Cory Heitkamp

Abstract Factory Pattern

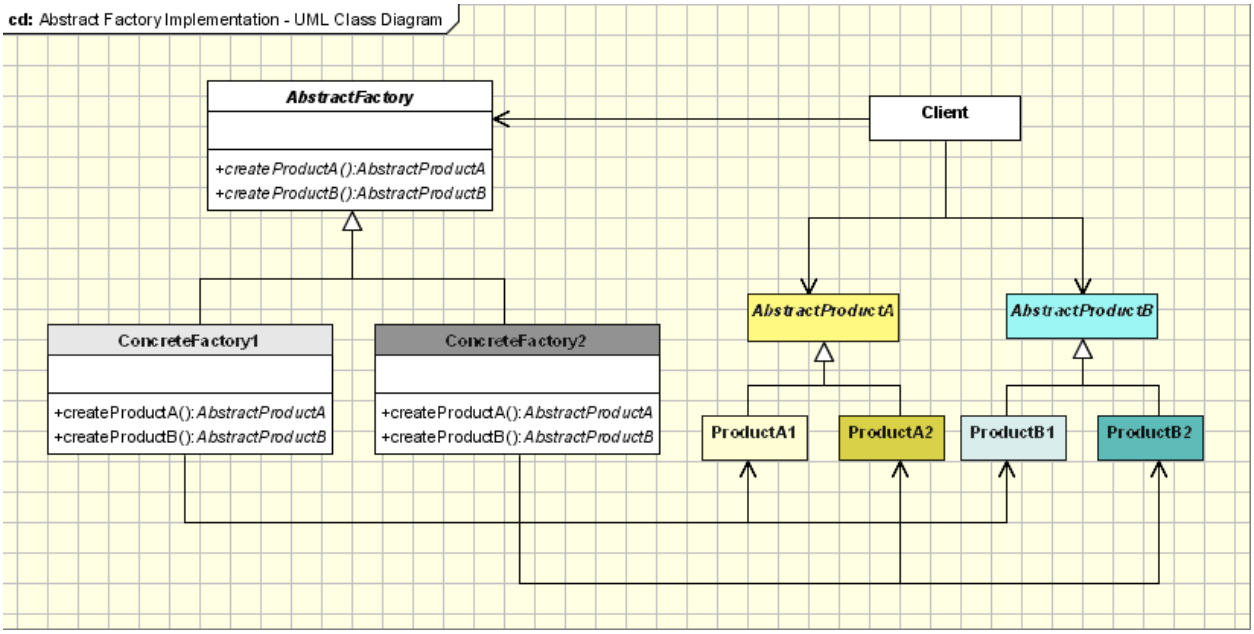
Design Patterns

9/26/16

Introduction:

For this week's design pattern, we were given the challenge of designing an application that utilizes the Abstract Factory pattern. The Abstract Factory provides an interface for creating similar objects. My application is used for creating a movie, whether it is an action movie or comedy, and then the product movie is described.

UML Diagram: This UML Diagram seen below shows just how the Abstract Factory Pattern is used, so that the factories can create any needed products.



This table below summarizes the classes that I've created and used to make the facade pattern.

Abstract Factory	Creates the abstract method for creating movie objects.
Comedy Factory	Defines the method to create a Comedy Movie.
Movie	An abstract class that has an undefined synopsis method, which will be used display the info.
Comedy Movie	Defines the synopsis, as well as defining variables needed for a comedy.
Action Movie	Defines the synopsis, as well as defining variables needed for a action.
Action Factory	Defines the method to create a Action Movie.
Form	My windows form application.

Narrative:

When I began writing this pattern, I began by defining the classes that I would be using, specifically the Movie, ActionMovie, and ComedyMovie classes. The Movie class is an abstract class, and it's function is used for displaying the product information.

```
public abstract class Movie
    {

        public abstract string synopsis();
    }
```

I then defined the Action Movie class. It inherits from the Movie class. It uses three string variables to contain the details of the movie.

```
  public class ActionMovie : Movie
    {
        string myTitle;
        string myGoodGuy;
        string myBadGuy;
        public ActionMovie(string cooltitle, string good, string bad)
        {
            myBadGuy = bad;
            myGoodGuy = good;
            myTitle = cooltitle;
        }
```

It also contains the synopsis function, which is used to report about the movie.

```
public override string synopsis()
        {
            return "In "+ myTitle +", the new blockbuster from Michael Bay, " + myGoodGuy
+
                  " must stop the maniac " + myBadGuy + " from blowing up the world.";
        }
```

I then wrote the Comedy Movie Class, which contains different variables to describe the film.

```
  public class ComedyMovie : Movie
    {
        public string myTitle;
        public string mydirector;
        public string mystar;
        public ComedyMovie(string cooltitle, string star, string director )
        {
            myTitle = cooltitle;
            mystar = star;
            mydirector = director;

        }
```

It also contains the synopsis function.

```
  public override string synopsis()
        {
            return myTitle + " is the blockbuster comedy of the year starring " +
                 mystar + " as a loveable slacker just trying to get by in this " +
mydirector + " film.";
        }
```

I then made the abstract Factory class, which contains the function to make movies.

```
public abstract class Factory
    {
        public abstract Movie makeMovie();

    }
```

The comedy factory then inherits from that class in order to produce comedy movies. It has a struct that passes in the name of the movie, because it is what the user will input. It then makes the comedy film.

