デザインパターン_シングルトン

目次

１．デザインパターンとは

みなさんはデザインパターンと言うのは何か知っていますでしょうか？
もしくはデザインパターンは知らないけどシングルトンは知っている、という人もいるかもしれません。

ここではデザインパターンとは何か？どうして必要か、というお話しをしていきます。

１．デザインパターンとは

ゲームを作る中で、色々とプログラム上の問題にぶつかっていると思います！

「クラス間の依存関係を減らしたい」

「処理の一部を差し替え可能にしたい」

「ステータスを上手く管理していきたい」

これは多くの開発者が通ってきた道であり、それぞれ独自の解決方法をとっていました。

しかし、これでは解決方法の質にばらつきがあり、非効率となっています。

そこで解決方法を「共有」することで、よくある問題とその効果的な解決方法をまとめて発信していく活動が広がった！　⇨ デザインパターン

１．デザインパターンとは

デザイン ＝ 設計、パターン ＝ お手本

設計のお手本となる手法のこと！

オブジェクト指向の言語全般で使用することが出来る。(C++,Java,C#)

オブジェクト指向のメリット：生産性の高さ(早く作れる)

信頼性の高さ(正しく動く)

デザインパターンを覚えることで、効率よく、バグが発生しにくいプログラムの作成が可能。

授業の中では、まずはこれは便利！と感じて、実際に活用できそうなものを中心にやっていきたいと思います。

２．シングルトンとは

今回は、デザインパターンの中でも代表格であるシングルトンを実装してみる。

＜注意点＞

デザインパターンはあくまで設計のお手本を示すもの。
自分が知っているシングルトンとは違う！となるかもしれませんが、それは
それで良いです。
要点さえ押さえていれば、自分の使いやすいように作れば良いのが
デザインパターンです。
シングルトンは超便利！絶対使うべき！　ということでは無いです。。
ピッタリの用途だけシングルトンを使って、それ以外は他の方法を使った方が
良いことも。

２．シングルトンとは

＜シングルトンとは＞

あるクラスのインスタンスを1つだけに制限するためのデザインパターン。
チーム制作で何度もインスタンスされると困ってしまうクラス(設定管理、ログ出力、サーバーとの接続管理など)を制限するために使用されます。

＜特徴＞

・1つのインスタンスだけが存在
・インスタンスには「クラス名.Instance」のような形でアクセスできる

シングルトンを実装する上で知っておかないといけない知識

⇨ C#におけるGeneric(ジェネリック)

シングルトンの前に、まずはジェネリックについてお話ししていこうと思います！

３．ジェネリックとは

英語の意味でのgenericとは

「汎用の」(形容詞)

「後発品」(名詞)

全く同じ処理をするコードがあるが、変数の型のみが異なる。そんな時に使えるのがジェネリックです！！

ちなみに、C++でいうと「テンプレート」という機能のことです。

(微妙に異なりますが。。。)

ちなみにジェネリックという機能はみなさん、既に使っている機能になります。。。

```
xxx.GetComponent<Rigidbody>();
int list = new List<int>();
```

３．ジェネリックとは

実際に簡単な例で試してみましょう！以下の要件を満たすContainer<T>クラスを作ってみます！TestGeneric1.csのファイルを作成してください。

・型引数Tを受け取る
・T型のデータを保存するItemというprivateなプロパティ(変数)を持つ
・SetItemメソッドでT型のItemを設定する
・GetItemメソッドで値を取得する

```
//10を出力
Container<int> intContainer = new Container<int>();
intContainer.SetItem(10);
Debug.Log(intContainer.GetItem());

//テストを出力
Container<string> strContainer = new Container<string>();
strContainer.SetItem("テスト");
Debug.Log(strContainer.GetItem());
```

３．ジェネリックとは

＜問題＞

TestGeneric2.csを作成し、任意の型のデータをためる「MyCollection」クラスを作ってください。

・Addメソッドで、データを追加できること

・PrintAllメソッドで、全てのデータをDebug.Logに表示させること

・Countメソッドで、個数を戻り値で返すこと

```
// 整数のコレクション
MyCollection<int> numbers = new MyCollection<int>();
numbers.Add(10);
numbers.Add(20);
numbers.Add(30);

numbers.PrintAll(); // 10, 20, 30
Debug.Log("個数: " + numbers.Count()); // 3

// 文字列のコレクション
MyCollection<string> fruits = new MyCollection<string>();
fruits.Add("Apple");
fruits.Add("Banana");

fruits.PrintAll(); // Apple, Banana
Debug.Log("個数: " + fruits.Count()); // 2
```

