ex);
        } catch (LineUnavailableException ex) {

            Logger.getLogger(SelectDecorationMode.class.getName
            ()).log(Level.SEVERE, null, ex);
        } catch (UnsupportedAudioFileException ex) {

            Logger.getLogger(SelectDecorationMode.class.getName
            ()).log(Level.SEVERE, null, ex);
        }
        this.setVisible(false);}
```

5.2.17 darkModeButtonMouseClicked in SelectDecorationMode

```
    private void
darkModeButtonMouseClicked(java.awt.event.MouseEvent evt) {
        try {
            // TODO add your handling code here:
            abstractFactoryDecoration = new
            AbstractFactoryDarkDecoration(background);
            decoration = new
            Decoration(abstractFactoryDecoration);
            roomDecorationPage = new
            RoomDecorationPage(background, decoration);
        } catch (IOException ex) {

           Logger.getLogger(SelectDecorationMode.class.getName(
           )).log(Level.SEVERE, null, ex);
        } catch (LineUnavailableException ex) {

           Logger.getLogger(SelectDecorationMode.class.getName(
           )).log(Level.SEVERE, null, ex);
        } catch (UnsupportedAudioFileException ex) {

           Logger.getLogger(SelectDecorationMode.class.getName(
           )).log(Level.SEVERE, null, ex);
        }
        this.setVisible(false);
    }
```

5.3. Strategy Design Pattern

5.3.1. RadioCalmingMusicBehavior

```
public class RadioCalmingMusicBehavior implements
RadioMusicBehavior{

    @Override
    public Clip playMusic() {
        //music
        File musicPath = new
File("src/s7assignment1halloweenroom4/media/calmingBg
m.wav");
        AudioInputStream audioInput;
        Clip clip = null;
        if (musicPath.exists()) {
            try {
                audioInput =
AudioSystem.getAudioInputStream(musicPath);
                clip = AudioSystem.getClip();
                clip.open(audioInput);
                clip.start();
            } catch (UnsupportedAudioFileException
ex) {

Logger.getLogger(RadioDarkMusicBehavior.class.getName
()).log(Level.SEVERE, null, ex);
            } catch (IOException ex) {

Logger.getLogger(RadioDarkMusicBehavior.class.getName
()).log(Level.SEVERE, null, ex);
            } catch (LineUnavailableException ex) {

Logger.getLogger(RadioDarkMusicBehavior.class.getName
()).log(Level.SEVERE, null, ex);
            }
        } else {
            System.out.println("oh no");
        }
        return clip;
    }

}
```

5.3.2. RadioDarkMusicBehavior

```
public class RadioDarkMusicBehavior implements
RadioMusicBehavior{

    @Override
    public Clip playMusic() {
        //music
        File musicPath = new
File("src/s7assignment1halloweenroom4/media/darkBgm.w
av");
        AudioInputStream audioInput;
        Clip clip = null;
        if (musicPath.exists()) {
            try {
                audioInput =
AudioSystem.getAudioInputStream(musicPath);
                clip = AudioSystem.getClip();
                clip.open(audioInput);
                clip.start();
            } catch (UnsupportedAudioFileException
ex) {

Logger.getLogger(RadioDarkMusicBehavior.class.getName
()).log(Level.SEVERE, null, ex);
            } catch (IOException ex) {

Logger.getLogger(RadioDarkMusicBehavior.class.getName
()).log(Level.SEVERE, null, ex);
            } catch (LineUnavailableException ex) {

Logger.getLogger(RadioDarkMusicBehavior.class.getName
()).log(Level.SEVERE, null, ex);
            }
        } else {
            System.out.println("oh no");
        }
        return clip;
    }

}
```

5.3.3. RadioLightMusicBehavior

```
public class RadioLightMusicBehavior implements
RadioMusicBehavior{

    @Override
    public Clip playMusic() {
        //music
        File musicPath = new
File("src/s7assignment1halloweenroom4/media/lightBgm.
wav");
        AudioInputStream audioInput;
        Clip clip = null;
        if (musicPath.exists()) {
            try {
                audioInput =
AudioSystem.getAudioInputStream(musicPath);
                clip = AudioSystem.getClip();
                clip.open(audioInput);
                clip.start();
            } catch (UnsupportedAudioFileException
ex) {

