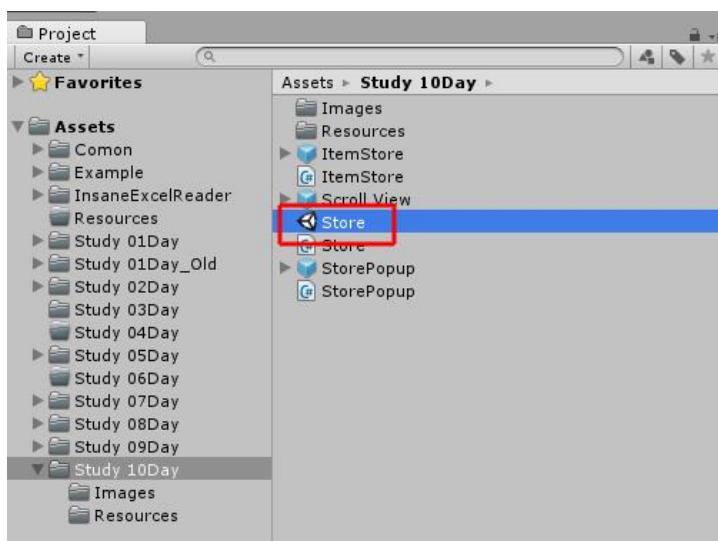
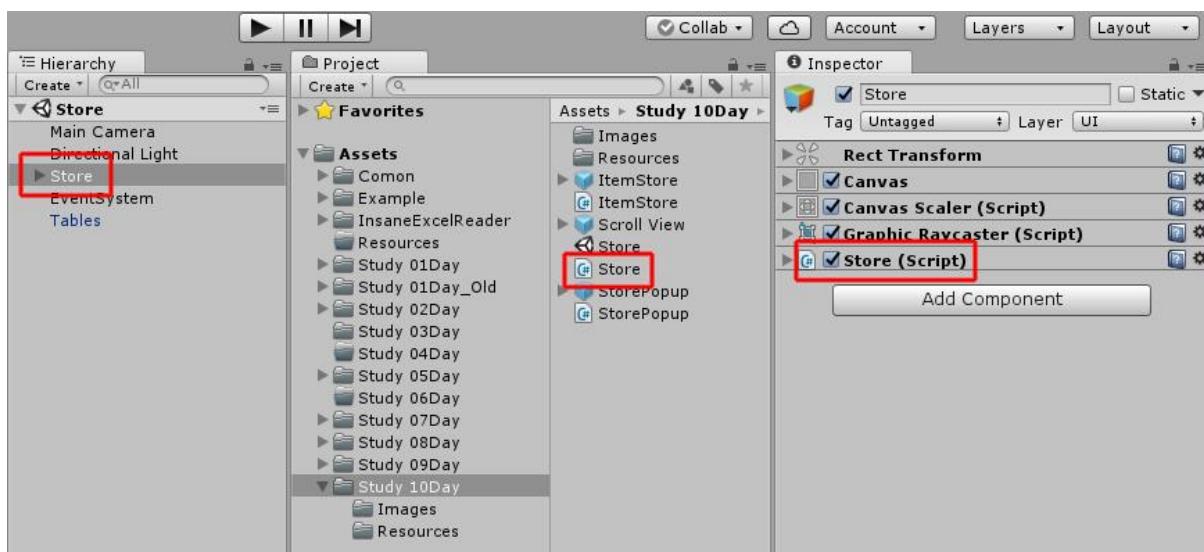


이번 시간에는 엑셀테이블에서 아이템 정보를 받아서 상점 구현을 해 봅니다. 구매한 상품은 회색으로 표현되고 더 이상 구매가 되지 않습니다. 구매한 상품은 파일입출력을 통해서 파일로 남게 되고, 파일에 있는 정보 값으로 구매한 상품인가, 아직 구매하지 않는 상품인가를 비교하게 됩니다.

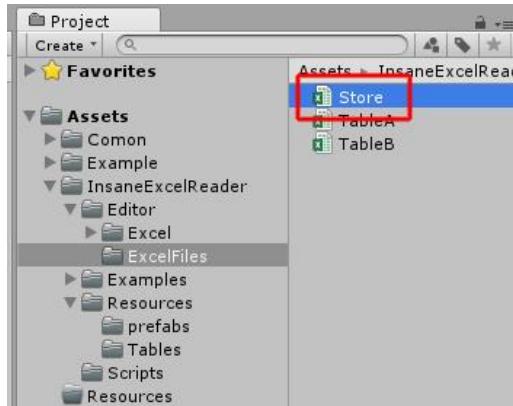
그럼 본격적으로 시작해 보도록 하겠습니다. Store라는 이름으로 쓴을 저장해 줍니다.



Store라는 이름의 Canvas오브젝트를 만들어 주고, Store라는 이름의 스크립트도 만들어 주고, Store오브젝트에 Store스크립트를 컴포넌트로 등록해 줍니다.



Store라는 이름으로 엑셀파일을 만들어 줍니다.



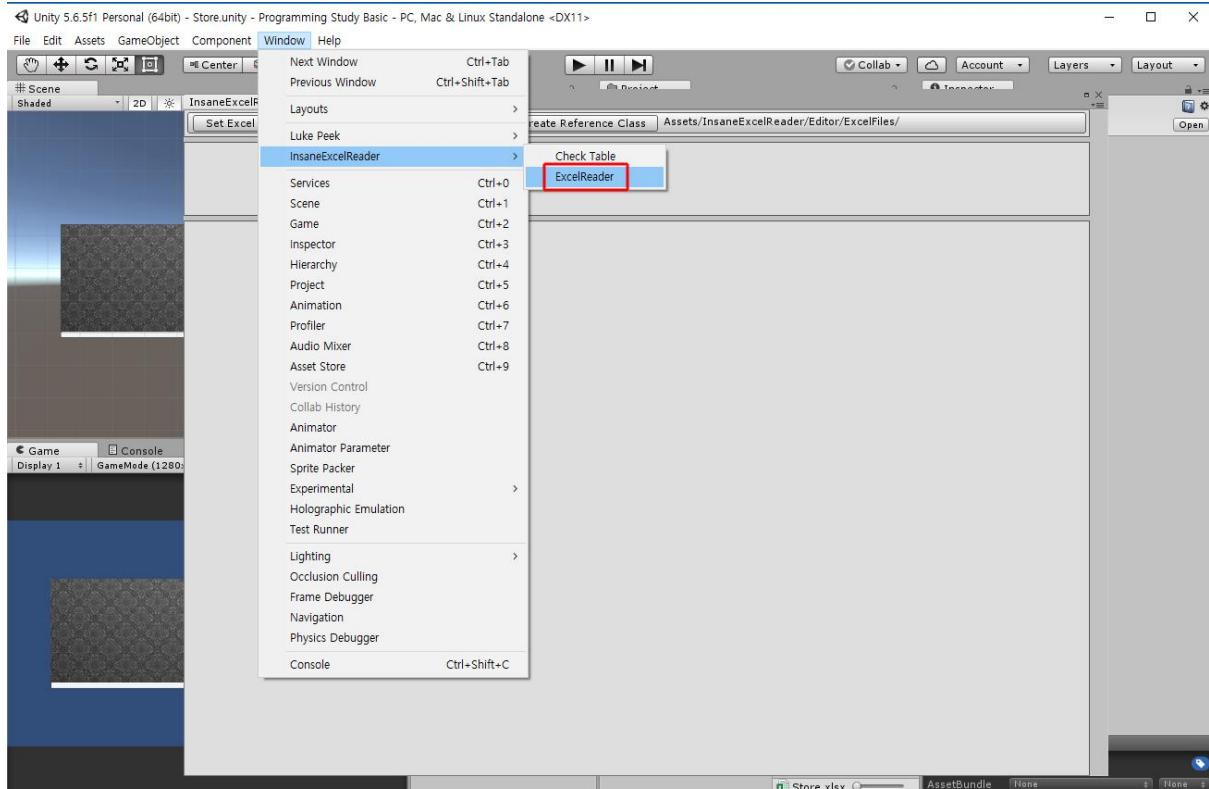
Store엑셀파일을 오픈하고 Store라는 이름으로 테이블을 만들어 주고 다음과 같이 값을 줍니다.

A screenshot of Microsoft Excel showing a table named 'Store'. The table has columns A through F. The data is as follows:

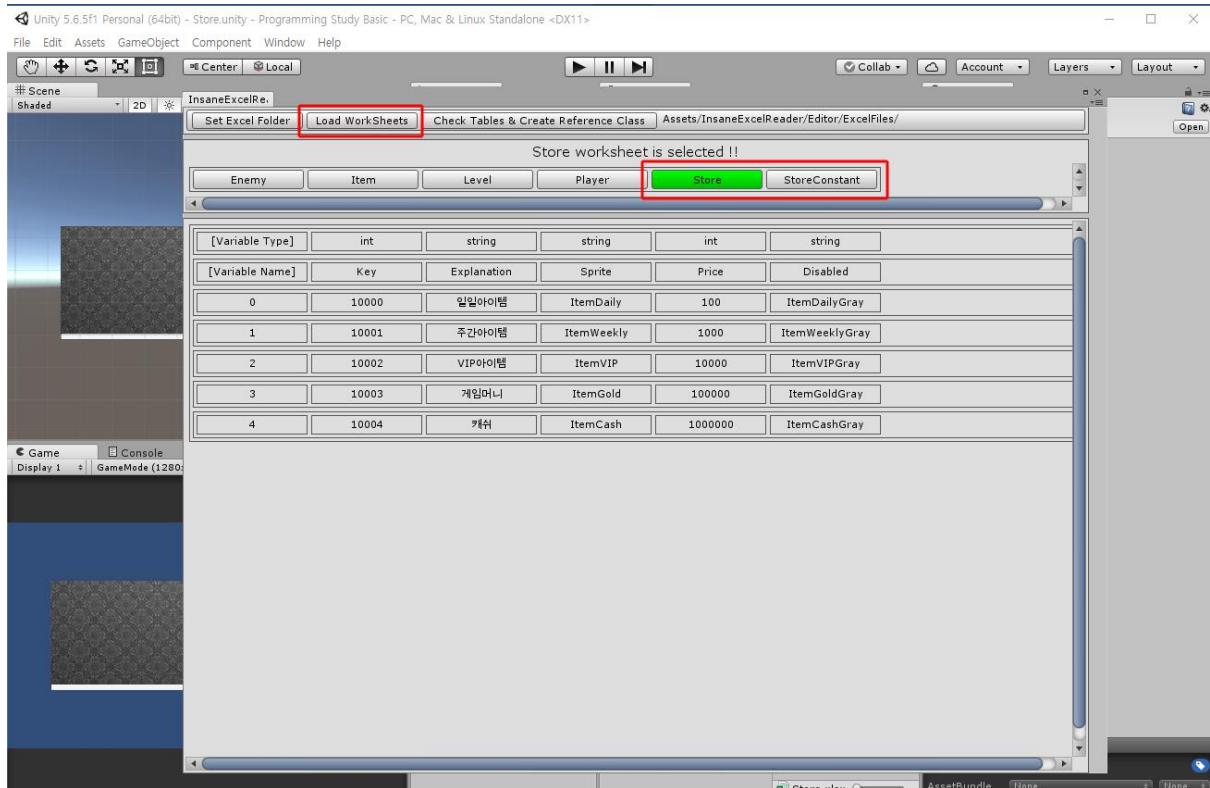
	A	B	C	D	E	F
1	int	string	string	int	string	
2	Key	Explanation	Sprite	Price	Disabled	
3	10000	일일아이템	ItemDaily	100	ItemDailyGray	
4	10001	주간아이템	ItemWeekly	1000	ItemWeeklyGray	
5	10002	VIP아이템	ItemVIP	10000	ItemVIPGray	
6	10003	게임머니	ItemGold	100000	ItemGoldGray	
7	10004	캐쉬	ItemCash	1000000	ItemCashGray	
8						
9						

이후에 Store테이블 정보를 Dictionary타입으로 가져올 예정입니다. 테이블을 보면 Dictionary타입으로 가져오기 위한 Key값과 아이템 설명, 아이템의 이미지에 해당하는 이름, 아이템 가격 등의 정보가 있는 것을 볼 수 있습니다.

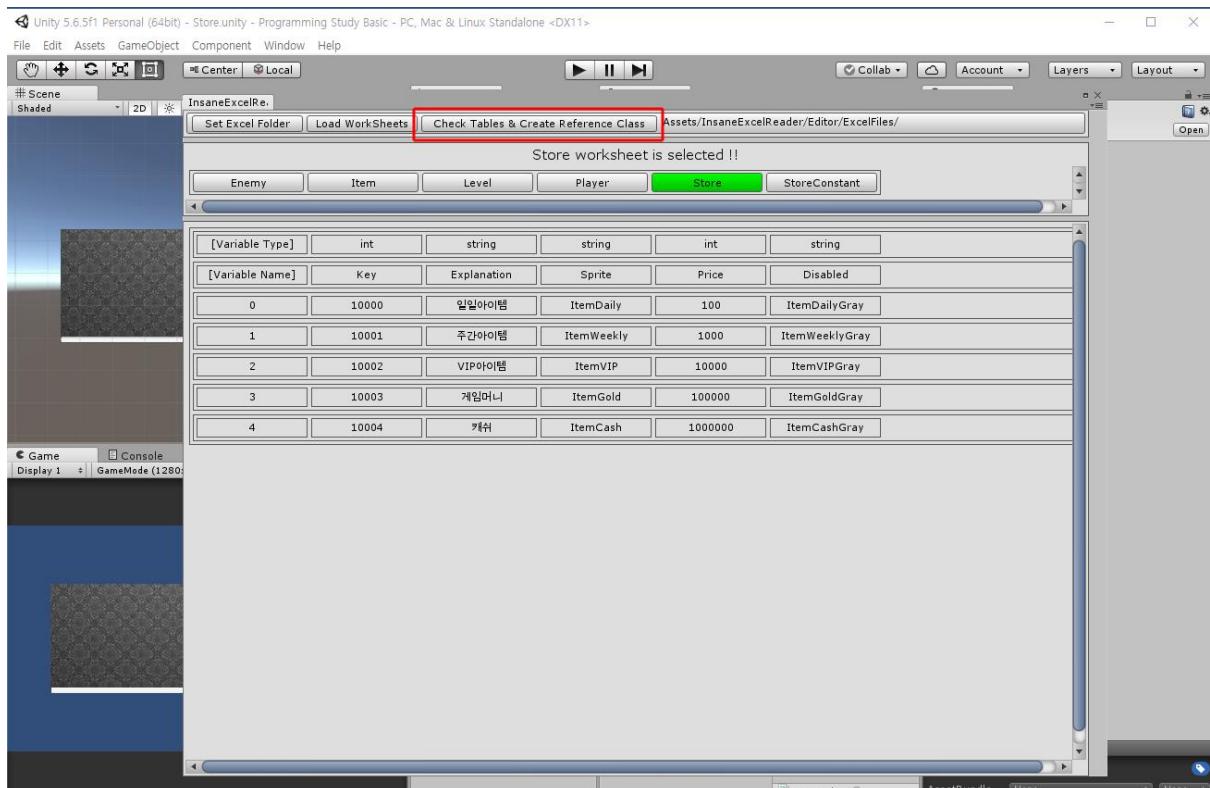
이제 엑셀파일의 내용을 바이너리 파일로 저장하기 위해서 엑셀리더틀을 오픈합니다.



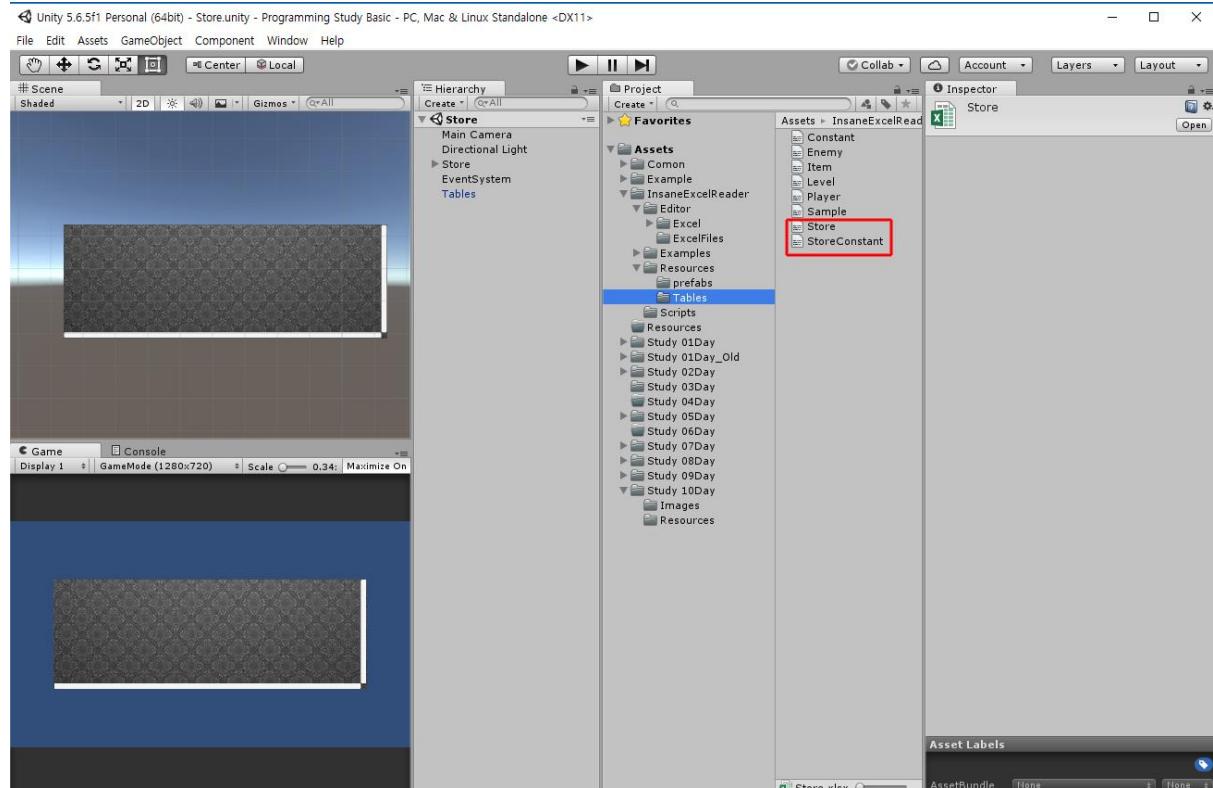
LoadWorkSheets버튼을 클릭해서 Store테이블과 StoreConstant테이블의 내용이 맞게 표현되었는지 확인해 봅니다.