３．ジェネリックとは

＜whereについて＞

以下のように、where接続子を使用することで特定のクラスを継承したクラスのみ対象のジェネリッククラスを使用することができる、という制限をかけることができます。

以下の例だと、MonoBehaviourクラスを継承したクラスのみTestAというクラスを使用することが出来る、という意味になります。

```
public class TestA<T> : MonoBehaviour where T : MonoBehaviour
```

また、以下のような書き方をすることで、Tは型の安全性が保証される。
そのためキャストを行うことが許される(C#の規則)

```
public class Singleton<T> : MonoBehaviour where T : Singleton<T>
```

４．シングルトン実装

それでは話を戻して、今度はシングルトンの実装をしていきたいと思います。
以下の実装例に合わせて作成してみてください。

＜実装例＞

・GameManager.cs内のUpdateでScoreをインクリメント

・UIText.csでGameManagerを取得し、Scoreを取得して画面に表示

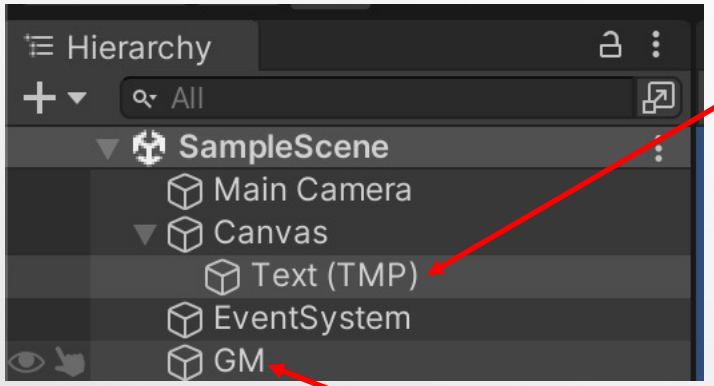
上記の実装例をシングルトンで実装してみるとどのようなコードになるか
確認するため、以下のコードを作成してください。