Logger.getLogger(RadioDarkMusicBehavior.class.getName
()).log(Level.SEVERE, null, ex);
            } catch (IOException ex) {

Logger.getLogger(RadioDarkMusicBehavior.class.getName
()).log(Level.SEVERE, null, ex);
            } catch (LineUnavailableException ex) {

Logger.getLogger(RadioDarkMusicBehavior.class.getName
()).log(Level.SEVERE, null, ex);
            }
        } else {
            System.out.println("oh no");
        }
        return clip;
    }

}
```

5.3.4. RadioMusicBehavior

```
public interface RadioMusicBehavior {

    public Clip playMusic();

}
```

5.3.5. Radio

```
public abstract class Radio {

    String radioImageSource;
    Background background;
    int x, y;
    Command command1;

    RadioMusicBehavior radioMusicBehavior;
    Clip clip;

    public Radio(Background background){
        this.background = background;
        this.radioImageSource =
"media/lightRadio.png";
    }

    public String getRadioImageSource(){
        return this.radioImageSource;
    }

    public JLabel setPlacement(JLabel imageLabel)
throws IOException, LineUnavailableException,
UnsupportedAudioFileException{

if(background.getBackgroundDescription().endsWith("li
vingroom")){
            this.x = 550;
            this.y = 350;
        }
        else
if(background.getBackgroundDescription().endsWith("be
droom")){
```

```
            this.x = 745;
            this.y = 265;
        }

        ImageIcon radioIcon;
        radioIcon = new
ImageIcon(getClass().getResource(this.radioImageSourc
e));
        JLabel lb = new JLabel(radioIcon);
        lb.setBounds(this.x, this.y,
lb.getPreferredSize().height,
lb.getPreferredSize().height);
        imageLabel.add(lb);
        return lb;
    }

    public Clip performPlayMusic(){
        this.clip =
this.radioMusicBehavior.playMusic();
        return this.clip;
    }

    public void setPlayMusic(RadioMusicBehavior
radioMusicBehavior){
        this.radioMusicBehavior = radioMusicBehavior;
    }

    public void on(){
        performPlayMusic();
    }

    public void off(){
        clip.stop();
    }

    public void setCommand(Command command){
        this.command1=command;
    }

}
```

5.3.6. RadioButtonsPage

```
public class RadioButtonsPage extends
javax.swing.JFrame {

    /**
     * Creates new form RadioButtonsPage
     */
    private static Radio radioDeco;
    RemoteControl remoteControl = new
RemoteControl();

    public RadioButtonsPage(Radio radioDeco) {
        initComponents();
        radioButton1.setOpaque(false);
        radioButton1.setContentAreaFilled(false);
        radioButton1.setBorderPainted(false);

        radioButton2.setOpaque(false);
        radioButton2.setContentAreaFilled(false);
        radioButton2.setBorderPainted(false);

        radioButton3.setOpaque(false);
        radioButton3.setContentAreaFilled(false);
        radioButton3.setBorderPainted(false);

        this.radioDeco = radioDeco;
        this.setResizable(false);
        this.setLocationRelativeTo(null);
    }

    /**
     * This method is called from within the
constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content
of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed"
desc="Generated Code">
    private void initComponents() {

        radioButton1 = new javax.swing.JButton();
        radioButton2 = new javax.swing.JButton();
        radioButton3 = new javax.swing.JButton();
        jLabel1 = new javax.swing.JLabel();
        radioOKButton = new javax.swing.JButton();
        offRadioButton = new javax.swing.JButton();
        onRadioButton = new javax.swing.JButton();


setDefaultCloseOperation(javax.swing.WindowConstants.
EXIT_ON_CLOSE);
        setPreferredSize(new java.awt.Dimension(620,
440));
        getContentPane().setLayout(null);

        radioButton1.setFont(new java.awt.Font("Times
New Roman", 3, 36)); // NOI18N
        radioButton1.setForeground(new
java.awt.Color(255, 255, 255));
        radioButton1.setText("1");
        radioButton1.addActionListener(new
java.awt.event.ActionListener() {
            public void
actionPerformed(java.awt.event.ActionEvent evt) {
                radioButton1ActionPerformed(evt);
            }
        });
        getContentPane().add(radioButton1);
        radioButton1.setBounds(250, 210, 80, 60);

