ТЕМА 7: ТИПИ, ЩО ДОПУСКАЮТЬ ЗНАЧЕННЯ null. СПЕЦІАЛЬНЕ ПЕРЕТВОРЕННЯ ТИПІВ

План

7.1. Оголошення типів, що допускають значення null. 1
7.2. Неявні та явні перетворення за участі типів, що допускають значення null. 2
7.3. Підіймання операцій. 2
7.4. Операції із значенням null. 2
7.5. Тип bool? та операції & і |. 3
7.6. Операція ?? (операція об'єднання з null) та null-умовна операція (?.). 3

7.1. Оголошення типів, що допускають значення null.

Типи посилань можуть представляти неіснуючі значення за допомогою посилання null. Проте типи значень не здатні звичним способом представляти значення null. Наприклад, всі числові типи а також bool є типами значень. А типам значень ніколи не може бути присвоєне значення null, оскільки це значення служить для представлення порожнього посилання на об'єкт:

```
bool myBool=null;     //помилка
int myInt=null;      //помилка
string myString=null; //
int[] myArray = null;
```

Мова C# підтримує концепцію типів даних, що допускають значення null. Такий тип може представляти всі значення типу плюс null. Це може бути зручним при роботі з реляційними базами даних, тому що у таблицях БД зустрічаються комірки, для яких значення не визначені. Тому для числових типів даних, що не мають ніякого присвоєного значення, існують типи даних, що допускають значення null.

Оголошення: `тип_змінної? ім'я_змінної=значення||null`.

```
bool? myBool=null;  //немає помилки
int? myInt=null;     //немає помилки
string? myString=null;     //помилка
int?[] myArray = {1,2,3,null };//немає помилки
```

Дізнатися, чи присвоєне значення, можна за допомогою властивості HasValue:

```
myInt = null;
if (myInt.HasValue)
     button2.Text = (myInt + 5).ToString();
else button2.Text = "has`nt Value";
```

Отримати значення можна за допомогою властивості **Value** або методу **GetValueOrDefault() | GetValueOrDefault(defaultValue):**

```
int? nullableInt = null;
if (nullableInt.HasValue)
```

```
        Console.WriteLine(nullableInt.Value);
else

        Console.WriteLine(nullableInt.GetValueOrDefault(1000));
```

7.2. Неявні та явні перетворення за участі типів, що допускають значення null.

Перетворення з T у T? є неявним, а із T? у T – явним. Наприклад:

```
int? nullableInt = null;
int x=5;
nullableInt = x; // неявне перетворення
//x = nullableInt;
x = (int)nullableInt; // явне перетворення
```

Явне перетворення типу повністю еквівалентне звертанню до властивості `Value` об'єкта типу, що допускає значення `null`. Тому, якщо `HasValue` рівне `null`, то генерується виняток `InvalidOperationException`.

7.3. Підіймання операцій.

Хоча для типу посилання T? не визначено такі операції, як <, >, +, наступний код компілюється:

```
int? a = 5;
int? b = 6;
bool r = a > b; //false
```

Код спрацьовує тому, що компілятор запозичує, або «підіймає» оперцію «<» у типу значення, що лежить в основі. Семантично вираз порівняння транслюється так:

```
r = (a.HasValue && b.HasValue) ? a.Value > b.Value : false;
```

Підіймання операцій означає можливість неявного використання операцій з T для типу T?.

7.4. Операції із значенням null.

Компілятор надає логіку по відношенню до `null` по-різному в залежності від категорії операції.

Операції еквівалентності (== та !=).

- Два значення `null` рівні між собою;
- якщо лише один операнд рівний `null`, то операнди не рівні;
- якщо обидва операнди відмінні від `null`, то порівнюються їх властивості `Value`.

Операції відношення (<, <=, >=, >). Робота операцій відношення ґрунтується на принципі, що порівняння операндів `null` не має змісту. Це значить, що порівняння `null` з `null` чи зі значенням, відмінним від `null`, дає в результаті `false` (трансляція у тернарну операцію `*?*:*`).

Всі інші операції (+, –, *, /, %, &, |, ^, <<, >>, +, ++, – –, !, ~). Ці операції повертають `null`, коли будь-який з операндів рівний `null`.

В операціях можна змішувати типи, що допускають та таких, що не допускають значення `null`.

7.5. Тип bool? та операції & і |.

У випадку, коли операнди тип bool?, операції & та | трактують `null` як *невідоме значення*. Тому `null | true == true`. Аналогічно `null & false == false`.

```
bool? n = null;
bool? f = false;
bool? t = true;
bool? res;

res = n | n; //null
res = n | f; //null
res = n | t; //true
res = n & n; //null
res = n & f; //false
res = n & t; //null
```

7.6. Операція ?? (операція об'єднання з `null`) та `null`-умовна операція (?.).

Ця операція (??) дозволяє для типів, що допускають значення `null`, задавати значення у випадку, якщо вони не мають значення.

Традиційний підхід:

```
if (a.HasValue)
       a += 5;
else
       a = 12;
```

Підхід із використанням операції ??:

```
a = a + 5 ?? 12;
```

Інший приклад:

```
string s = null;
Console.Write(s ?? " Ще не присвоєне значення");
```

null-умовна операція (?.).

```
string str = null;
int? l = str.Length; //System.NullReferenceException
int? l = str?.Length; //l is null
int? l = str?.Length ?? 0; //l is 0
```