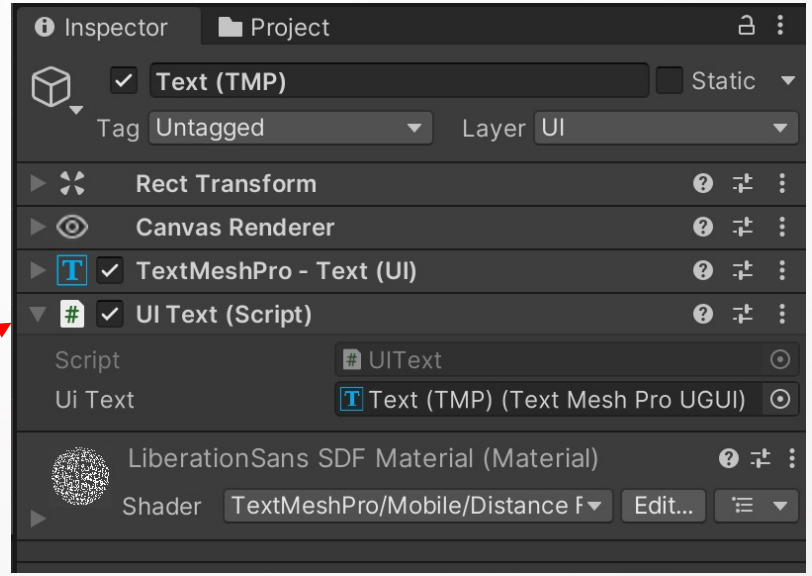
GameManager.cs / UIText.cs

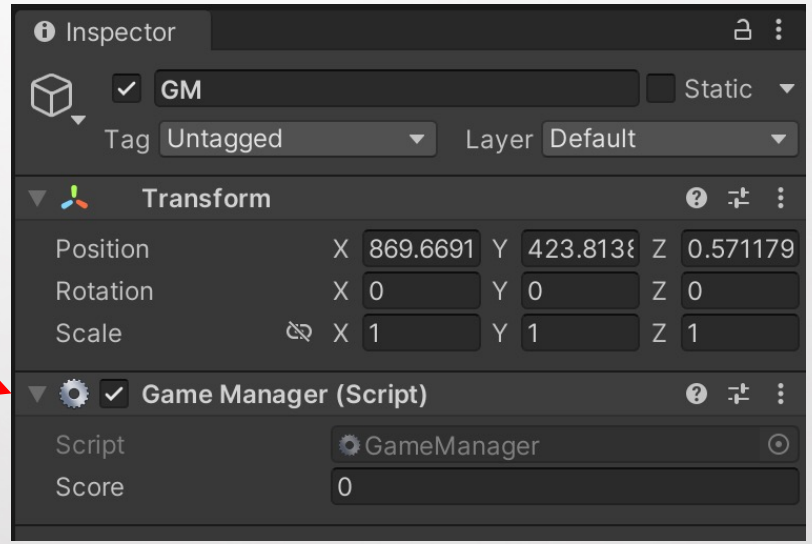
出来て確認終わった人は、実装内容を確認してみてください。
(GameManager.csを2つアタッチしたらどのようになるのか確認してください)

４．シングルトン実装

配置は、以下のようにお願いします。







５．シングルトン応用

では次にGameManagerではなく、SoundManagerやEffectManager
を作りたい場合、どうしますか？
似たようなコードを書くのは生産性と信頼性の低下が懸念されます。。
なので、新たにシングルトンクラスをGenericを使用して作成してみてください。
そして、今存在するGameManagerクラスとは別に、GameManager2クラス
を作成してみてください。(内容は同様に変数Scoreをインクリメントするだけで
OK)

終わった人は、作成したシングルトンクラスを更に効率的に使えるように、
以下の修正をしてみてください。
・シーンを跨いでも消えないように修正

5．シングルトン応用

＜シーンを跨いでも消えないように修正＞

```
public class Singleton<T> : MonoBehaviour where T : Singleton<T>
{
    public static T Instance = null;

    private void Awake()
    {
        //初めてのインスタンスであれば自身を代入する
        if (Instance == null)
        {
            //thisをキャストして代入する
            //(Sigletonクラスを継承していることが条件)
            Instance = (T)this;

            //シーンを跨いでも消えないように
            DontDestroyOnLoad(gameObject);
            return;
        }

        //既に代入済みであれば破棄する
        Destroy(this);
    }
}
```

６．シングルトンの問題点

・インスタンスが1つしか作れない

⇨当たり前ですが。。。ファイル操作やネットワーク通信はインスタンスが複数あるとバグの元になりますが、実際のところシングルトンを使うメリットとしてはグローバルにアクセスできる点だと思います。
複数のインスタンスを用意しておいて、用途によって切り替える方が便利なんじゃない、、、って人はサービスロケーターという別のデザインパターンがあるので、気になる人は調べてみてください。

・シングルトンはCでいうグローバル変数と同じ

⇨オブジェクト指向において、クラスは出来るだけ独立性を持った方が良いとされる。が、シングルトンを使うとクラス間の結合が強くなってしまう。
なので、多様はしない方が良いかも・・・。

６．シングルトンの問題点

・ジェネリックのシングルトンの問題点

⇨Awakeが乗っ取られる点。解消方法はあるため、どのようにしたら良いか考えてSingletonクラスに実装してみてください。

６．シングルトンの問題点

以下のように、Awake内でメソッドを作成して継承先で実装させることで対応可能。

```
private void Awake()
{
    //初めてのインスタンスであれば自身を代入する
    if (Instance == null)
    {
        //thisをキャストして代入する
        //(Sigletonクラスを継承していることが条件)
        Instance = (T)this;

        //シーンを跨いでも消えないように
        DontDestroyOnLoad(gameObject);

        //Awakeの任意の処理を実施
        Instance.SingletonAwake();
        return;
    }

    //既に代入済みであれば破棄する
    Destroy(this);
}

protected virtual void SingletonAwake()
{

}
```