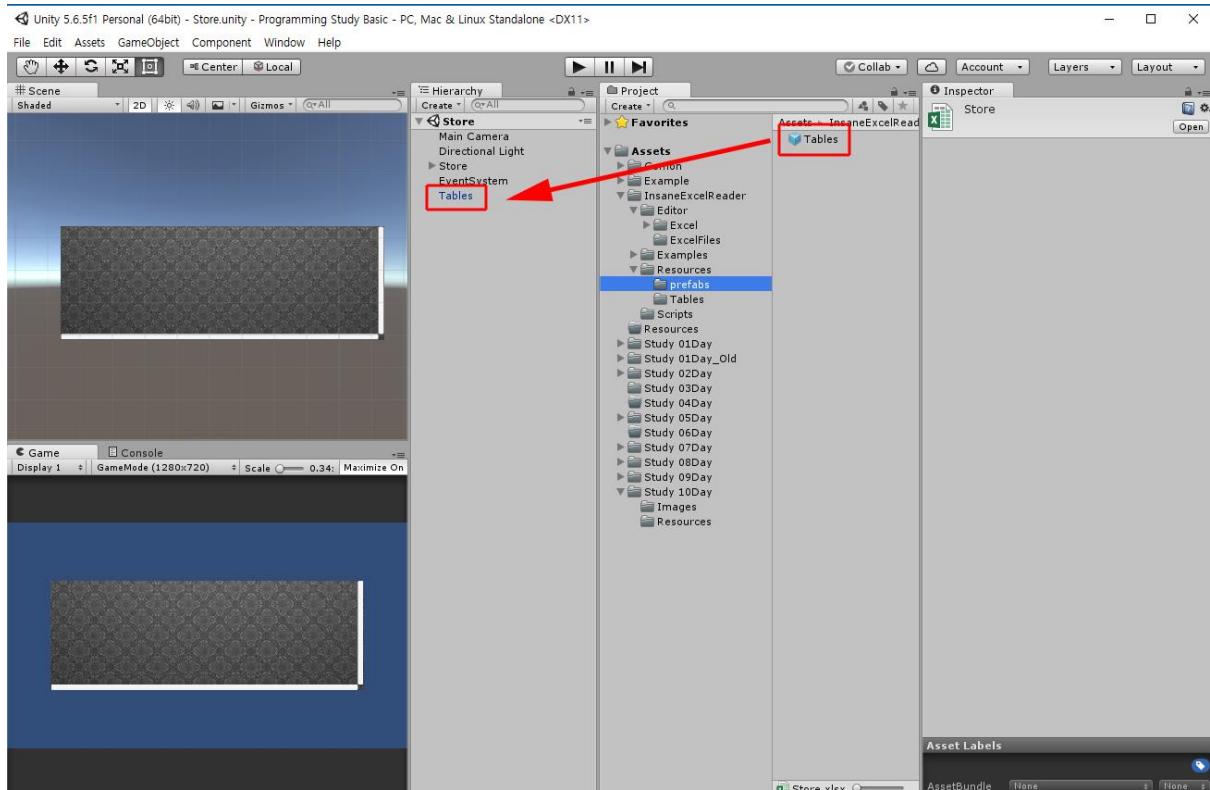
Check Tables & Create Reference Class 버튼을 클릭해서 테이블의 내용을 바이너리 파일로 저장합니다.



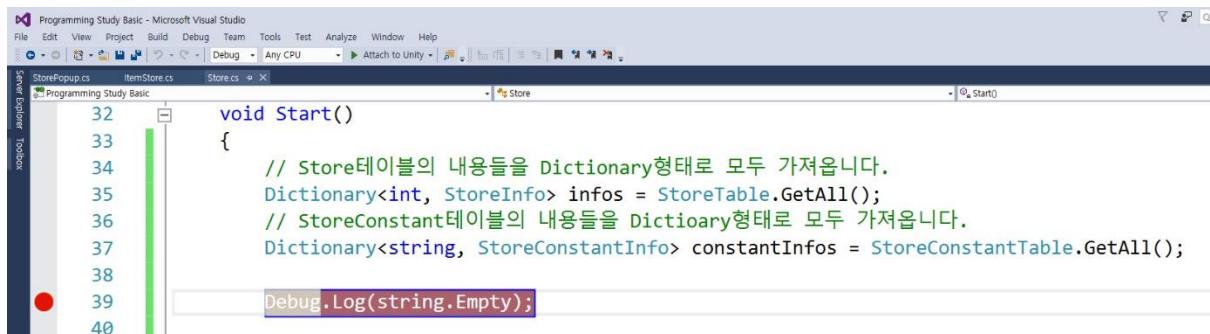
저장된 바이너리 파일을 확인해 봅니다.



테이블 정보를 가져오기 위해서는 씬에 Tables프리팹에 존재해야 합니다. Tables프리팹을 씬에 드래그 해 줍니다.



Store스크립트를 오픈해서 테이블 정보를 읽어 봅니다. Start() 함수에서 추가해줍니다.



디버거로 값이 제대로 들어왔는지 확인해 봅니다.



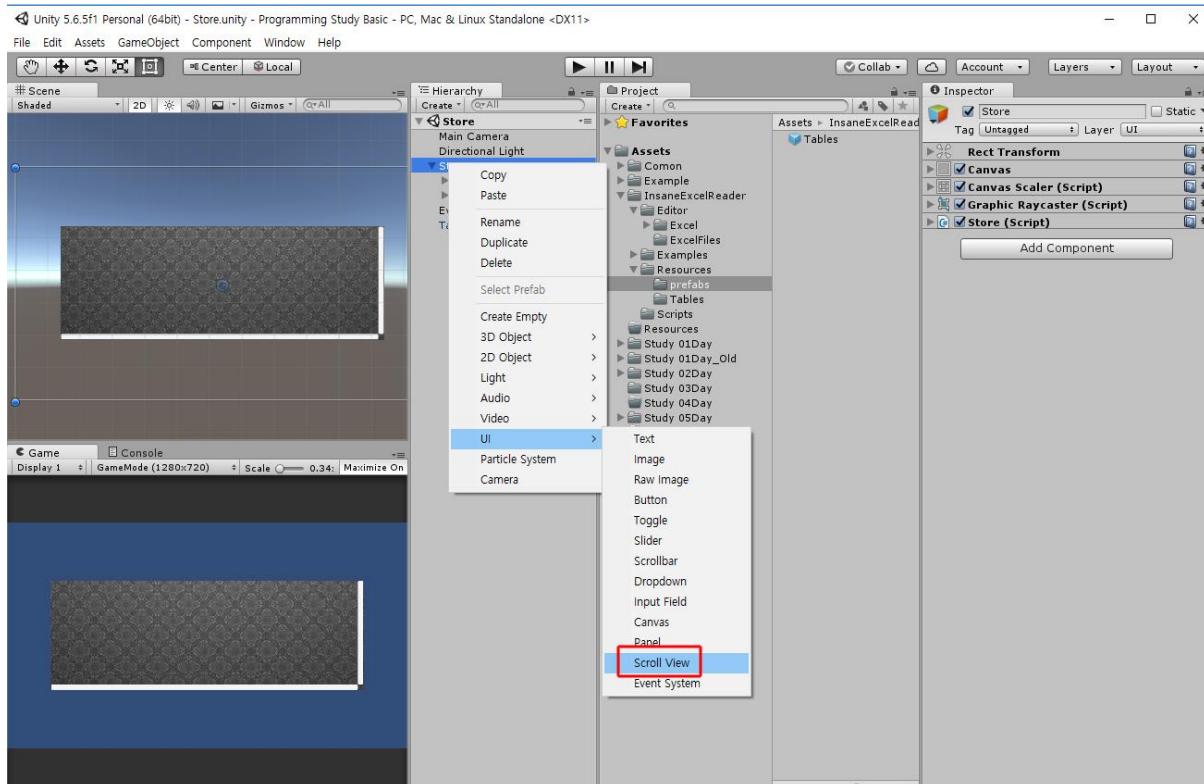
```
void Start()
{
    // Store테이블의 내용들을 Dictionary형태로 모두 가져옵니다.
    Dictionary<int, StoreInfo> infos = StoreTable.GetAll();
    // StoreConstant테이블의 내용들을
    Dictionary<string, StoreConstant> constantInfos = StoreConstantTable.GetAll();
}

Debug.Log(string.Empty);
```

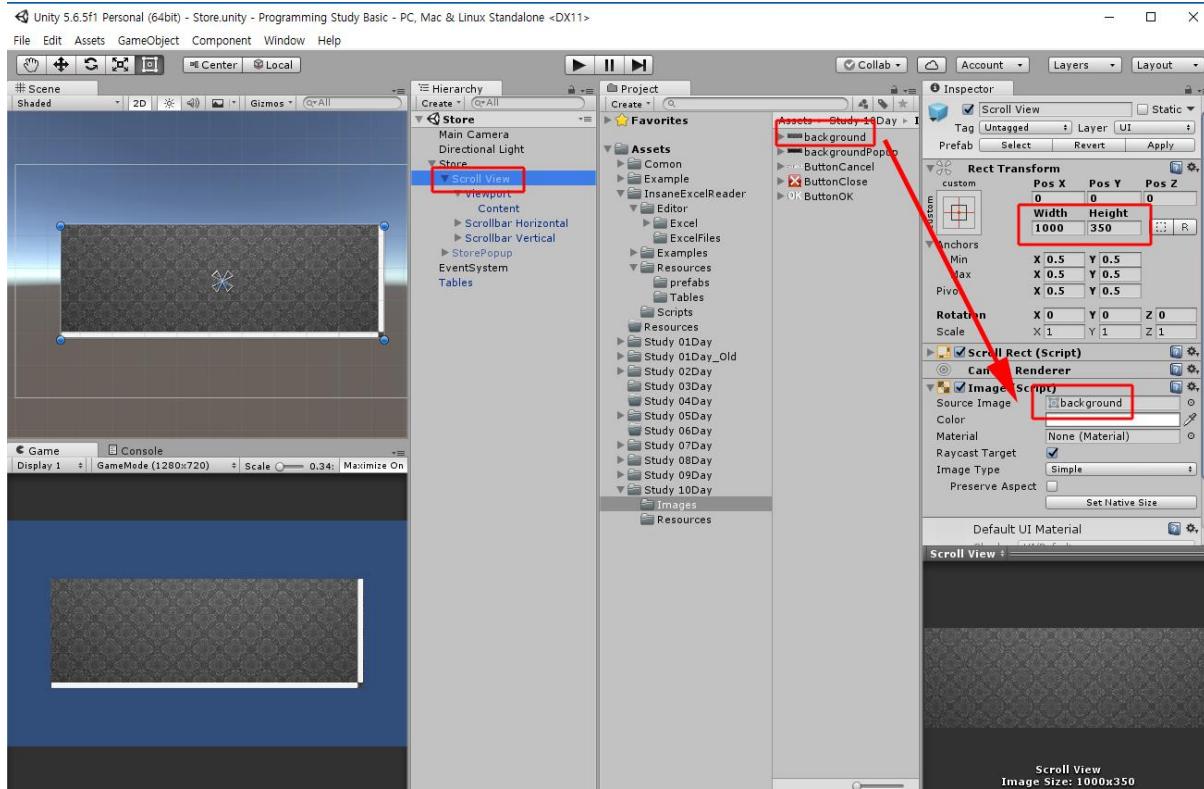


```
void Start()
{
    // Store테이블의 내용들을 Dictionary형태로 모두 가져옵니다.
    Dictionary<int, StoreInfo> infos = StoreTable.GetAll();
    // StoreConstant테이블의 내용들을 Dictioary형태로 모두 가져옵니다.
    Dictionary<string, StoreConstantInfo> constantInfos = StoreConstantTable.GetAll();
    Debug.Log(string.Empty);
}
```

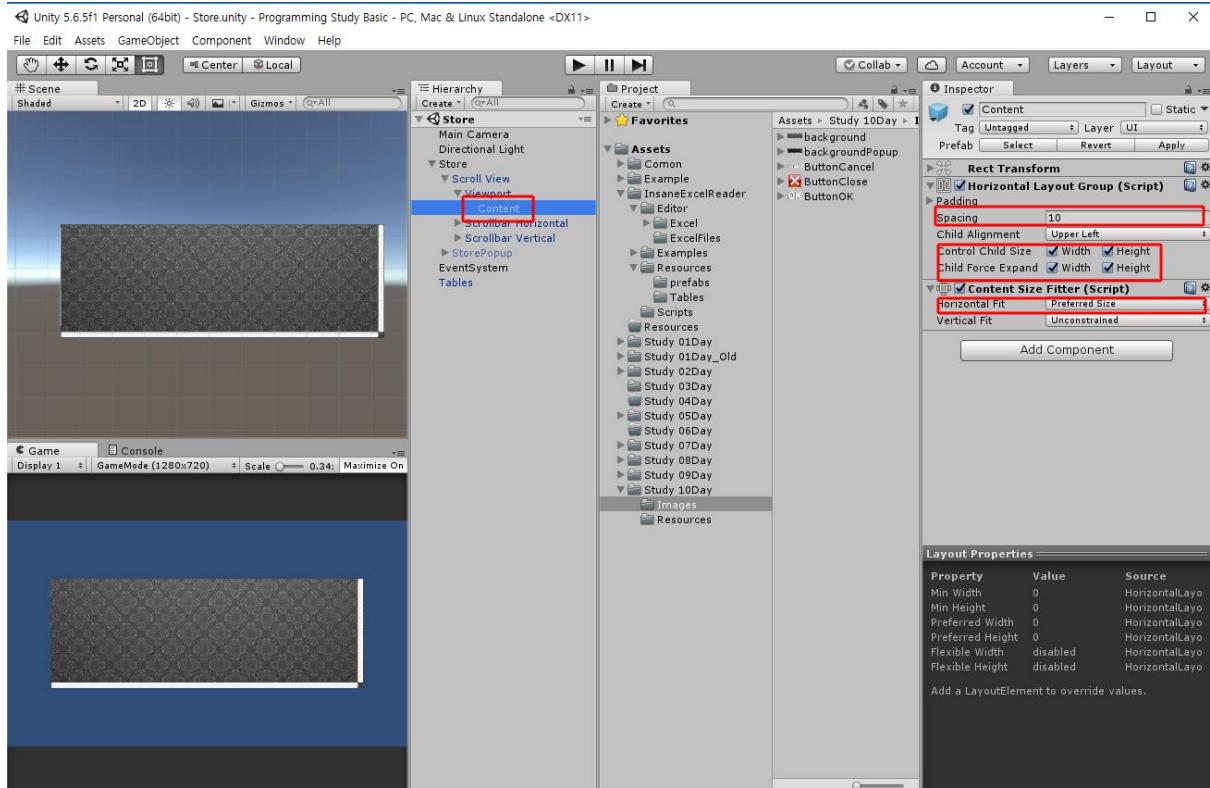
상점 아이템들을 스크롤뷰의 아이템들로 표시할 예정입니다. Store오브젝트를 선택한 상태에서 오른쪽 마우스를 클릭하고 UI – Scroll View를 선택해 줍니다.



1000 X 350의 스크롤뷰의 배경이미지를 준비했습니다. 인스펙터에서 적용해 줍니다.



스크롤뷰 아이템들은 Content오브젝트가 관리해야 합니다. 아이템들을 관리하기 위해서는 2개의 컴포넌트가 필요합니다. 적용해 줍니다.