        radioButton2.setFont(new java.awt.Font("Times
New Roman", 3, 36)); // NOI18N
        radioButton2.setForeground(new
java.awt.Color(255, 255, 255));
        radioButton2.setText("2");
        radioButton2.setActionCommand("");
        radioButton2.addActionListener(new
java.awt.event.ActionListener() {
            public void
actionPerformed(java.awt.event.ActionEvent evt) {
                radioButton2ActionPerformed(evt);
            }
        });
        getContentPane().add(radioButton2);
        radioButton2.setBounds(350, 210, 80, 60);
        radioButton3.setFont(new java.awt.Font("Times
New Roman", 3, 36)); // NOI18N
        radioButton3.setForeground(new
java.awt.Color(255, 255, 255));
        radioButton3.setText("3");
        radioButton3.addActionListener(new
java.awt.event.ActionListener() {
            public void
actionPerformed(java.awt.event.ActionEvent evt) {
                radioButton3ActionPerformed(evt);
            }
        });
        getContentPane().add(radioButton3);
        radioButton3.setBounds(450, 200, 80, 60);

        jLabel1.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/s7assi
gnment1halloweenroom4/media/radioButtons.png"))); //
NOI18N
        jLabel1.setText("jLabel1");
        getContentPane().add(jLabel1);
        jLabel1.setBounds(-270, -160, 990, 540);

        radioOKButton.setText("OK");
        radioOKButton.addActionListener(new
java.awt.event.ActionListener() {
            public void
actionPerformed(java.awt.event.ActionEvent evt) {
                radioOKButtonActionPerformed(evt);
            }
        });
        getContentPane().add(radioOKButton);
        radioOKButton.setBounds(450, 350, 150, 50);

        offRadioButton.setText("OFF");
        offRadioButton.addActionListener(new
java.awt.event.ActionListener() {
            public void
actionPerformed(java.awt.event.ActionEvent evt) {
                offRadioButtonActionPerformed(evt);
            }
        });
        getContentPane().add(offRadioButton);
        offRadioButton.setBounds(290, 350, 120, 50);
        onRadioButton.setText("ON");
        onRadioButton.addActionListener(new
java.awt.event.ActionListener() {
            public void
actionPerformed(java.awt.event.ActionEvent evt) {
                onRadioButtonActionPerformed(evt);
            }
        });
        getContentPane().add(onRadioButton);
        onRadioButton.setBounds(180, 350, 110, 50);

        pack();
    }// </editor-fold>

    private void
radioButton1ActionPerformed(java.awt.event.ActionEven
t evt) {
        // TODO add your handling code here:
        radioDeco.clip.stop();
        radioDeco.setPlayMusic(new
RadioDarkMusicBehavior());
        radioDeco.performPlayMusic();
    }

    private void
radioButton2ActionPerformed(java.awt.event.ActionEven
t evt) {
        // TODO add your handling code here:
        radioDeco.clip.stop();
        radioDeco.setPlayMusic(new
RadioLightMusicBehavior());
        radioDeco.performPlayMusic();
    }

    private void
radioButton3ActionPerformed(java.awt.event.ActionEven
t evt) {
        // TODO add your handling code here:
        radioDeco.clip.stop();
        radioDeco.setPlayMusic(new
RadioCalmingMusicBehavior());
        radioDeco.performPlayMusic();
    }
    private void
radioOKButtonActionPerformed(java.awt.event.ActionEve
nt evt) {
        // TODO add your handling code here:
        this.setVisible(false);
    }

    private void
onRadioButtonActionPerformed(java.awt.event.ActionEve
nt evt) {
        // TODO add your handling code here:

        OnRadio onRadio = new OnRadio(radioDeco);
        OffRadio offRadio = new OffRadio(radioDeco);
        remoteControl.setCommand(onRadio, offRadio);
        remoteControl.onButtonWasPushed();
//        radioDeco.setCommand(new OnRadio(new
Radio(radioDeco.clip)));
//        radioDeco.command1.execute();
//        this.setVisible(false);
    }

    private void
offRadioButtonActionPerformed(java.awt.event.ActionEv
ent evt) {
        // TODO add your handling code here:
//        RemoteControl remoteControl = new
RemoteControl();
        OnRadio onRadio = new OnRadio(radioDeco);
        OffRadio offRadio = new OffRadio(radioDeco);
        remoteControl.setCommand(onRadio, offRadio);
        remoteControl.offButtonWasPushed();
//        this.setVisible(false);
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed"
desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not
available, stay with the default look and feel.
```

