Как работает ограничение where T: class?

Аргумент типа должен быть ссылочным типом. Это ограничение также применяется к любому типу класса, интерфейса, делегата или массива. В контексте, допускающем значения NULL, T должен быть ссылочным типом, не допускающим значения NULL.

Иногда, необходимо связать переменную обобщенного типа с null, но мы не можем сделать этого без использования ограничений, так как у вас нет гарантий, что переменная обобщенного типа не является типом значений для которых null бессмыслен.

Но это можно исправить ограничением class в where условии. Объявляя то, что обобщенный тип, должен быть class, мы говорим, что он является ссылочным типом и может принимать значение null для экземпляров этого типа:

```
public static class Test
   {
      public static T_Out Maybe<T_In, T_Out>(this T_In value, Func<T_In,
T_Out> accessor)
         where T_Out : class
         where T_In : class
      {
         return (value != null)
                  ? accessor(value)
                  : null;
      }
   }
```

Таким образом, при необходимости доступа к свойству ссылки, нам все равно является она null или нет, мы спокойно сможем работать с ней дальше. Для того, что бы это сделать оба типа, входной тип и выходной тип, должны быть ссылочными типами.

При применении ограничения where T : class не рекомендуется использовать операторы == и != для параметра типа, поскольку в этом случае будет проверяться только удостоверение ссылки, а не равенство значений. Такое поведение будет наблюдаться даже в том случае, если эти операторы будут перегружены в типе, используемом в качестве аргумента. Рассмотрим данную особенность на примере, который будет возвращать значение false даже в том случае, если класс String перегружает оператор ==.

```
public static void OpEqualsTest<T>(T S1, T S2) where T : class
   {
      System.Console.WriteLine(S1 == S2);
   }

   private static void TestStringEquality()
   {
      string s1 = "target";
      System.Text.StringBuilder sb = new System.Text.StringBuilder("target");
      string s2 = sb.ToString();
      OpEqualsTest<string>(s1, s2);

   }
```

Компилятору известно только то, что T является ссылочным типом во время компиляции, и он должен использовать операторы по умолчанию, которые действительны для всех ссылочных типов.