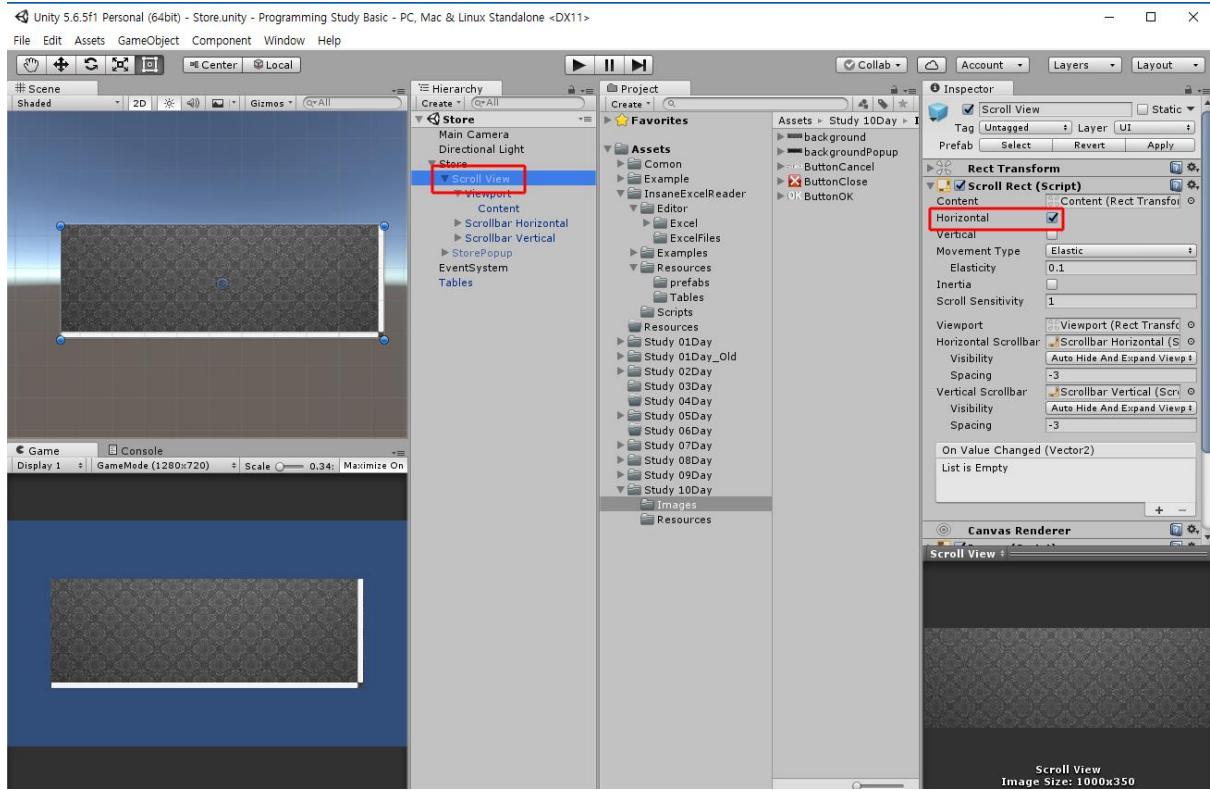
상점의 경우 좌우로 스크롤이 됩니다. 따라서 Horizontal Layout Group컴포넌트가 필요합니다. 또한 Content Size Filter컴포넌트가 필요한데요. 마찬가지로

Horizontal Fit항목에서 Preferred Size를 선택해 줍니다.

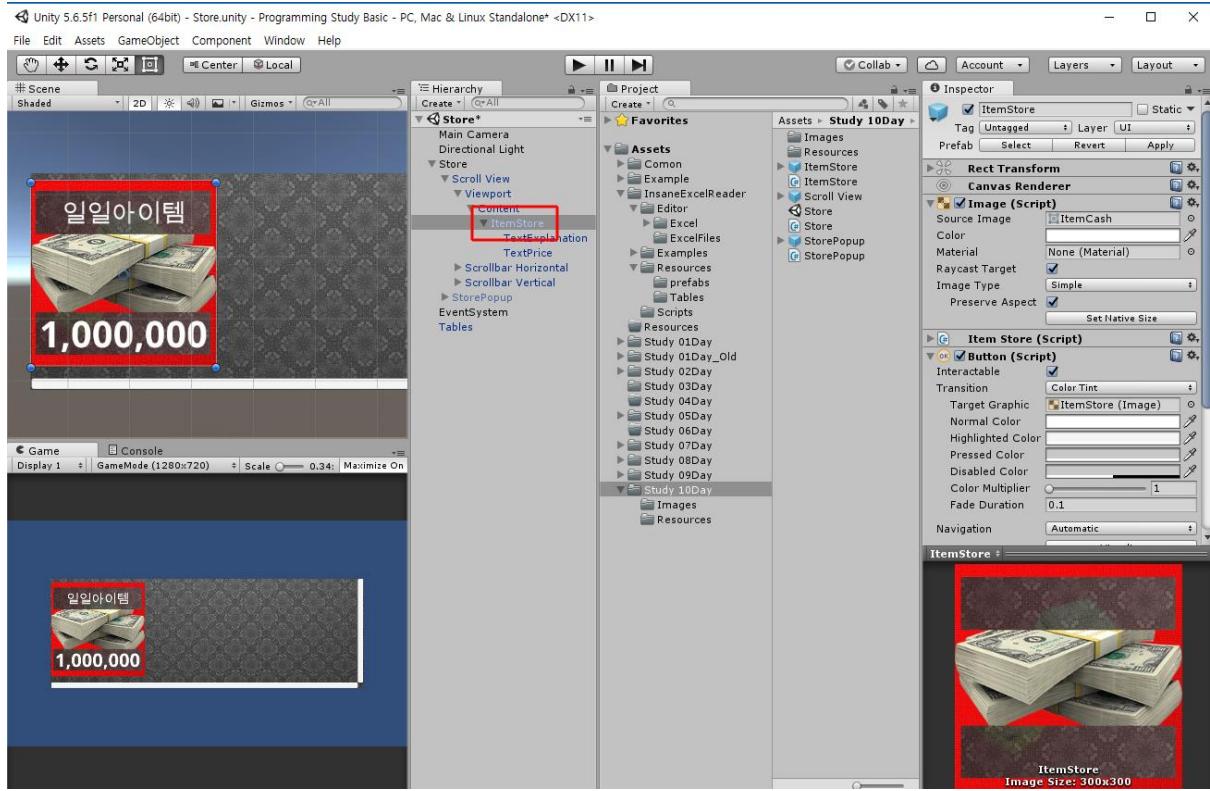
만일 좌우가 아닌 상하로 스크롤이 된다고 하면 Vertical Layout Group컴포넌트가 필요합니다. 또한 Content Size Filter컴포넌트가 필요한데요. 마찬가지로

Vertical Fit항목에서 Preferred Size를 선택해 주어야 합니다.

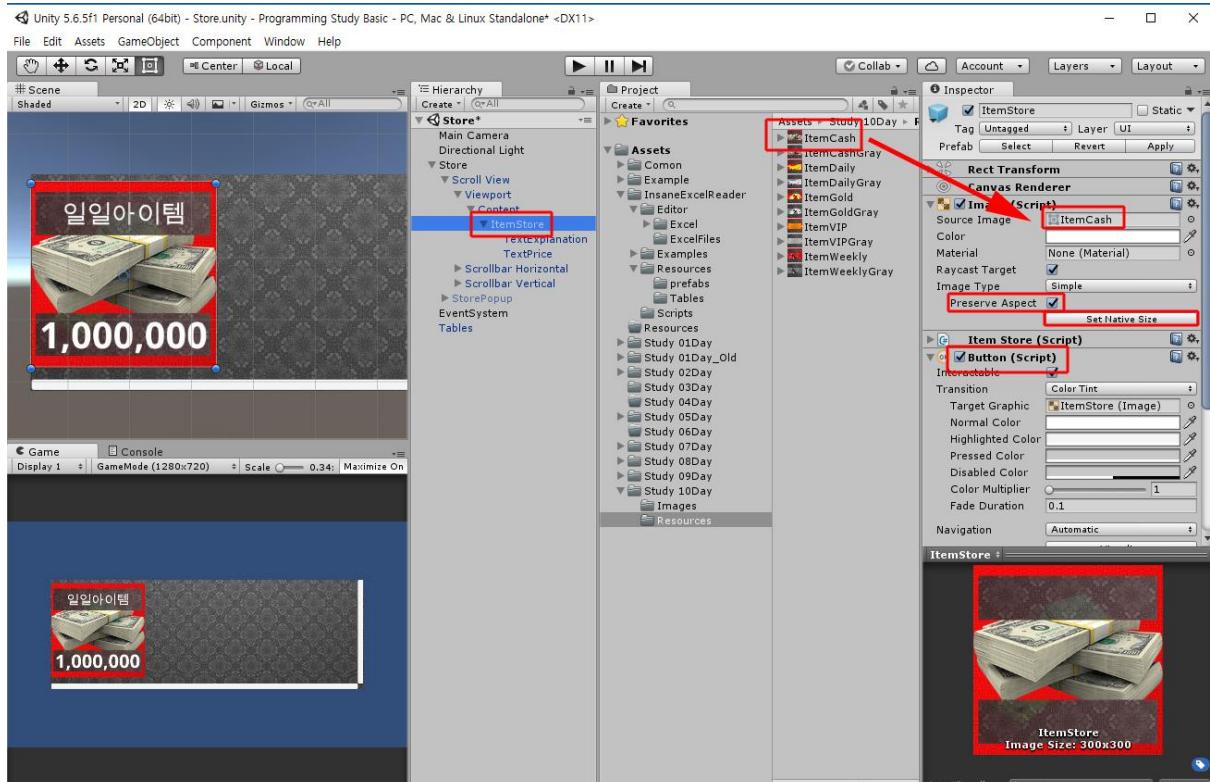
하나 더 해줘야 하는 것이 있습니다. 좌우로만 스크롤이 될 것이기 때문에 Scroll Rect컴포넌트에서 Horizontal만 체크해 줍니다.



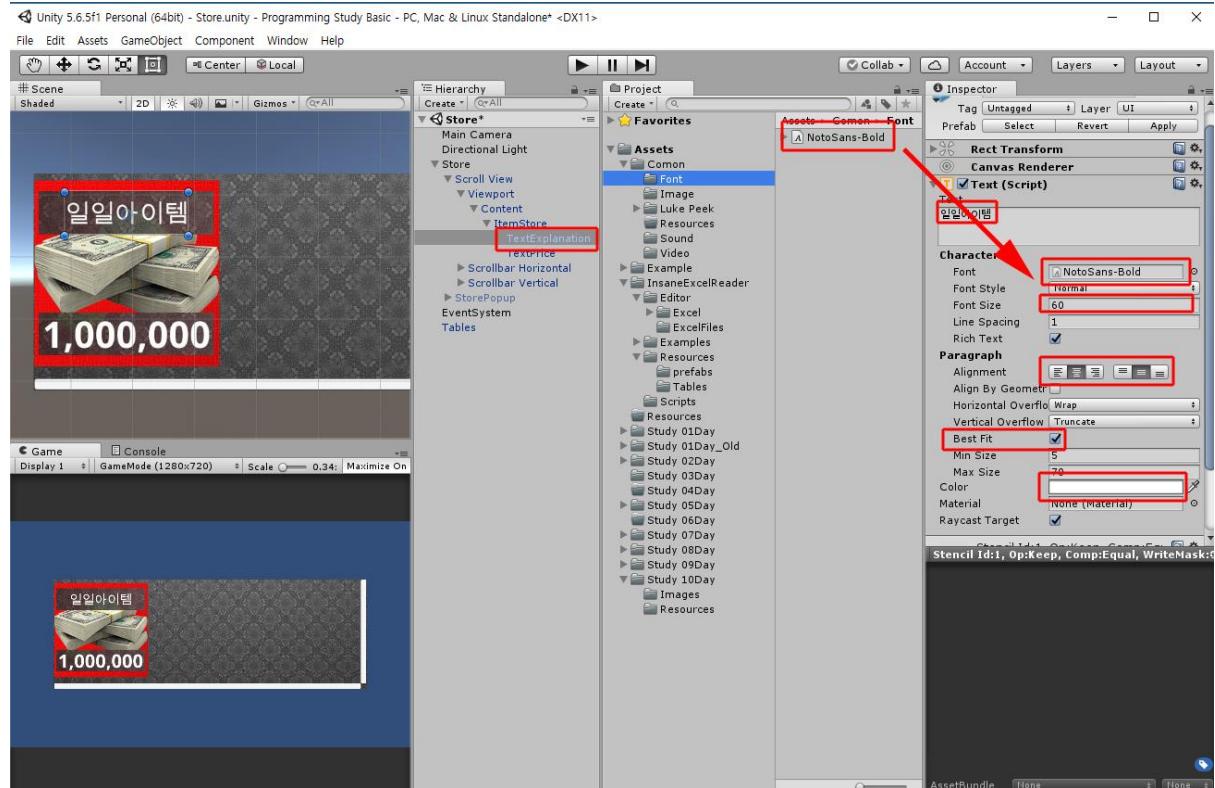
기본적인 스크롤뷰 세팅이 모두 끝났습니다. 이제 아이템 프리팹을 만들어 줍니다. Content오브젝트를 선택한 상태에서 오른쪽 마우스를 클릭하고 Image UI를 선택하고 생성된 Image UI의 이름을 ItemStore로 변경해 줍니다.



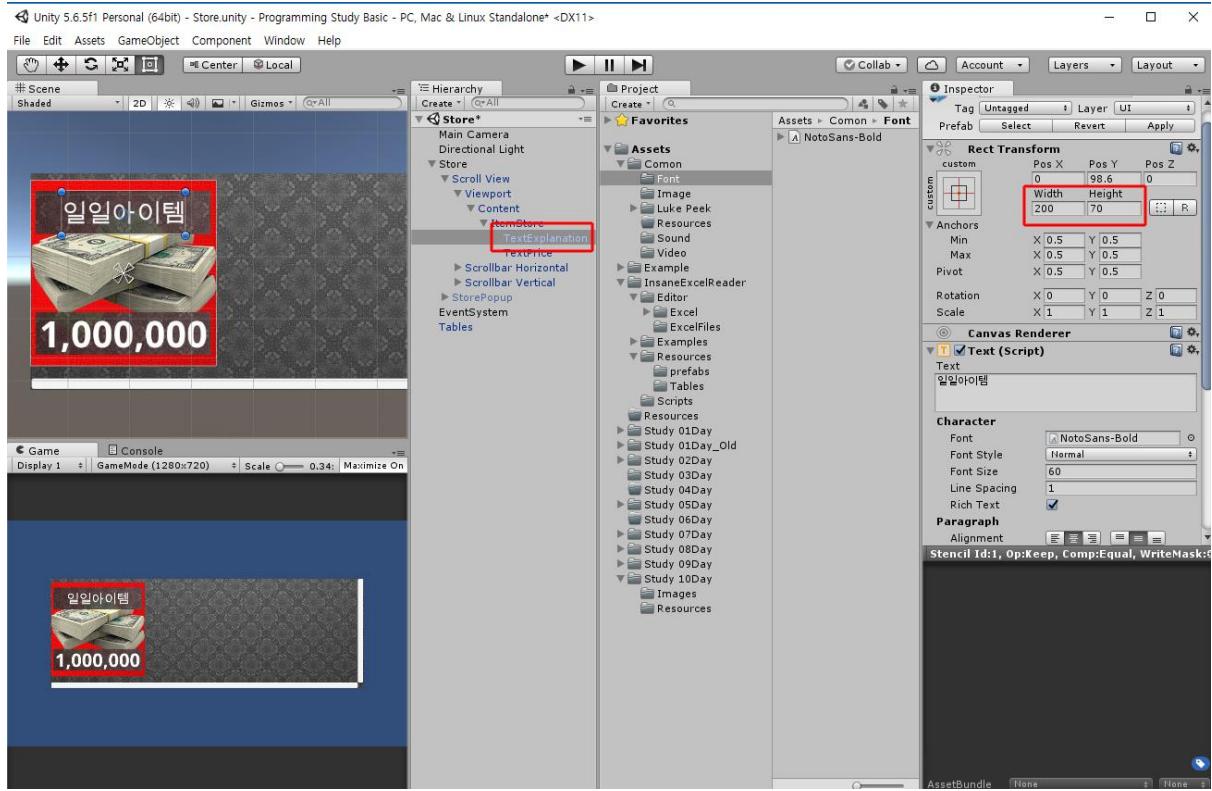
스프라이트도 적용해 주시고, 버튼으로 쓸 것이기 때문에 Button 컴포넌트도 추가해 줍니다.



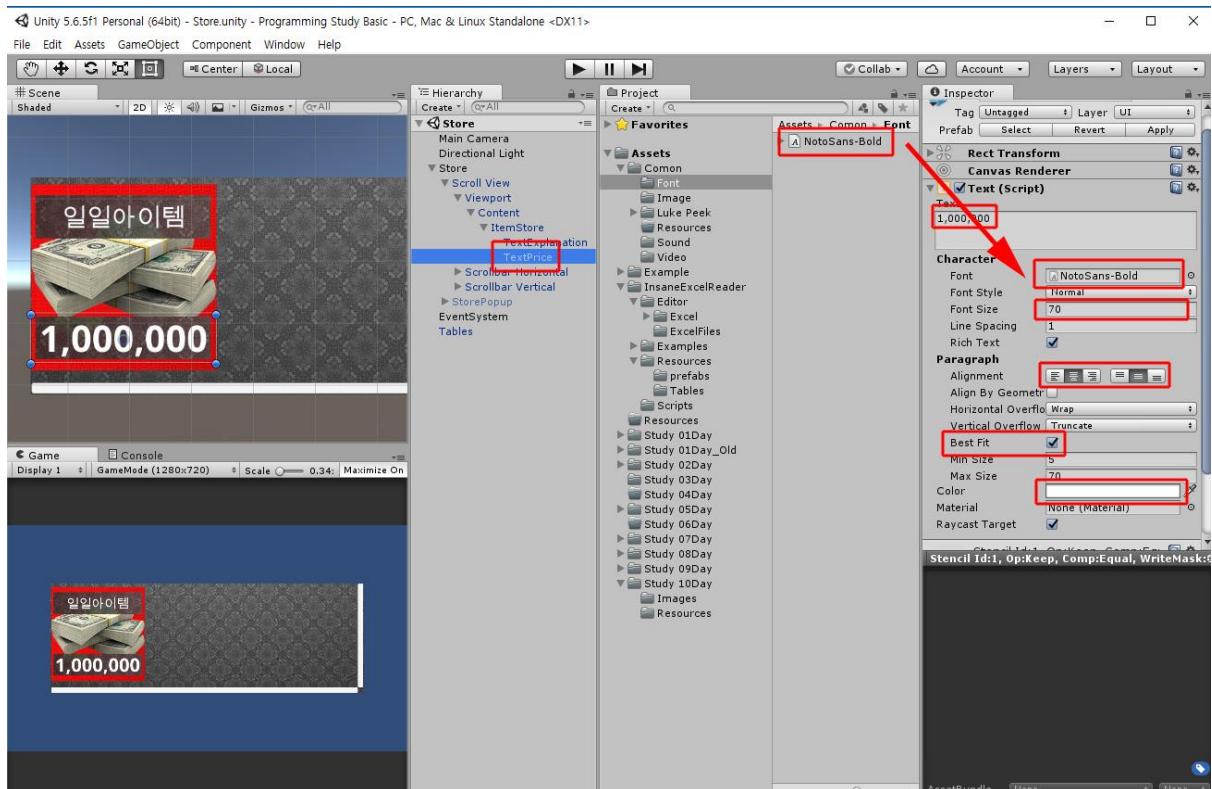
아이템 설명을 표시할 Text UI를 만들어 줍니다.