```
         * For details see
http://download.oracle.com/javase/tutorial/uiswing/lo
okandfeel/plaf.html
         */
        try {
            for
(javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName()))
{

javax.swing.UIManager.setLookAndFeel(info.getClassNam
e());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(RadioButtonsPage.c
lass.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(RadioButtonsPage.c
lass.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(RadioButtonsPage.c
lass.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        } catch
(javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(RadioButtonsPage.c
lass.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        }
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new
Runnable() {
            public void run() {
                new
RadioButtonsPage(radioDeco).setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JLabel jLabel1;
    private javax.swing.JButton offRadioButton;
    private javax.swing.JButton onRadioButton;
    private javax.swing.JButton radioButton1;
    private javax.swing.JButton radioButton2;
    private javax.swing.JButton radioButton3;
    private javax.swing.JButton radioOKButton;
```

5.3.7. LightRadio

```
            public LightRadio(Background background) {
        super(background);
        RadioMusicBehavior lightMusic = new
RadioLightMusicBehavior();
        setPlayMusic(lightMusic);
    }
```

5.3.8. DarkRadio

```
            public DarkRadio(Background background) {
        super(background);
        RadioMusicBehavior darkMusic = new RadioDarkMusicBehavior();
        setPlayMusic(darkMusic);
    }
```

5.5. Decorator Design Pattern

5.5.1 Abstract class Background

```
public abstract class Background {

    String backgroundImageSource;
    String backgroundDescription;

    public String getBackgroundImageSource(){
        return this.backgroundImageSource;
    }

    public abstract String getBackgroundDescription();

}
```

5.5.2 Abstract Class BackgroundDecorator

```
public abstract class BackgroundDecorator extends
Background{

    Background background;

    @Override
    public abstract String getBackgroundImageSource();

    public Background getWrappedBackground(){
        return this.background;
    }

}
```

5.5.3 Class Fireworks

```
public class Fireworks extends BackgroundDecorator{

    int fireworkCounterInt;
    public Fireworks(Background background, int
fireworkCounterInt){
        this.background = background;
        this.fireworkCounterInt = fireworkCounterInt;
    }

    @Override
    public String getBackgroundImageSource() {
        String imageSource = "";
        Random random = new Random();
        try {
            String bgSource =
this.background.getBackgroundImageSource();
            BufferedImage im = ImageIO.read(new
File(bgSource));
            System.out.println("im: " + bgSource);
            BufferedImage im2 = ImageIO.read(new File(

"src/s7assignment1halloweenroom4/media/fireworks.png"));
            //int randomInt = random.nextInt(900);
            int positionWidth = 0;
            switch (fireworkCounterInt){
                case 1:
                    positionWidth =
(im.getWidth()-im2.getWidth())/2;
                    break;
                case 2:
                    positionWidth = 200;
                    break;
                case 3:
                    positionWidth = 600;
                    break;
            }

            Graphics2D g = im.createGraphics();

g.setComposite(AlphaComposite.getInstance(AlphaComposite.S
RC_OVER, 1.0f));
//            g.drawImage(im2,
(im.getWidth()-im2.getWidth())/2,
(im.getHeight()-im2.getHeight())/6, null);
            g.drawImage(im2, positionWidth,
(im.getHeight()-im2.getHeight())/6, null);

            g.dispose();
            imageSource = "decoratedBackground.jpg";
            ImageIO.write(im, "jpg", new
File(imageSource));
            return imageSource;
        } catch (IOException ex) {

Logger.getLogger(Fireworks.class.getName()).log(Level.SEVE
RE, null, ex);
        }
        return imageSource;
    }


    @Override
    public String getBackgroundDescription() {
        Random random = new Random();
        int randomInt = random.nextInt(100);
        return this.background.getBackgroundDescription()
+ " " + randomInt;
    }