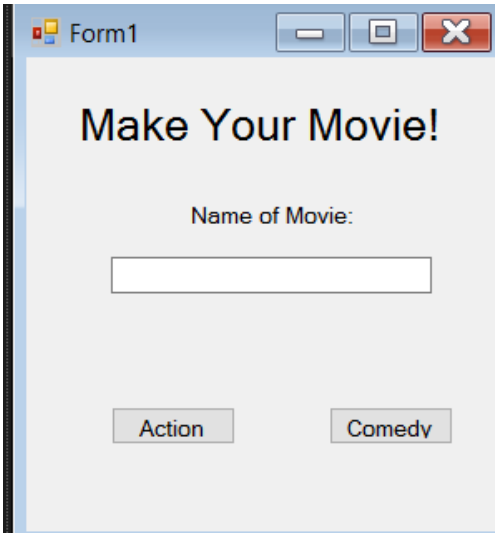
```
namespace AbstractFactory
{
    public class ComedyFactory : Factory
    {
        public string name;
        public ComedyFactory(string newname)
        {
            name = newname;
        }
        public override Movie makeMovie()
        {
            return new ComedyMovie(name, "Seth Rogen", "Judd Appatow");
        }
    }
}
```

The names Seth Rogen and Judd Appatow are used because they make a lot of comedy movies. I then made the Action Factory class. It is quite similar to the Comedy Factory class, but it produces action films.

```
public class ActionFactory : Factory
    {
        public string name;
        public ActionFactory(string newname)
        {
```

```
            name = newname;
        }
        public override Movie makeMovie()
        {
            return new ActionMovie(name, "Max Power", "Dr. Malicious");
        }
```

I then made the Form itself. It can be seen below:



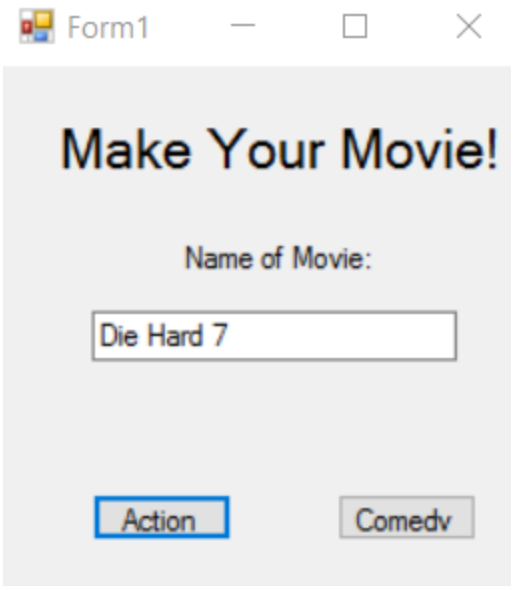
Objects of the Factory and Movie classes are made at the beginning of the code. The action button then defines those objects, and uses the synopsis function to display the details in a message box.

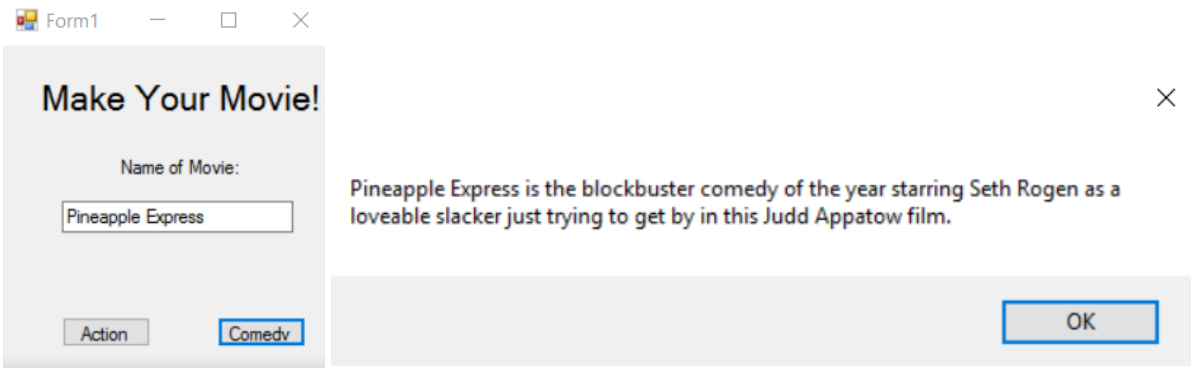
```
    private void actionbutton_Click(object sender, EventArgs e)
        {
            film = new ActionFactory(nametextBox.Text);
            Blockbuster = film.makeMovie();
            MessageBox.Show(Blockbuster.synopsis());
        }
```

The comedy button is very similar.

```
    private void comedybutton_Click(object sender, EventArgs e)
        {
            film = new ComedyFactory(nametextBox.Text);
            Blockbuster = film.makeMovie();
            MessageBox.Show(Blockbuster.synopsis());
        }
```

When you run it and create a movie, it looks like so:





Reflection: The Abstract Factory Pattern was a pattern I enjoyed making, as it seems like a pattern that will be quite useful to use. I believe my program does a good job of using this pattern.