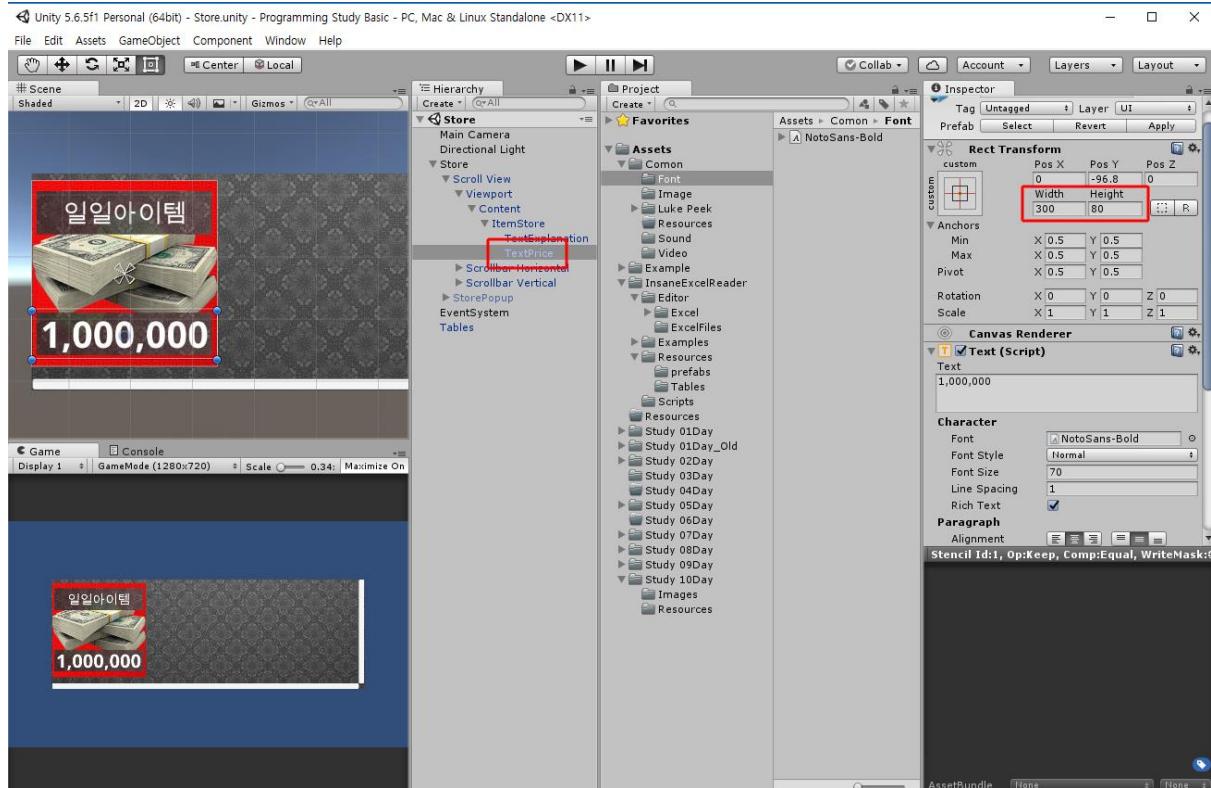
Font Size가 60이니 Rect Transform의 Height값을 70으로 조절해 줍니다.



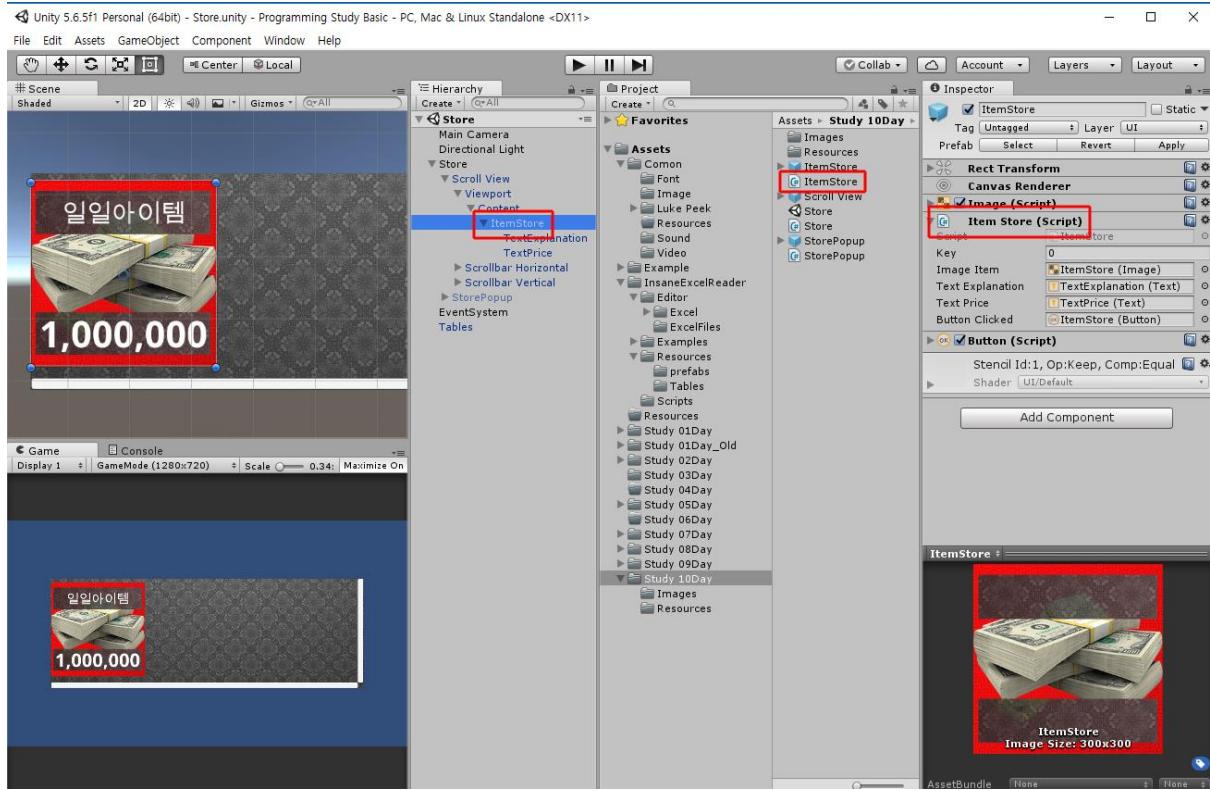
아이템의 가격을 표시할 Text UI를 만들어 줍니다.



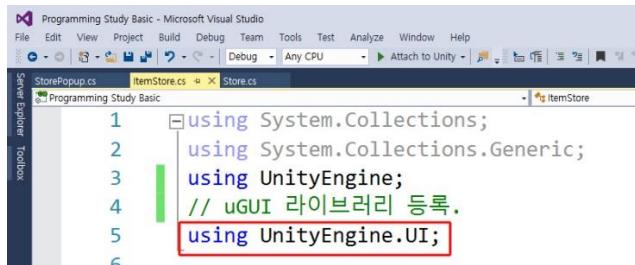
Font Size가 70이니 Rect Transform의 Height값을 80으로 설정해 줍니다.



ItemStore라는 이름의 스크립트를 만들어 주시고, ItemStore오브젝트에 컴포넌트로 등록해 줍니다.

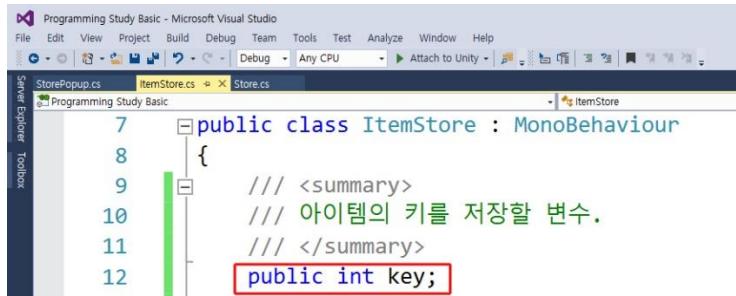


ItemStore스크립트를 오픈하고 코드를 추가해 줍니다. 가장 먼저 해 주어야 하는 일은 uGUI 라이브러리를 등록해 주는 일입니다.



상점의 아이템들의 정보는 Store테이블에서 가져온다고 했습니다.

또한 Dictionary타입으로 가져온다고 했습니다. 아이템들을 구분하기 위한 Key값을 저장할 변수를 만들어 줍니다.

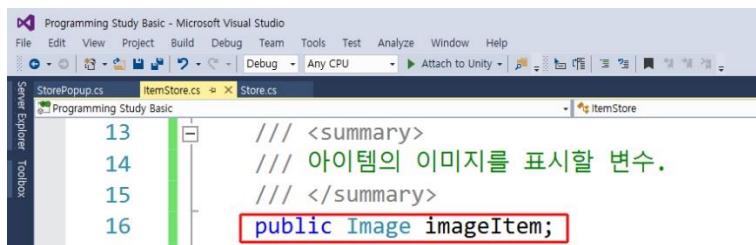


```

7  public class ItemStore : MonoBehaviour
8  {
9      /// <summary>
10     /// 아이템의 키를 저장할 변수.
11     /// </summary>
12     public int key;

```

아이템의 이미지를 표시할 UI에 접근할 변수를 만들어 줍니다.

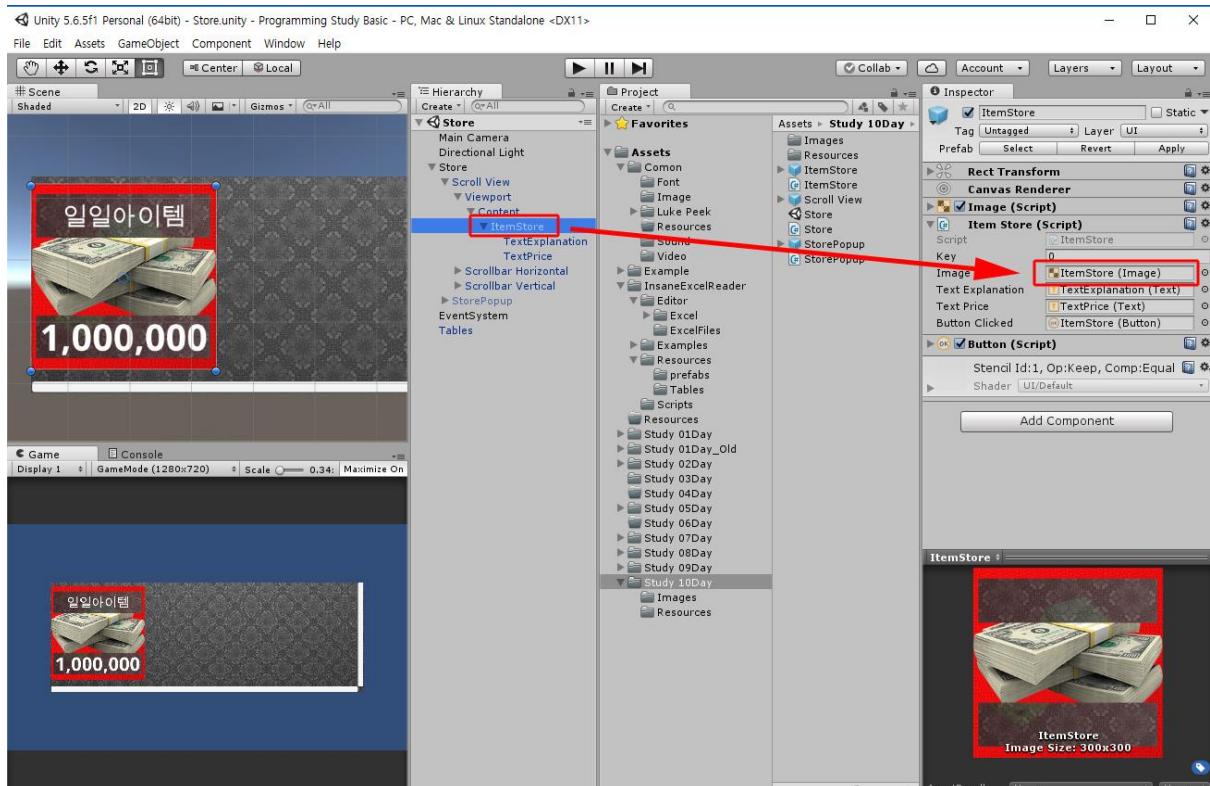


```

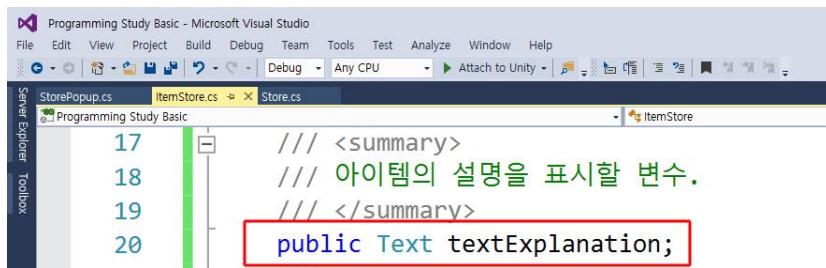
13     /// <summary>
14     /// 아이템의 이미지를 표시할 변수.
15     /// </summary>
16     public Image imageItem;

```

인스펙터에서 적용해 줍니다.

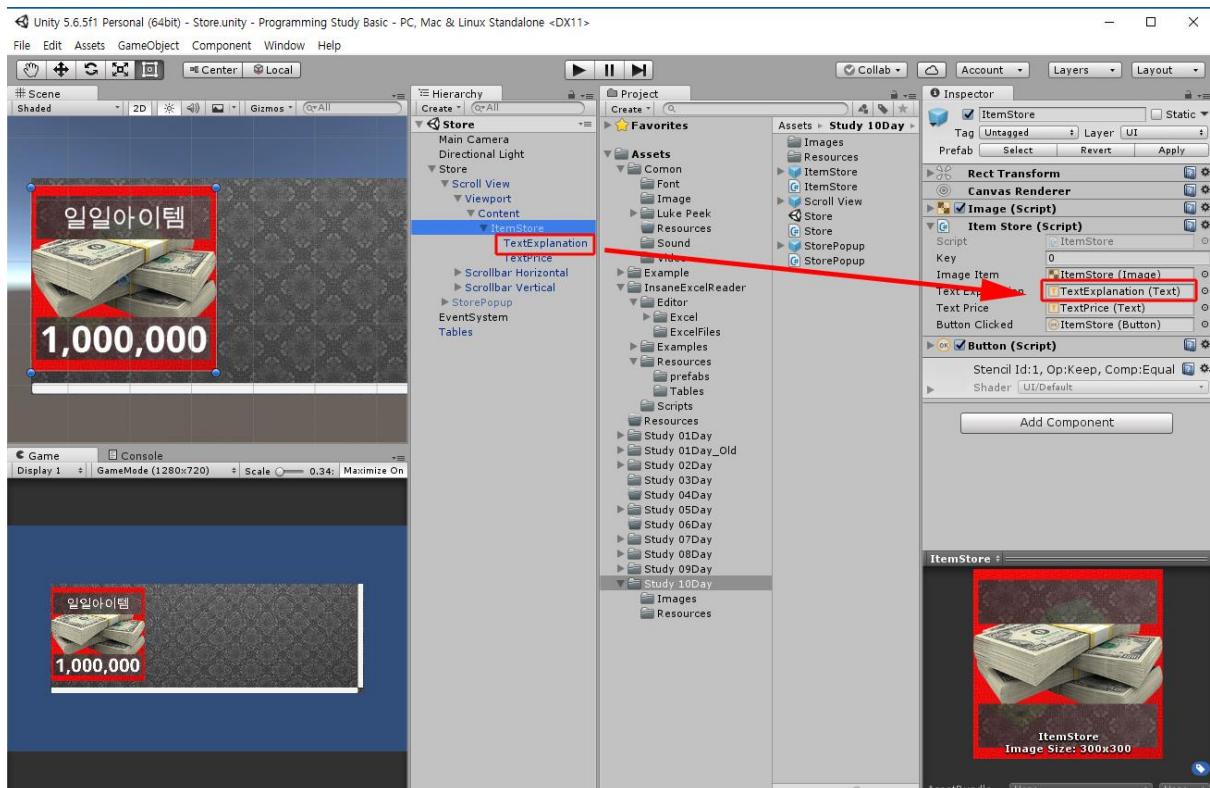


아이템의 설명을 표시할 UI에 접근할 변수를 만들어 줍니다.

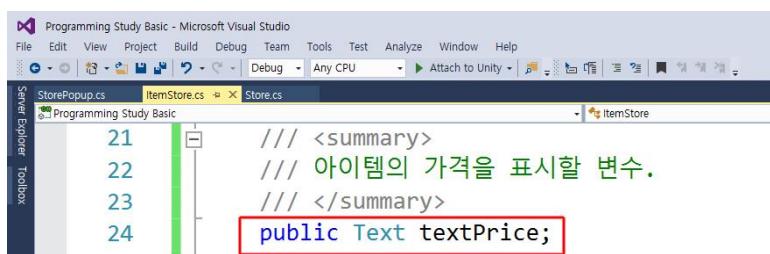


```
Programming Study Basic - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
File Edit View Project Build Debug Team Tools Test Analyze Window Help
StorePopup.cs ItemStore.cs Store.cs
Programming Study Basic
17     /// <summary>
18     /// 아이템의 설명을 표시할 변수.
19     /// </summary>
20     public Text textExplanation;
```

인스펙터에서 적용해 줍니다.

