Tori Lentz

Dave Retterer

Design Patterns

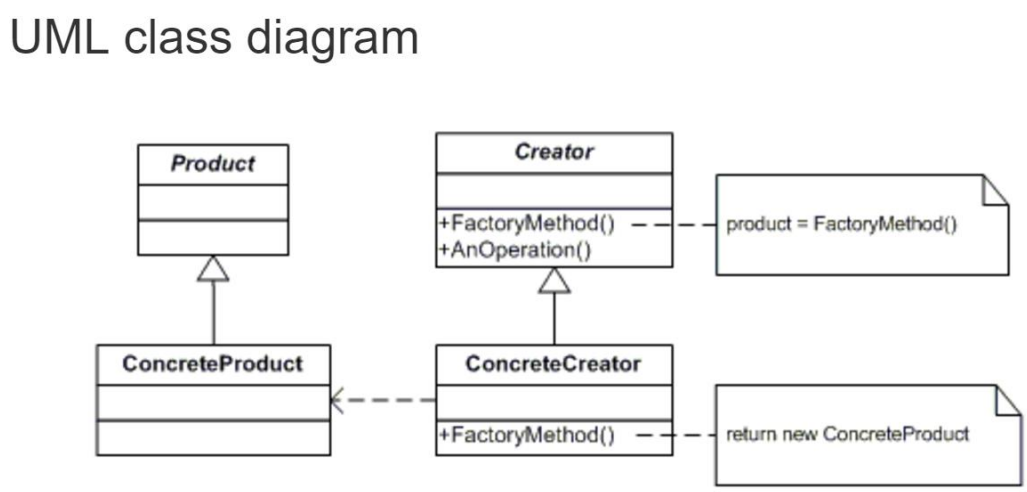
November 21, 2016

Factory Method Pattern

Introduction

The purpose of this assignment is to create and implement the Factory Method Pattern. This pattern, also known as the Virtual Constructor, works off of the same idea in which a library works off of. This pattern uses different abstract classes for defining and then maintaining the relationships between the objects. The library knows when an object needs to be created, however, it does not know what type of object will need creating and leaves function up to its subclasses.

UML Diagram



There are several different parts that make up this pattern. The first part is the Creator class which declares the factory method and will return an object of the type Product. Another thing that may occur within this class is, the factory method may be called in order to create a Product object. The next class is the Concrete Creator which overrides the factory method to return an instance of a Concrete Product. The Concrete Product class implements the Product interface and finally, is the Product class which defines the interface of objects that the factory method will create.

Code and Description

Creator

```
public abstract class Game //Creator Abstract Class
{
    private List<GameElements> _gameElements = new List<GameElements>();

    public Game()
    {
```

```
            this.CreateElements();
        }

        public List<GameElements> Games
        {
            get { return _gameElements; }
        }

        public abstract void CreateElements();
    }
```

This class initiates the list of elements that I am going to use throughout my program. The list is a list of game elements for different kinds of games.

Product

```
    public abstract class GameElements //Product Abstract Class
    {
    }
```

My abstract product class creates the class for game elements.

Concrete Creator 1

```
    public class Platform : Game
    {
        public override void CreateElements()
        {
            Games.Add(new Characters());
            Games.Add(new Objective());
            Games.Add(new TwoDLayout());
        }
    }
```

All of my creator classes create the different games I am dealing with and then they add on the different game elements that have to do with that particular type of game.

Concrete Creator 2

```
    public class RPG : Game
    {
        public override void CreateElements()
        {
            Games.Add(new Characters());
            Games.Add(new Skills());
            Games.Add(new World());
            Games.Add(new Objective());
            Games.Add(new Map());
            Games.Add(new ThreeDLayout());
        }
    }
```

Concrete Creator 3

```
    public class Strategy : Game
    {
        public override void CreateElements()
        {
            Games.Add(new Objective());
            Games.Add(new HowToPlay());
        }
    }
```

Product 1

This is an example of one of my product classes that will be added as a game element to my concrete creator class.

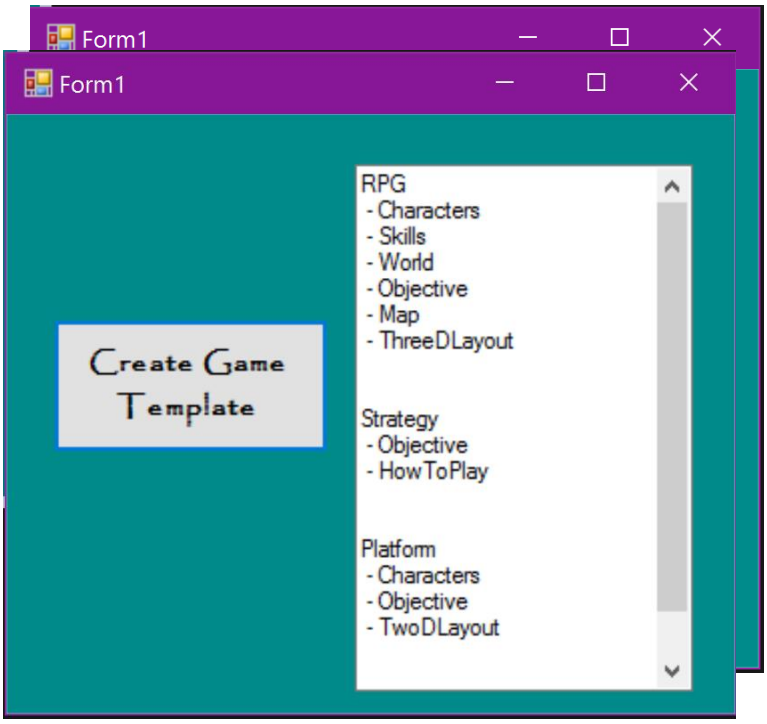
```
    public class Characters : GameElements
//Concrete Product 1
    {
    }
```

Form

Here is my form code. Within my form, I have my three different game types initialized from the array. Within my button click event, I have an algorithm that will print the title of each type of game with each element within that game below them.

```
        public partial class Form1 : Form
    {
        Game[] games = new Game[3];

        public Form1()
        {
            InitializeComponent();

            games[0] = new RPG();
            games[1] = new Strategy();
            games[2] = new Platform();
        }

        private void btn_List_Click(object sender, EventArgs e)
        {

            foreach (Game game in games)
            {
                tb_list.Text = tb_list.Text + game.GetType().Name + "\r\n";
                foreach (GameElements gameelement in game.Games)
                {
                    tb_list.Text = tb_list.Text + " - " + gameelement.GetType().Name +
"\r\n";
                }
                tb_list.Text = tb_list.Text + "\r\n\r\n";
            }
        }
    }
```

Screen Shots



Form after button click

Observations

Overall, this assignment went well. The only part I struggled with was trying to output all of the information back out to the textbox. After looking at some other programs, either that I had previously done or that I found on the internet, I was able to figure it out and get my code to work the way I wanted it to.