}
```

5.5.4 Class RoomDecorationPage

```
else if (event.getSource() == fireworkButton) {
            if(fireworkCounterInt > 0){
                fireworksRevert = true;
                fireworkCounter.setText("0");
                BackgroundDecorator backgroundDecorator =
(BackgroundDecorator) background;
                for(int i = 1; i < fireworkCounterInt; i
++){
                    backgroundDecorator =
(BackgroundDecorator)
backgroundDecorator.getWrappedBackground();
                }
                Background tempBackground =
backgroundDecorator.getWrappedBackground();
                this.background = tempBackground;
                fireworkCounterInt = 0;
                try {
```

```
setBackgroundImageLabel(this.background.getBackgroundImage
Source());
                } catch (IOException ex) {

Logger.getLogger(RoomDecorationPage.class.getName()).log(L
evel.SEVERE, null, ex);
                }
            }
        }
        else if (event.getSource() == fireworkAddButton) {
            fireworkCounterInt ++;
            if(fireworkCounterInt <= 3){
                fireworks = true;
                if(fireworkCounterInt==FIREWORKS_COUNTER){
                    fireworkCounter.setText("Max");
                }else{

fireworkCounter.setText(String.valueOf(fireworkCounterInt)
);
                }
                background = new Fireworks(background,
fireworkCounterInt);
                try {
                    String imgSrc =
background.getBackgroundImageSource();
                    setBackgroundImageLabel(imgSrc);
                } catch (IOException ex) {

Logger.getLogger(RoomDecorationPage.class.getName()).log(L
evel.SEVERE, null, ex);
                }
            }
            else{
                fireworkCounterInt --;
                fireworks = false;
            }
        }
```

5.6. Facade Design Pattern

5.6.1 Class SelectDecorationMode

```
private void
lightModeButtonMouseClicked(java.awt.event.MouseEvent evt)
{
        try {
            // TODO add your handling code here:
            abstractFactoryDecoration = new
AbstractFactoryLightDecoration(background);
            decoration = new
DecorationFacade(abstractFactoryDecoration);
            roomDecorationPage = new
RoomDecorationPage(background, decoration);
        } catch (IOException ex) {

Logger.getLogger(SelectDecorationMode.class.getName()).log
(Level.SEVERE, null, ex);
        } catch (LineUnavailableException ex) {

Logger.getLogger(SelectDecorationMode.class.getName()).log
(Level.SEVERE, null, ex);
        } catch (UnsupportedAudioFileException ex) {

Logger.getLogger(SelectDecorationMode.class.getName()).log
(Level.SEVERE, null, ex);
        }
        this.setVisible(false);
    }

    private void
darkModeButtonMouseClicked(java.awt.event.MouseEvent evt)
{
        try {
            // TODO add your handling code here:
            abstractFactoryDecoration = new
AbstractFactoryDarkDecoration(background);
            decoration = new
DecorationFacade(abstractFactoryDecoration);
            roomDecorationPage = new
RoomDecorationPage(background, decoration);
        } catch (IOException ex) {
```

```
Logger.getLogger(SelectDecorationMode.class.getName()).log
(Level.SEVERE, null, ex);
        } catch (LineUnavailableException ex) {

Logger.getLogger(SelectDecorationMode.class.getName()).log
(Level.SEVERE, null, ex);
        } catch (UnsupportedAudioFileException ex) {

Logger.getLogger(SelectDecorationMode.class.getName()).log
(Level.SEVERE, null, ex);
        }
        this.setVisible(false);
    }
```

5.6.2 Class RoomDecorationPage

```
public RoomDecorationPage(Background background,
DecorationFacade decoration) throws IOException,
LineUnavailableException, UnsupportedAudioFileException{
        this.setLocationRelativeTo(null);

        this.background = background;
        this.decoration = decoration;
        decoration.createDecoration();
```

5.6.3 Class DecorationFacade

```
public class DecorationFacade {

    Cushions cushions;
    Pumpkins pumpkins;
    Lights lights;
    Radio radio;
    Broom broom;