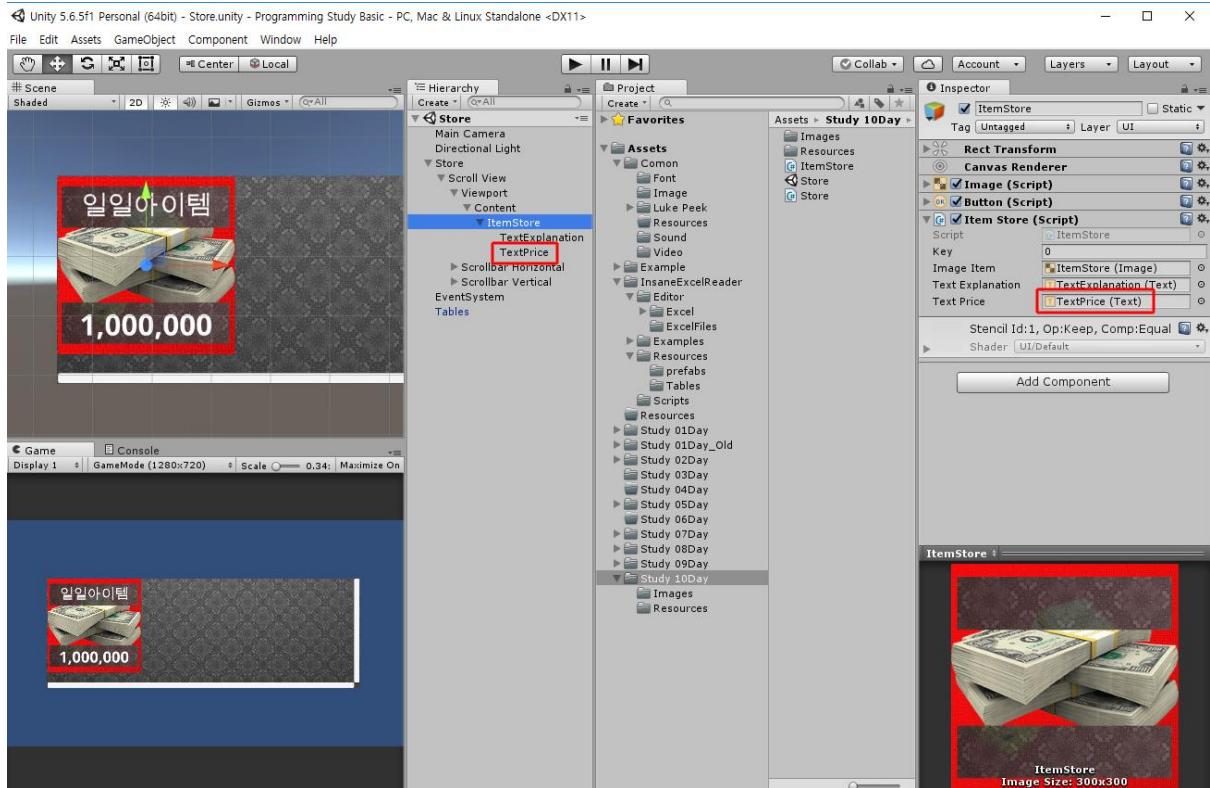


아이템의 가격을 표시할 UI에 접근할 수 있는 변수를 만들어 줍니다.

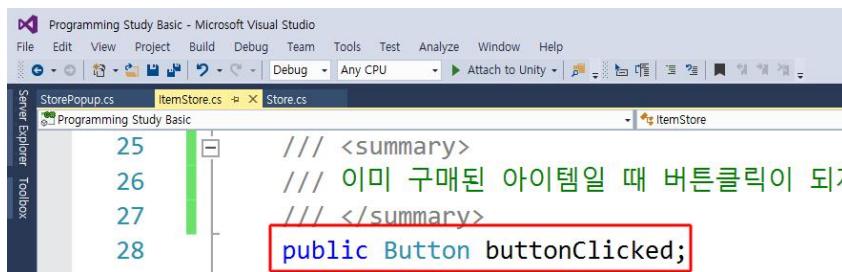


```
Programming Study Basic - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
File Edit View Project Build Debug Team Tools Test Analyze Window Help
StorePopup.cs ItemStore.cs Store.cs
Programming Study Basic
21     /// <summary>
22     /// 아이템의 가격을 표시할 변수.
23     /// </summary>
24     public Text textPrice;
```

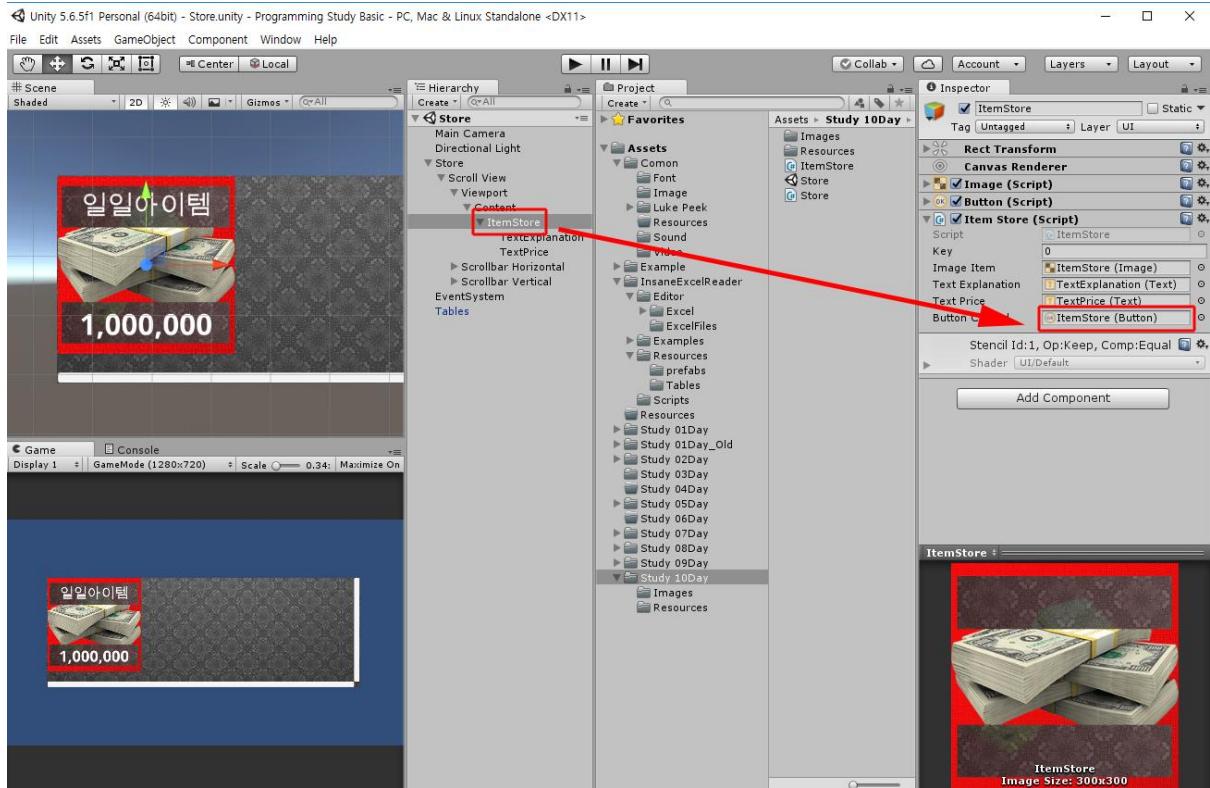
인스펙터에서 적용해 줍니다.



상점에서 아이템을 구매하면 추가로 구매가 되지 않도록 버튼 클릭을 막을 생각입니다. 따라서 버튼에 접근할 수 있는 변수를 만들어 줍니다.

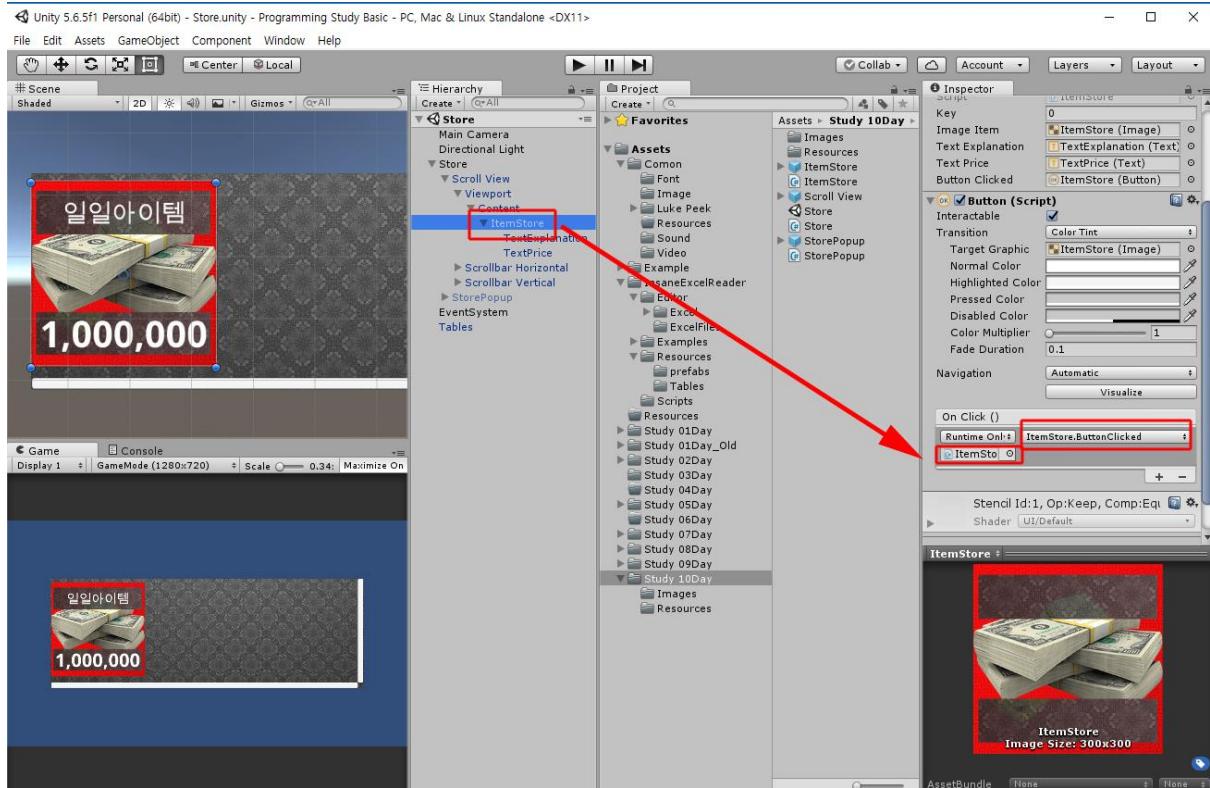


인스펙터에서 적용해 줍니다.



필요한 변수를 모두 만들었으니 이제 함수를 만들어 봅니다. 아이템이 클릭되었을 때 호출되는 함수를 만들어 줍니다.





이제 아이템의 정보를 표시하는 함수를 만들어 줍니다.

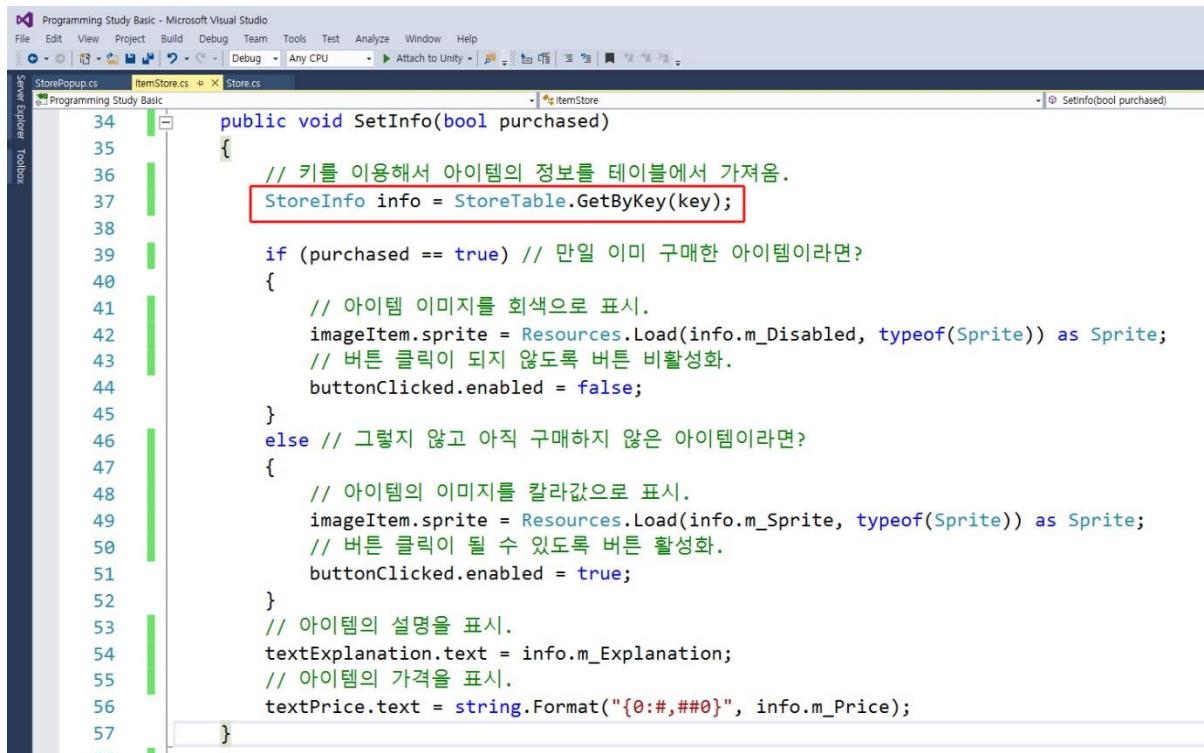
```

public void SetInfo(bool purchased)
{
    // 키를 이용해서 아이템의 정보를 테이블에서 가져옴.
    StoreInfo info = StoreTable.GetByKey(key);

    if (purchased == true) // 만일 이미 구매한 아이템이라면?
    {
        // 아이템 이미지를 회색으로 표시.
        imageItem.sprite = Resources.Load(info.m_Disabled, typeof(Sprite)) as Sprite;
        // 버튼 클릭이 되지 않도록 버튼 비활성화.
        buttonClicked.enabled = false;
    }
    else // 그렇지 않고 아직 구매하지 않은 아이템이라면?
    {
        // 아이템의 이미지를 칼라값으로 표시.
        imageItem.sprite = Resources.Load(info.m_Sprite, typeof(Sprite)) as Sprite;
        // 버튼 클릭이 될 수 있도록 버튼 활성화.
        buttonClicked.enabled = true;
    }
    // 아이템의 설명을 표시.
    textExplanation.text = info.m_Explanation;
    // 아이템의 가격을 표시.
    textPrice.text = string.Format("{0:#,##0}", info.m_Price);
}

```

Dictionary타입은 Key를 이용해서 정보 값을 가져올 수 있다고 했습니다. Key를 이용해서 StoreInfo정보를 가져옵니다.