    AbstractFactoryDecoration abstractFactoryDecoration;

    public DecorationFacade(AbstractFactoryDecoration
abstractFactoryDecoration){
        this.abstractFactoryDecoration =
abstractFactoryDecoration;
    }

    public void createDecoration(){
        cushions =
this.abstractFactoryDecoration.createCushions();
        pumpkins =
this.abstractFactoryDecoration.createPumpkins();
        lights =
this.abstractFactoryDecoration.createLights();
        radio =
this.abstractFactoryDecoration.createRadio();
        broom =
this.abstractFactoryDecoration.createBroom();
    }

    public Cushions getCushions(){
        return this.cushions;
    }

    public Pumpkins getPumpkins(){
        return this.pumpkins;
    }

    public Lights getLights(){
        return this.lights;
    }

    public Radio getRadio(){
        return this.radio;
    }

    public Broom getBroom(){
        return this.broom;
    }

}
```

5.7. Command Design Pattern

5.7.1. Command class - Interface

```
public interface Command {
    //interface for command
    public void execute();
}
```

5.7.2. OnRadio class - Concrete Command

```
import javax.sound.sampled.Clip;

/**
 *
 * @author Harith Ariffin
 */
public class OnRadio implements Command{
    Radio radio;

    public OnRadio(Radio radio)
    {
        this.radio = radio;
    }
    @Override
    public void execute()
    {
         radio.on();
    }
}
```

5.7.3. OffRadio class - Concrete Command

```
import javax.sound.sampled.Clip;

/**
 *
 * @author Harith Ariffin
 */
public class OffRadio implements Command{
    Radio radio;

    public OffRadio(Radio radio)
    {
        this.radio = radio;
    }
    @Override
    public void execute()
    {
         radio.off();
    }
}
```

5.7.4. RemoteControl class - Invoker

```
public class RemoteControl {
    Command onCommand;
    Command offCommand;

    public RemoteControl(){
        Command noCommand = new NoRadioCommand();
        onCommand = noCommand;
        offCommand = noCommand;
    }

      public  void  setCommand(Command  onCommand,  Command
offCommand){
        this.onCommand = onCommand;
        this.offCommand = offCommand;
    }

    public void onButtonWasPushed(){
        onCommand.execute();
//        undoCommand = onCommands[slot];
    }

    public void offButtonWasPushed(){
        offCommand.execute();
//        undoCommand = offCommands[slot];
    }

//    public String toString(){
//        StringBuffer stringBuff = new StringBuffer();
//              stringBuff.append("\n------  Remote  Control
------\n");
//        for(int i  =0; i < onCommands.length; i++){
//                stringBuff.append("[slot  "  +  i  +  "]  "  +
onCommands[i].getClass().getName()    +    "         "    +
offCommands[i].getClass().getName() + "\n");
//        }
//        return stringBuff.toString();
//    }
}
```

5.7.5. Radio class - Receiver

```
public abstract class Radio {

    String radioImageSource;
    Background background;
    int x, y;
    Command command1;

    RadioMusicBehavior radioMusicBehavior;
    Clip clip;

public void on(){
        performPlayMusic();
    }

    public void off(){
        clip.stop();
    }

    public void setCommand(Command command){
        this.command1=command;
    }
}
```

5.7.6. RadioButtonsPage class - Client (GUI)

```
public class RadioButtonsPage extends javax.swing.JFrame {

    /**
     * Creates new form RadioButtonsPage
     */
    private static Radio radioDeco;
    RemoteControl remoteControl = new RemoteControl();


private void
onRadioButtonActionPerformed(java.awt.event.ActionEvent
evt) {
        // TODO add your handling code here:

        OnRadio onRadio = new OnRadio(radioDeco);
        OffRadio offRadio = new OffRadio(radioDeco);
        remoteControl.setCommand(onRadio, offRadio);
        remoteControl.onButtonWasPushed();
//        radioDeco.setCommand(new OnRadio(new
Radio(radioDeco.clip)));
//        radioDeco.command1.execute();
//        this.setVisible(false);
    }

    private void
offRadioButtonActionPerformed(java.awt.event.ActionEvent
evt) {
        // TODO add your handling code here:
//        RemoteControl remoteControl = new
RemoteControl();
        OnRadio onRadio = new OnRadio(radioDeco);
        OffRadio offRadio = new OffRadio(radioDeco);
        remoteControl.setCommand(onRadio, offRadio);
        remoteControl.offButtonWasPushed();
//        this.setVisible(false);
    }
}
```