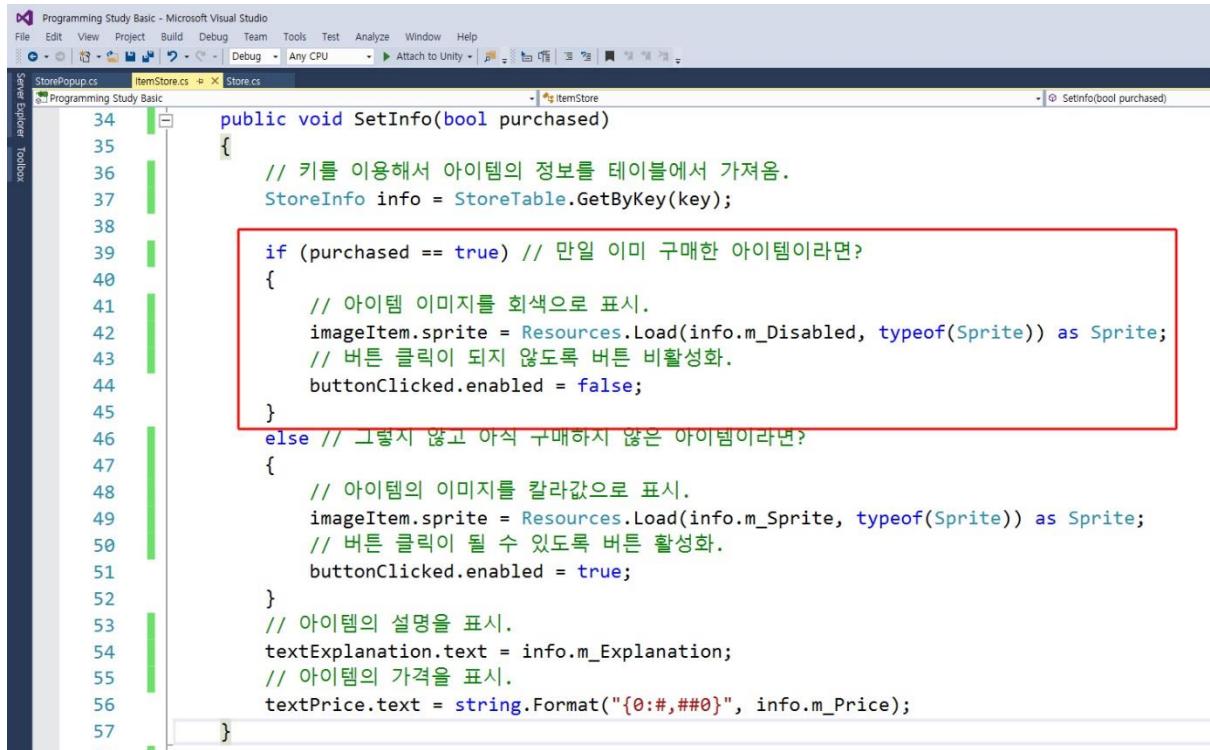


The screenshot shows the Microsoft Visual Studio IDE with the 'Programming Study Basic' project open. The 'Store.cs' file is the active code editor. The code is written in C# and defines a method 'SetInfo(bool purchased)'. A specific line of code, 'StoreInfo info = StoreTable.GetByKey(key);', is highlighted with a red box. A green vertical bar on the left margin indicates the current line of code. The code logic handles setting the item's state based on whether it is purchased or not, including loading the correct sprite and enabling/disabling the button.

```
34     public void SetInfo(bool purchased)
35     {
36         // 키를 이용해서 아이템의 정보를 테이블에서 가져옴.
37         StoreInfo info = StoreTable.GetByKey(key);
38
39         if (purchased == true) // 만일 이미 구매한 아이템이라면?
40         {
41             // 아이템 이미지를 회색으로 표시.
42             imageItem.sprite = Resources.Load(info.m_Disabled, typeof(Sprite)) as Sprite;
43             // 버튼 클릭이 되지 않도록 버튼 비활성화.
44             buttonClicked.enabled = false;
45         }
46         else // 그렇지 않고 아직 구매하지 않은 아이템이라면?
47         {
48             // 아이템의 이미지를 칼라값으로 표시.
49             imageItem.sprite = Resources.Load(info.m_Sprite, typeof(Sprite)) as Sprite;
50             // 버튼 클릭이 될 수 있도록 버튼 활성화.
51             buttonClicked.enabled = true;
52         }
53         // 아이템의 설명을 표시.
54         textExplanation.text = info.m_Explanation;
55         // 아이템의 가격을 표시.
56         textPrice.text = string.Format("{0:#,##0}", info.m_Price);
57     }

```

만일 구매한 아이템이라고 한다면 아이템 이미지를 회색으로 표시할 것입니다. 그리고 버튼 컴포넌트를 비활성화 시켜서 버튼이 클릭 되지 않도록 할 것입니다.



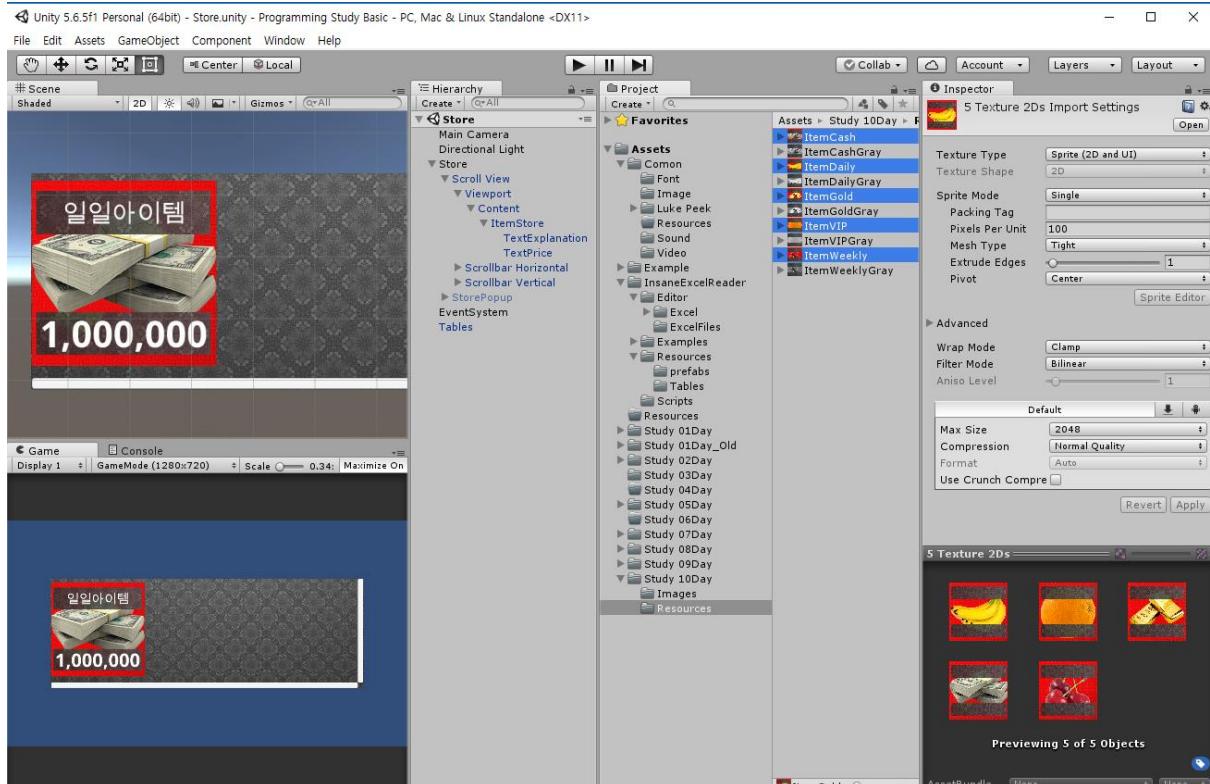
```

public void SetInfo(bool purchased)
{
    // 키를 이용해서 아이템의 정보를 테이블에서 가져옴.
    StoreInfo info = StoreTable.GetByKey(key);

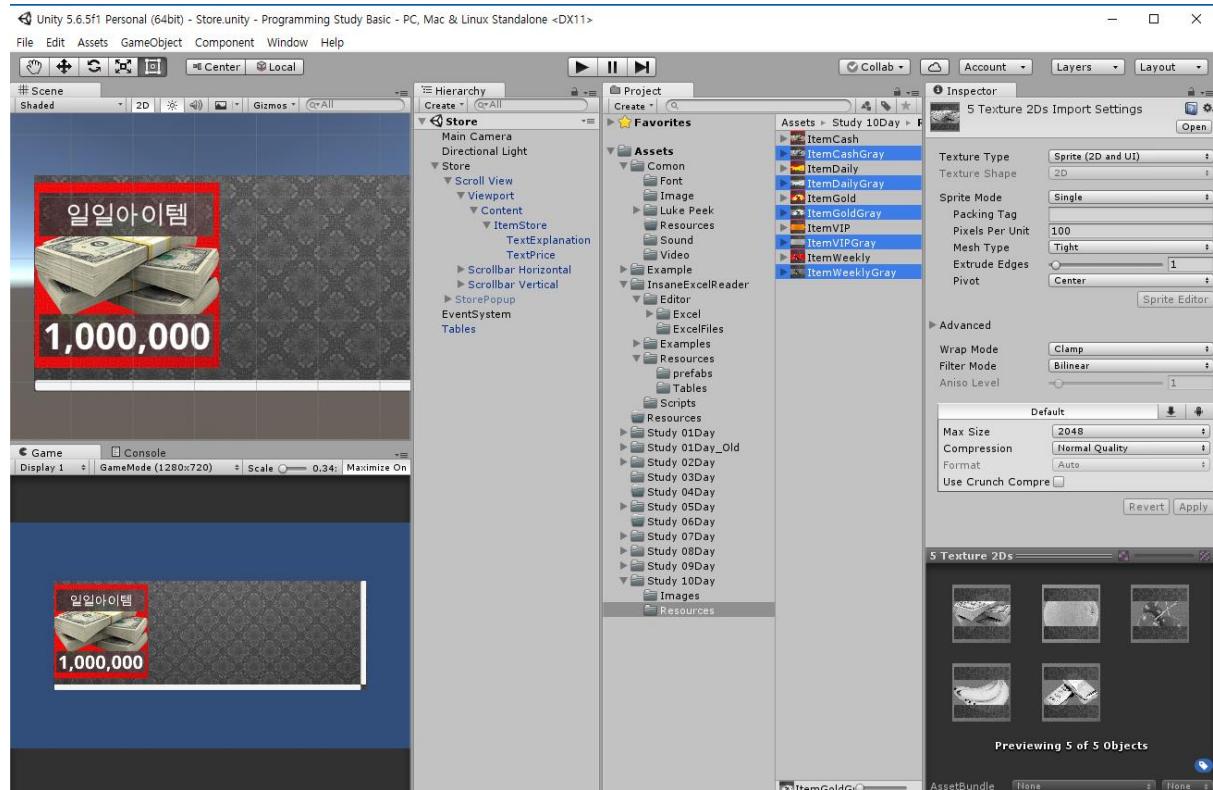
    if (purchased == true) // 만일 이미 구매한 아이템이라면?
    {
        // 아이템 이미지를 회색으로 표시.
        imageItem.sprite = Resources.Load(info.m_Disabled, typeof(Sprite)) as Sprite;
        // 버튼 클릭이 되지 않도록 버튼 비활성화.
        buttonClicked.enabled = false;
    }
    else // 그렇지 않고 아직 구매하지 않은 아이템이라면?
    {
        // 아이템의 이미지를 칼라값으로 표시.
        imageItem.sprite = Resources.Load(info.m_Sprite, typeof(Sprite)) as Sprite;
        // 버튼 클릭이 될 수 있도록 버튼 활성화.
        buttonClicked.enabled = true;
    }
    // 아이템의 설명을 표시.
    textExplanation.text = info.m_Explanation;
    // 아이템의 가격을 표시.
    textPrice.text = string.Format("{0:#,##0}", info.m_Price);
}

```

아이템의 칼라이미지가 준비되어 있습니다.



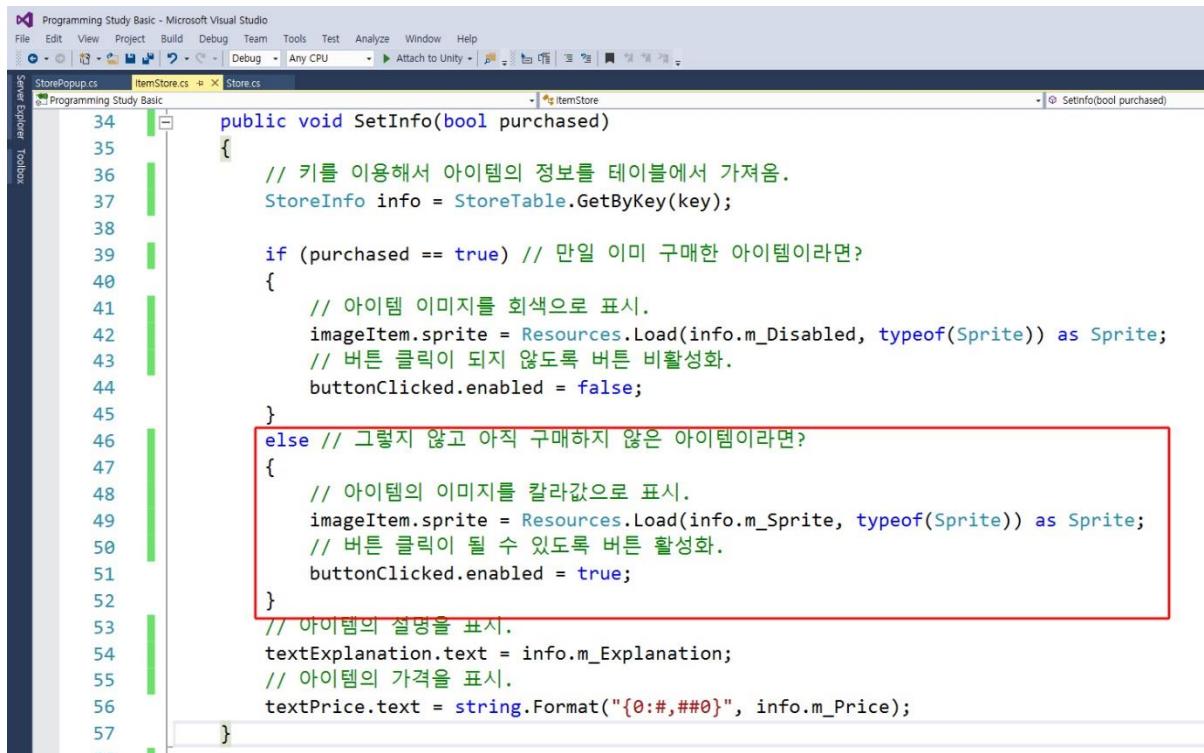
아이템의 회색이미지도 준비되어 있습니다.



이 정보 값이 Store테이블에 정의 되어 있습니다.

A	B	C	D	E
Key	Explanation	Sprite	Price	Disabled
10000	일일아이템	ItemDaily	100	ItemDailyGray
10001	주간아이템	ItemWeekly	1000	ItemWeeklyGray
10002	VIP아이템	ItemVIP	10000	ItemVIPGray
10003	게임머니	ItemGold	100000	ItemGoldGray
10004	캐쉬	ItemCash	1000000	ItemCashGray

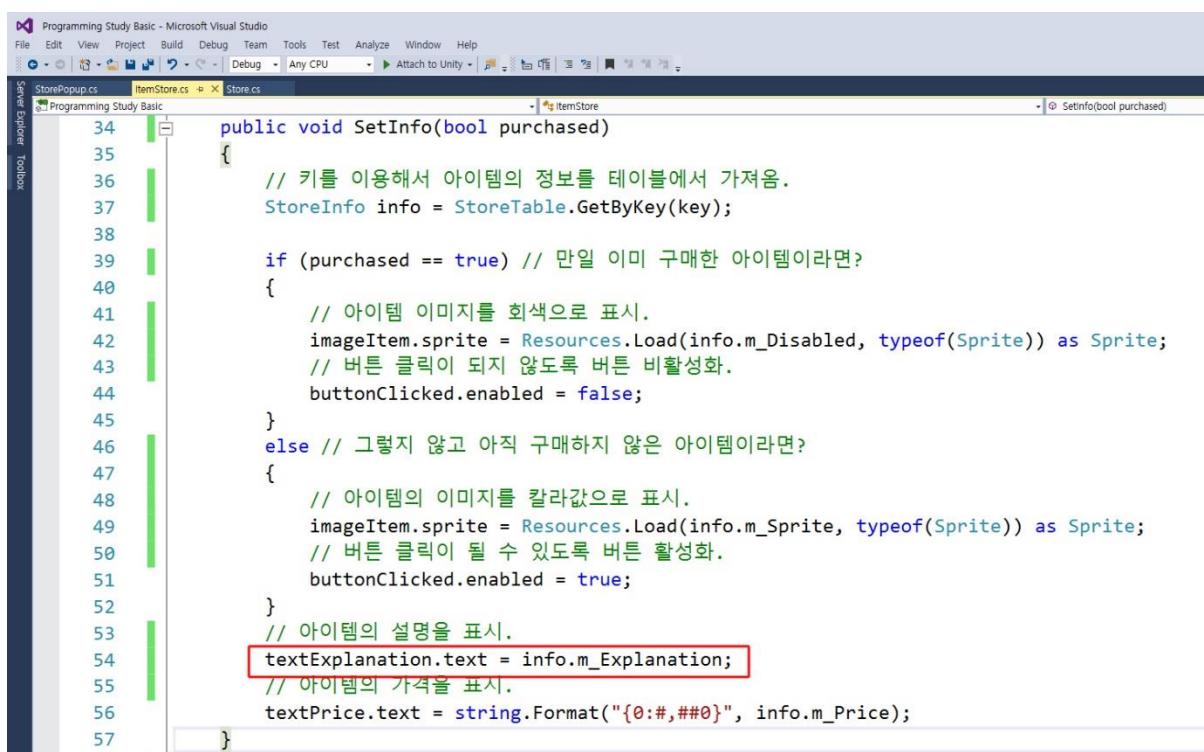
아직 구매한 아이템이 아니면 이미지를 칼라로 표시해 주고 버튼 클릭이 될 수 있도록 버튼 컴포넌트를 활성화 해줍니다.



```
public void SetInfo(bool purchased)
{
    // 키를 이용해서 아이템의 정보를 테이블에서 가져옴.
    StoreInfo info = StoreTable.GetByKey(key);

    if (purchased == true) // 만일 이미 구매한 아이템이라면?
    {
        // 아이템 이미지를 회색으로 표시.
        imageItem.sprite = Resources.Load(info.m_Disabled, typeof(Sprite)) as Sprite;
        // 버튼 클릭이 되지 않도록 버튼 비활성화.
        buttonClicked.enabled = false;
    }
    else // 그렇지 않고 아직 구매하지 않은 아이템이라면?
    {
        // 아이템의 이미지를 칼라값으로 표시.
        imageItem.sprite = Resources.Load(info.m_Sprite, typeof(Sprite)) as Sprite;
        // 버튼 클릭이 될 수 있도록 버튼 활성화.
        buttonClicked.enabled = true;
    }
    // 아이템의 설명을 표시.
    textExplanation.text = info.m_Explanation;
    // 아이템의 가격을 표시.
    textPrice.text = string.Format("{0:#,##0}", info.m_Price);
}
```

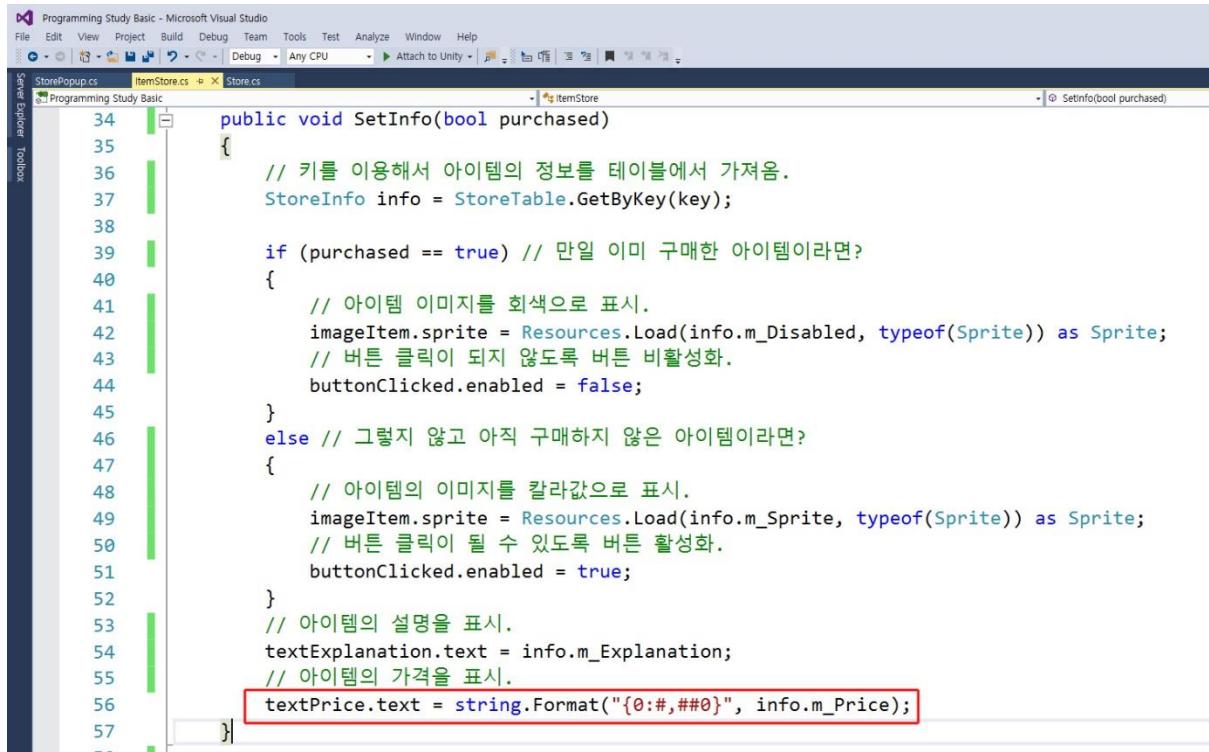
이제 아이템의 설명을 표시해 줍니다.



```
public void SetInfo(bool purchased)
{
    // 키를 이용해서 아이템의 정보를 테이블에서 가져옴.
    StoreInfo info = StoreTable.GetByKey(key);

    if (purchased == true) // 만일 이미 구매한 아이템이라면?
    {
        // 아이템 이미지를 회색으로 표시.
        imageItem.sprite = Resources.Load(info.m_Disabled, typeof(Sprite)) as Sprite;
        // 버튼 클릭이 되지 않도록 버튼 비활성화.
        buttonClicked.enabled = false;
    }
    else // 그렇지 않고 아직 구매하지 않은 아이템이라면?
    {
        // 아이템의 이미지를 칼라값으로 표시.
        imageItem.sprite = Resources.Load(info.m_Sprite, typeof(Sprite)) as Sprite;
        // 버튼 클릭이 될 수 있도록 버튼 활성화.
        buttonClicked.enabled = true;
    }
    // 아이템의 설명을 표시.
    textExplanation.text = info.m_Explanation;
    // 아이템의 가격을 표시.
    textPrice.text = string.Format("{0:#,##0}", info.m_Price);
}
```

마지막으로 아이템의 가격을 표시해 줍니다.

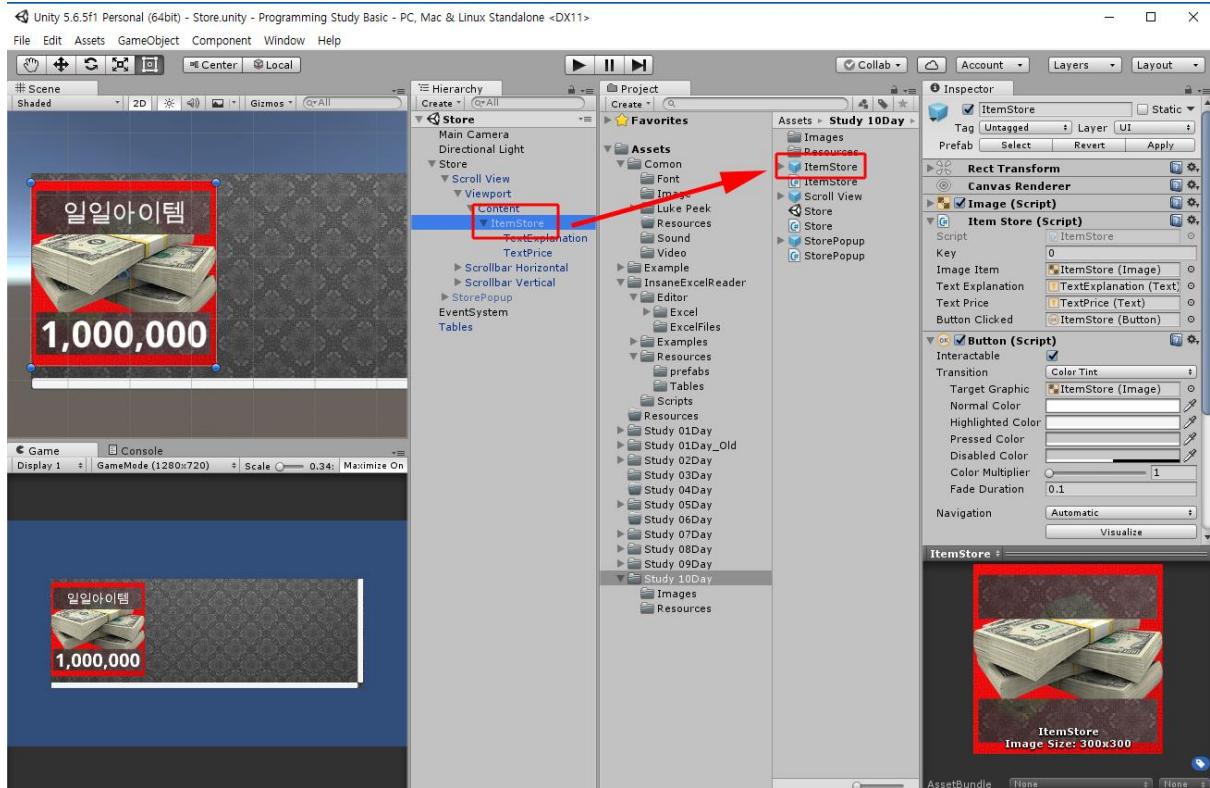


```
File Edit View Project Build Debug Team Tools Test Analyze Window Help
File Explorer Tools
StorePopup.cs ItemStore.cs Store.cs
Programming Study Basic
34     public void SetInfo(bool purchased)
35     {
36         // 키를 이용해서 아이템의 정보를 테이블에서 가져옴.
37         StoreInfo info = StoreTable.GetByKey(key);
38
39         if (purchased == true) // 만일 이미 구매한 아이템이라면?
40         {
41             // 아이템 이미지를 회색으로 표시.
42             imageItem.sprite = Resources.Load(info.m_Disabled, typeof(Sprite)) as Sprite;
43             // 버튼 클릭이 되지 않도록 버튼 비활성화.
44             buttonClicked.enabled = false;
45         }
46         else // 그렇지 않고 아직 구매하지 않은 아이템이라면?
47         {
48             // 아이템의 이미지를 칼라값으로 표시.
49             imageItem.sprite = Resources.Load(info.m_Sprite, typeof(Sprite)) as Sprite;
50             // 버튼 클릭이 될 수 있도록 버튼 활성화.
51             buttonClicked.enabled = true;
52         }
53         // 아이템의 설명을 표시.
54         textExplanation.text = info.m_Explanation;
55         // 아이템의 가격을 표시.
56         textPrice.text = string.Format("{0:#,##0}", info.m_Price);
57     }
}
```

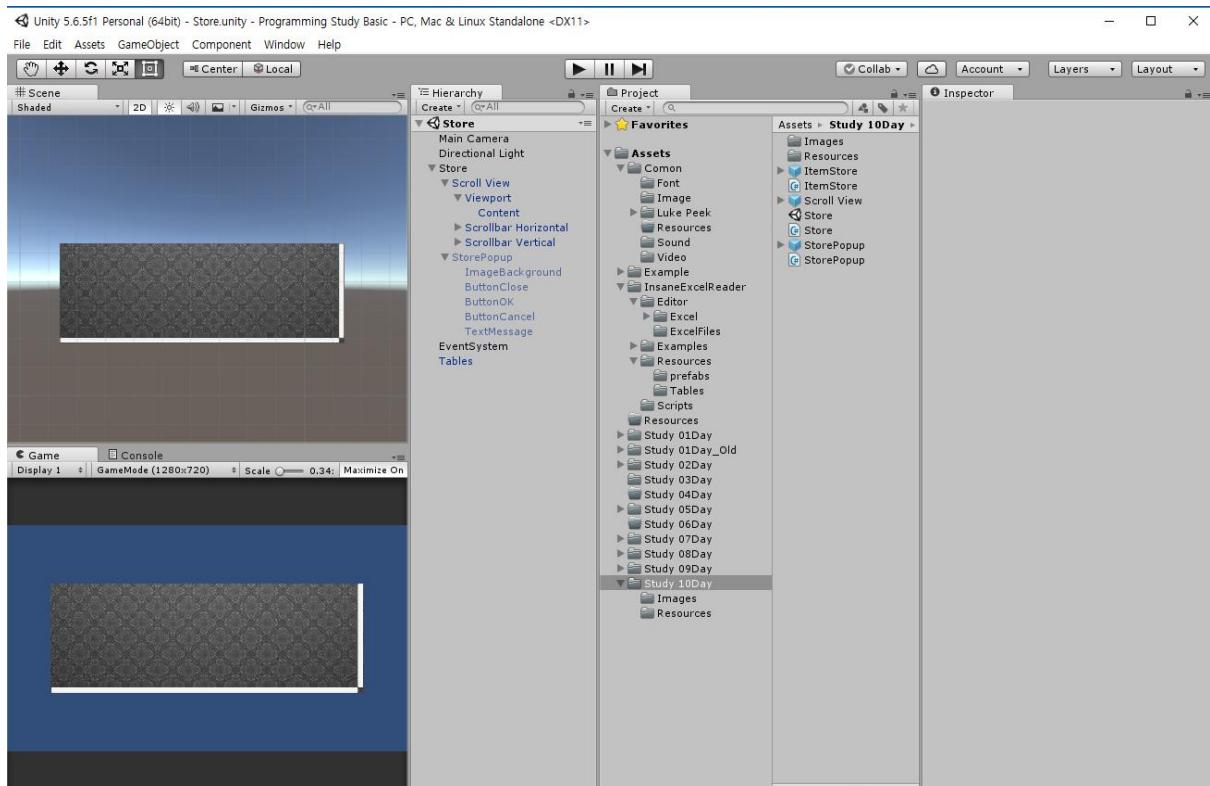
string.Format() 문이 종전과 달라 보이는데요. 요 포맷을 쓰면

10000000이 1,000,000으로 표시됩니다. 가격을 표시할 때 보통 이런 방식을 쓰죠.

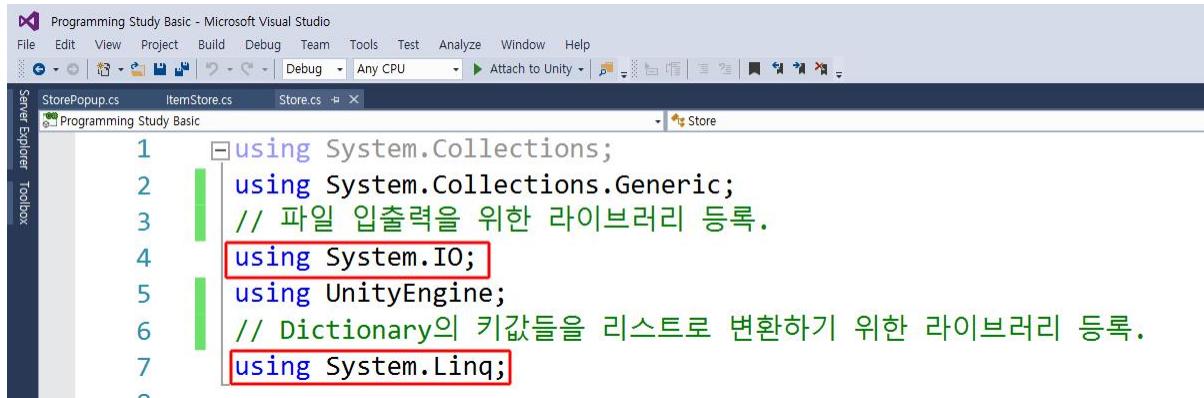
아이템은 실시간으로 생성할 것이기 때문에 프리팹으로 만들어 줍니다.



ItemStore프리팹도 실시간으로 생성할 것이니 씬에서 지워줍니다.

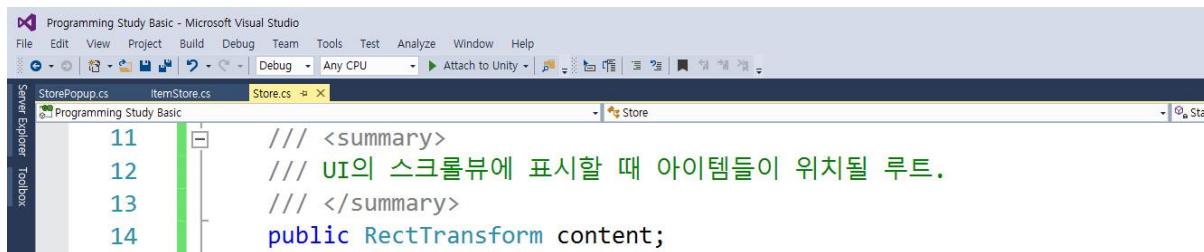


이제 본격적으로 Store스크립트 작업을 해 보도록 합니다. 미리 필요한 라이브러리를 등록해 줍니다.



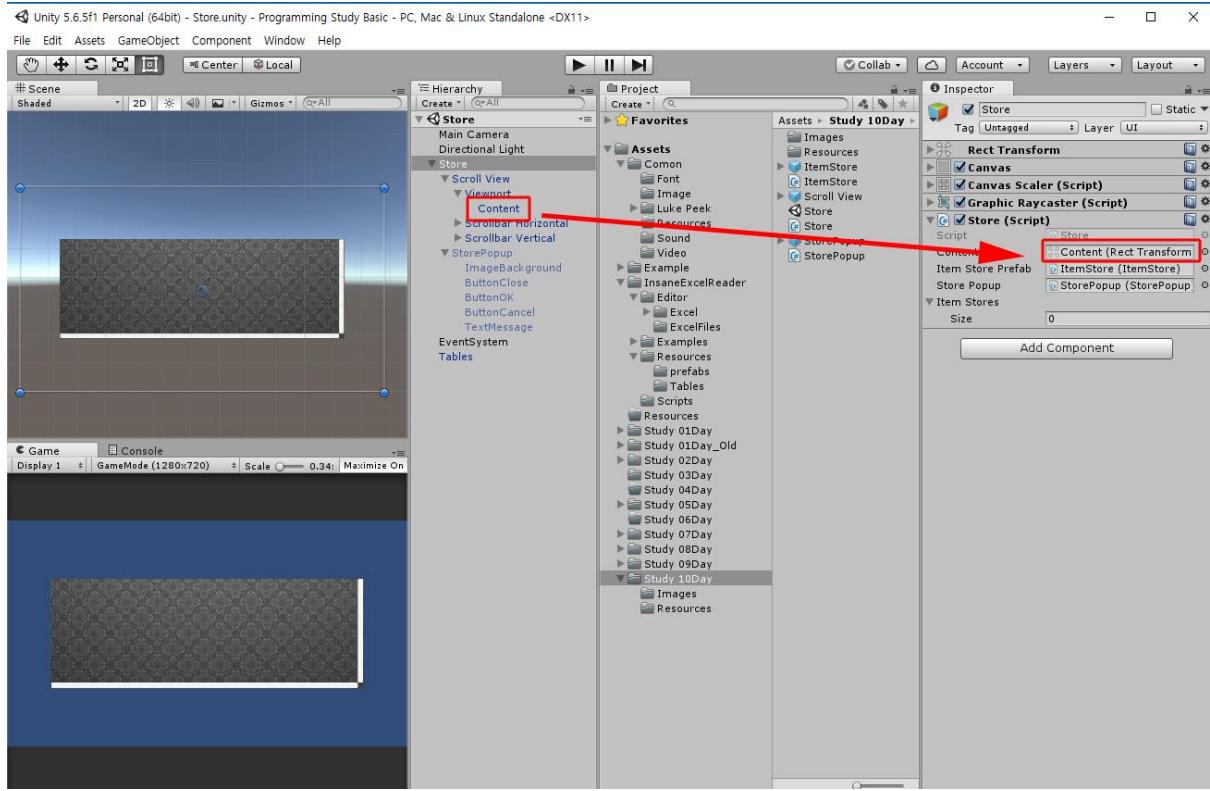
```
1 using System.Collections;
2 using System.Collections.Generic;
3 // 파일 입출력을 위한 라이브러리 등록.
4 using System.IO;
5 using UnityEngine;
6 // Dictionary의 키값들을 리스트로 변환하기 위한 라이브러리 등록.
7 using System.Linq;
```

필요한 변수를 만들어 보도록 합니다. 스크롤뷰의 아이템들은 Content오브젝트 아래에 위치해야 한다고 했습니다. Content오브젝트에 접근할 수 있는 변수를 만들어 줍니다.

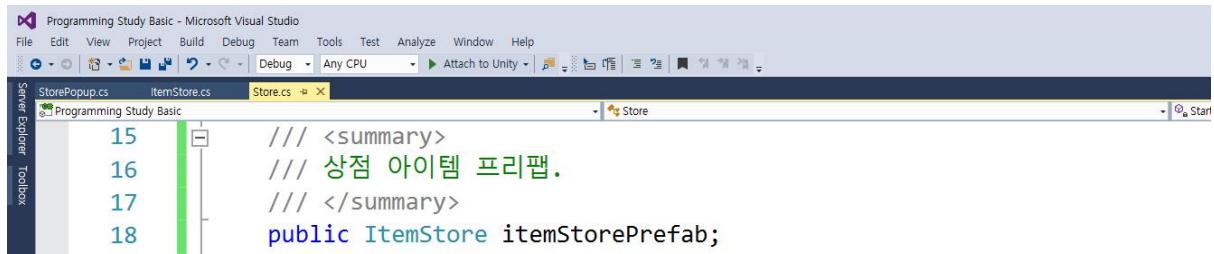


```
11 /// <summary>
12 /// UI의 스크롤뷰에 표시할 때 아이템들이 위치될 루트.
13 /// </summary>
14 public RectTransform content;
```

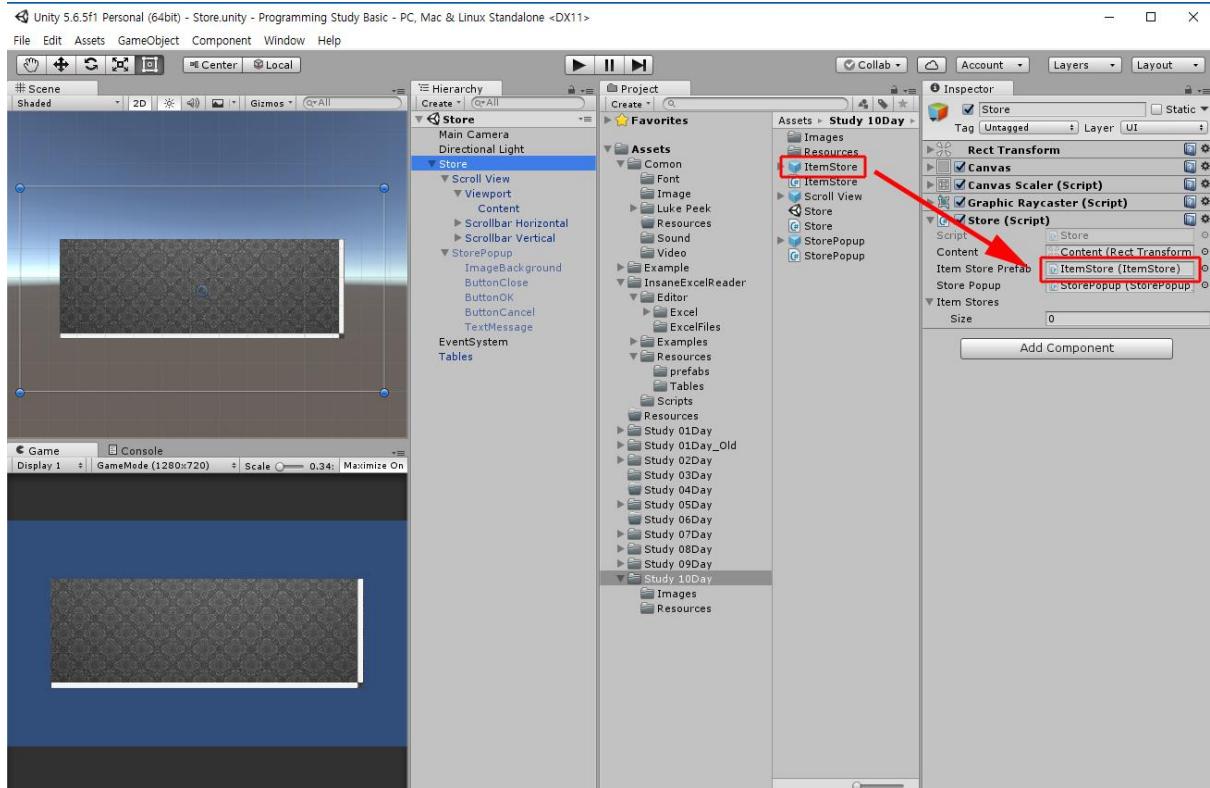
인스펙터에서 적용해 줍니다.



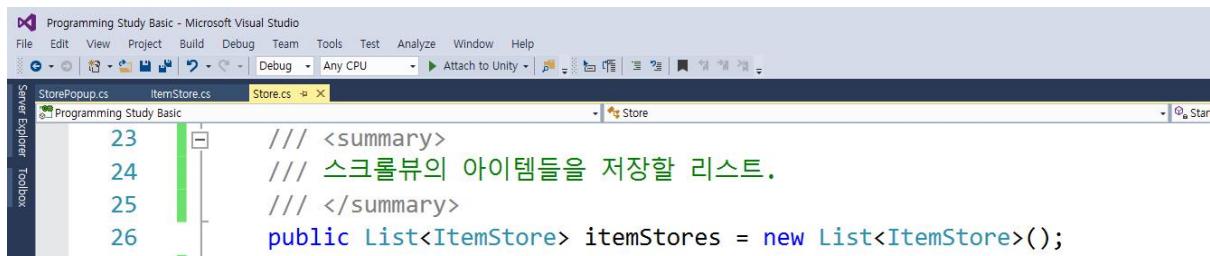
상점 아이템을 실시간으로 생성한다고 했습니다. 상점 아이템을 저장해 둘 변수를 만들어 줍니다.



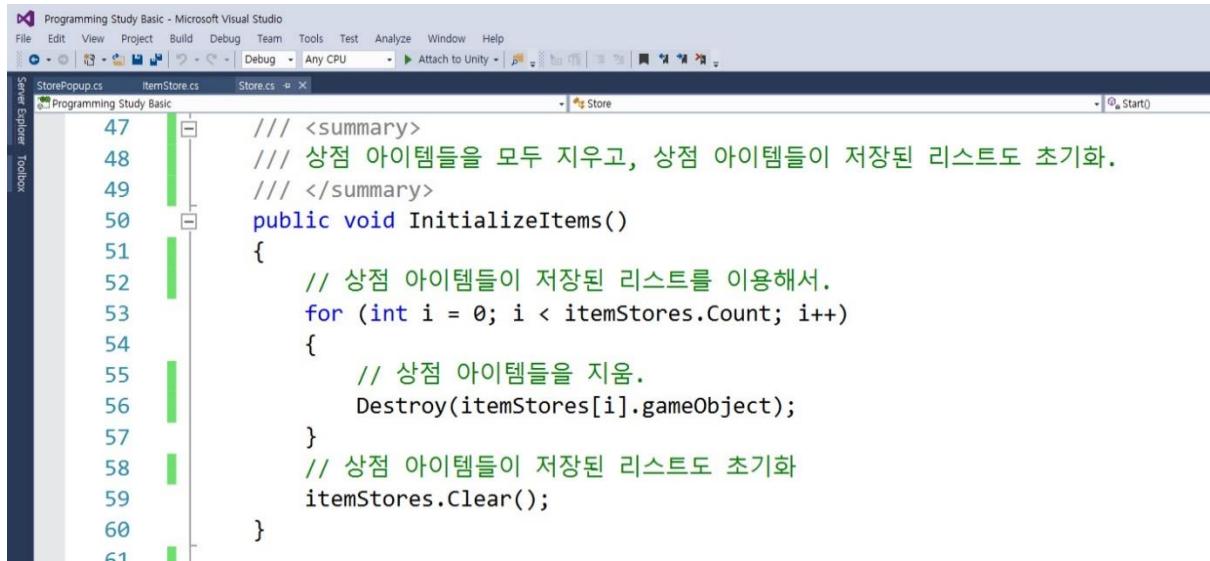
인스펙터에서 적용해 줍니다.



상점 아이템들이 스크롤뷰에 표시 되었을 때, 상점아이템들을 담을 변수가 필요합니다.

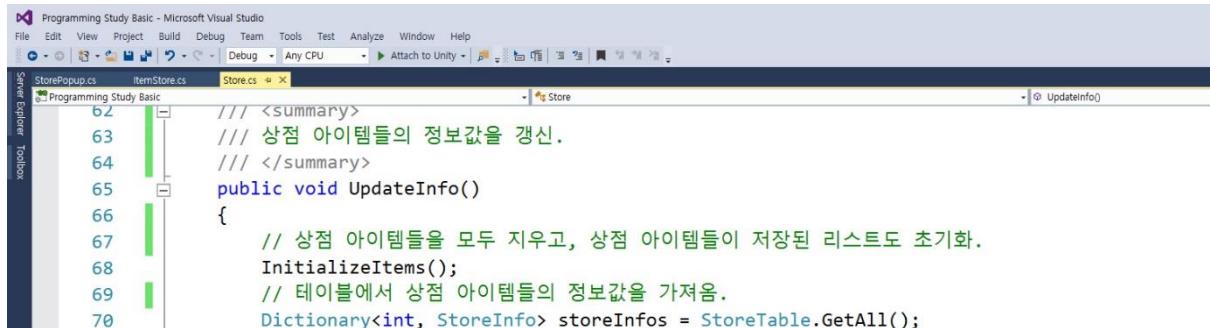


필요한 변수가 모두 만들어졌습니다. 이제 필요한 함수를 만들어 봅니다. 가장 먼저 만들 함수는 InitializeItems() 함수입니다. 스크롤뷰의 아이템들이 갱신되었을 때 스크롤뷰에 아이템들이 있다면 지워주고, 다시 갱신된 아이템들도 생성해 줘야 하기 때문입니다.

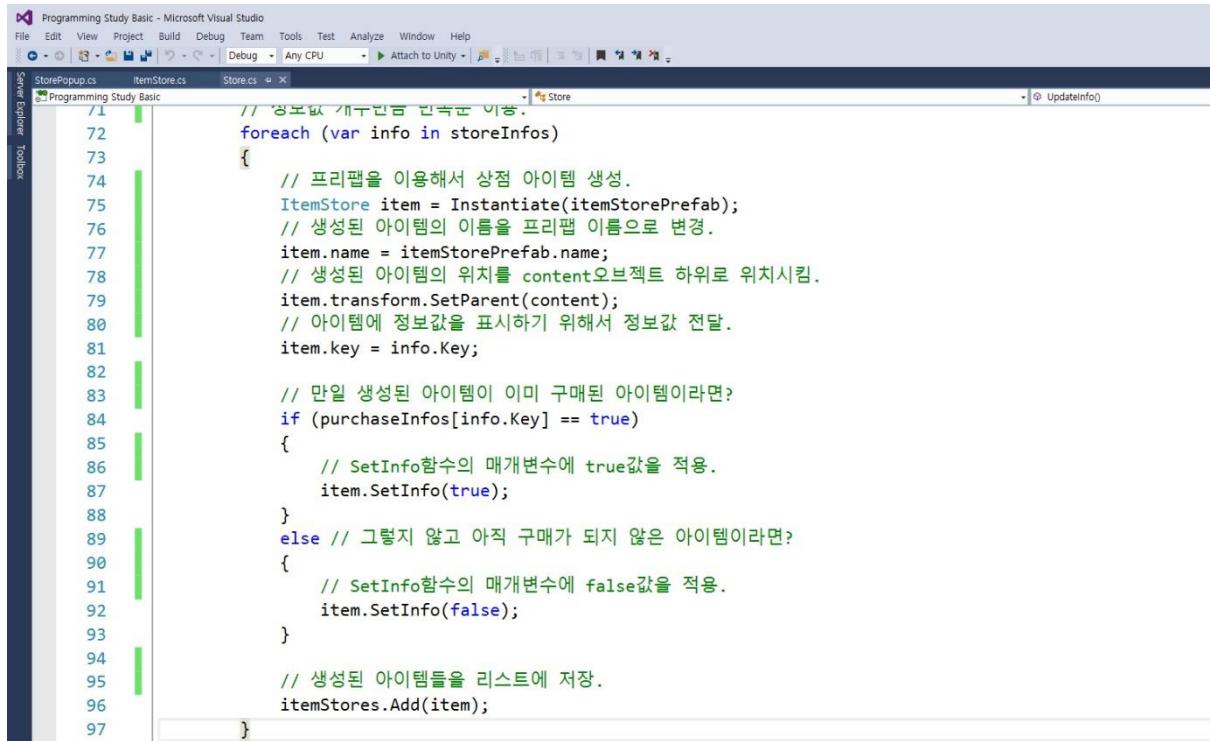


```
47     /// <summary>
48     /// 상점 아이템들을 모두 지우고, 상점 아이템들이 저장된 리스트도 초기화.
49     /// </summary>
50     public void InitializeItems()
51     {
52         // 상점 아이템들이 저장된 리스트를 이용해서.
53         for (int i = 0; i < itemStores.Count; i++)
54         {
55             // 상점 아이템들을 지움.
56             Destroy(itemStores[i].gameObject);
57         }
58         // 상점 아이템들이 저장된 리스트도 초기화
59         itemStores.Clear();
60     }
61 
```

아이템들을 생성하고, 상점 아이템들의 정보값을 업데이트해주는 함수를 만들어 줍니다. `UpdateInfo()` 함수입니다.



```
62     /// <summary>
63     /// 상점 아이템들의 정보값을 갱신.
64     /// </summary>
65     public void UpdateInfo()
66     {
67         // 상점 아이템들을 모두 지우고, 상점 아이템들이 저장된 리스트도 초기화.
68         InitializeItems();
69         // 테이블에서 상점 아이템들의 정보값을 가져옴.
70         Dictionary<int, StoreInfo> storeInfos = StoreTable.GetAll(); 
```

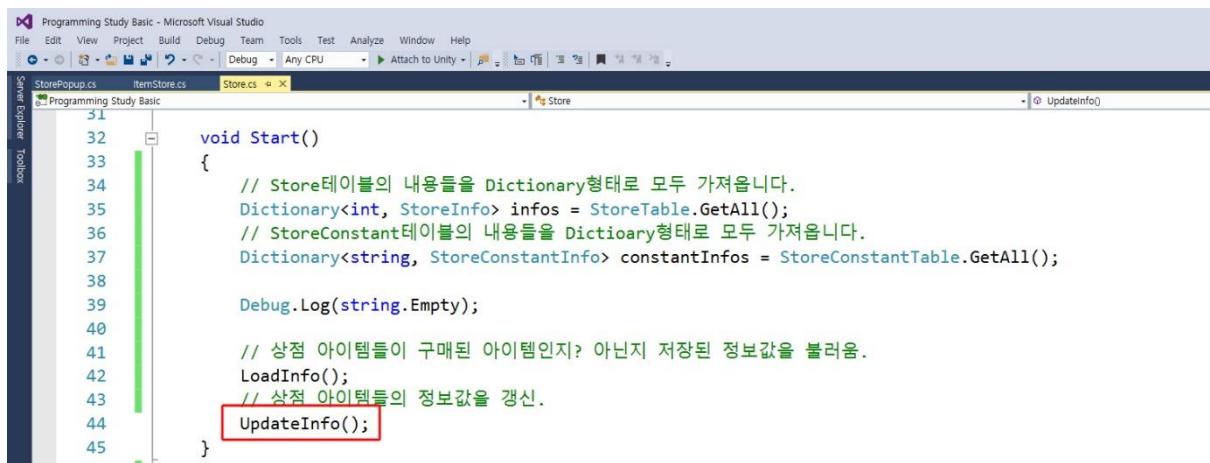


```
    // 공포값 개수입력 딱지로 이름.
    foreach (var info in storeInfos)
    {
        // 프리팹을 이용해서 상점 아이템 생성.
        ItemStore item = Instantiate(itemStorePrefab);
        // 생성된 아이템의 이름을 프리팹 이름으로 변경.
        item.name = itemStorePrefab.name;
        // 생성된 아이템의 위치를 content오브젝트 하위로 위치시킴.
        item.transform.SetParent(content);
        // 아이템에 정보값을 표시하기 위해서 정보값 전달.
        item.key = info.Key;

        // 만일 생성된 아이템이 이미 구매된 아이템이라면?
        if (purchaseInfos[info.Key] == true)
        {
            // SetInfo함수의 매개변수에 true값을 적용.
            item.SetInfo(true);
        }
        else // 그렇지 않고 아직 구매가 되지 않은 아이템이라면?
        {
            // SetInfo함수의 매개변수에 false값을 적용.
            item.SetInfo(false);
        }

        // 생성된 아이템들을 리스트에 저장.
        itemStores.Add(item);
    }
}
```

마찬가지로 Start() 함수 안에 넣어 줍니다.



```
void Start()
{
    // Store테이블의 내용들을 Dictionary형태로 모두 가져옵니다.
    Dictionary<int, StoreInfo> infos = StoreTable.GetAll();
    // StoreConstant테이블의 내용들을 Dictionary형태로 모두 가져옵니다.
    Dictionary<string, StoreConstantInfo> constantInfos = StoreConstantTable.GetAll();

    Debug.Log(string.Empty);

    // 상점 아이템들이 구매된 아이템인지? 아닌지 저장된 정보값을 불러옴.
    LoadInfo();
    // 상점 아이템들의 정보값을 갱신.
    UpdateInfo();
}
```

이제 플레이를 하고 결과를 확인합니다. 제대로 Store테이블에 정보 값에 따라서 아이템들의 정보 값이 표시되는 것을 확인할 수 있습니다.

