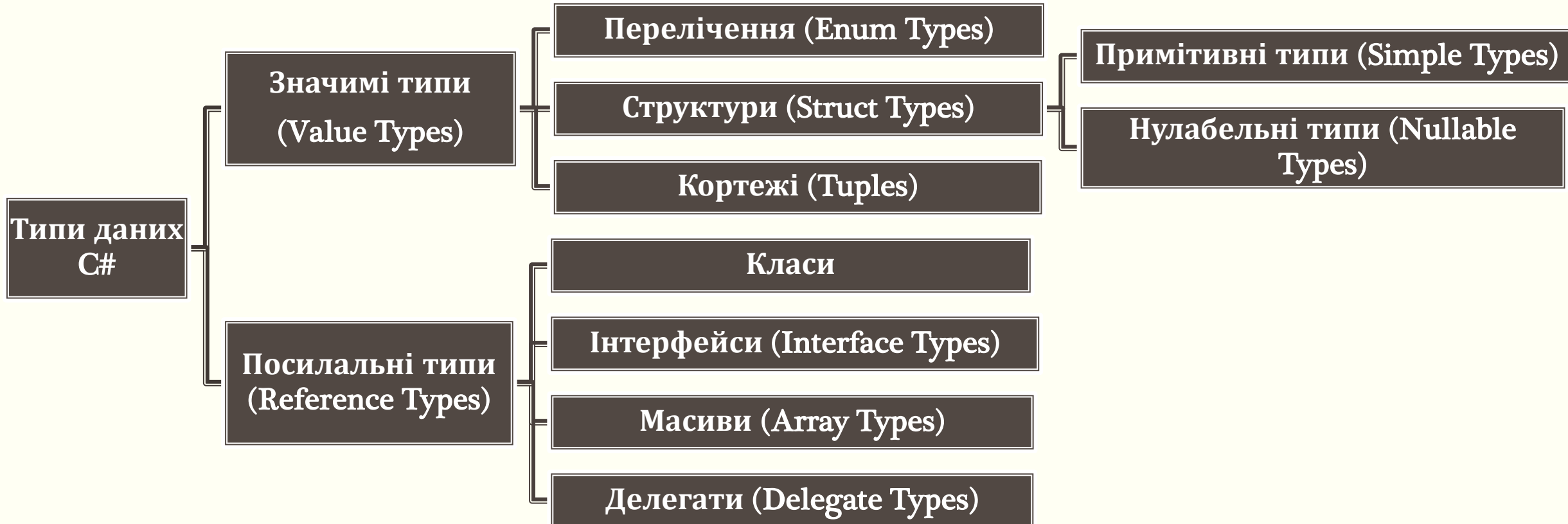




СИСТЕМА ТИПІВ МОВИ C#

Питання 1.4.

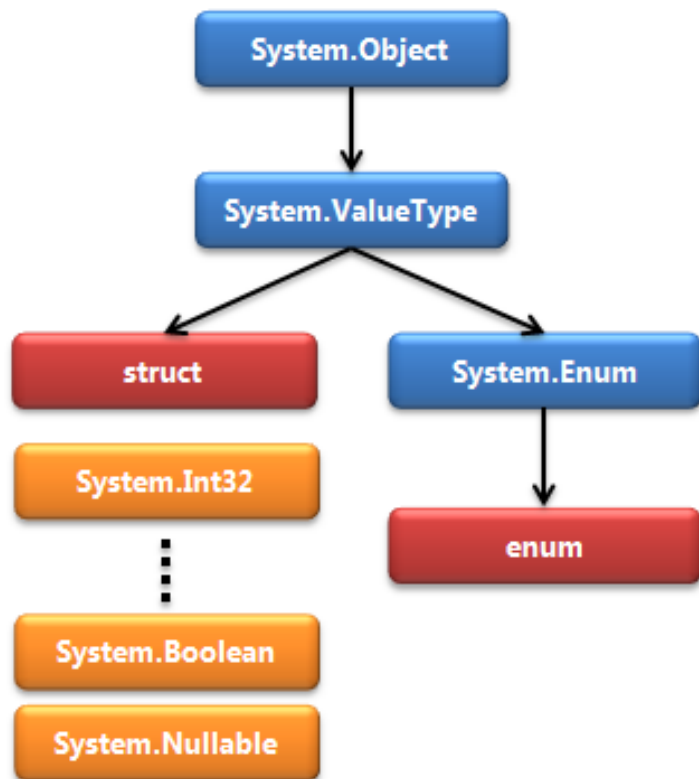
Система типів мови C#



Системні типи (значимі)

Ключове слово типу C#	Тип .NET	CLS	Діапазон значень	Розмір
<u>bool</u>	<u>System.Boolean</u>	+	true або false	1 байт (всередині – 0 або 1)
<u>byte</u>	<u>System.Byte</u>	+	0...255	8-розрядне ціле беззнакове число
<u>sbyte</u>	<u>System.SByte</u>	-	-128...127	8-розрядне ціле знакове число
<u>char</u>	<u>System.Char</u>	+	U+0000 ... U+FFFF	16 розрядів (UTF-16)
<u>decimal</u>	<u>System.Decimal</u>	+	$\pm 1,0 \cdot 10^{-28} \dots \pm 7,9228 \cdot 10^{28}$	16 байтів, 28-29 знаків
<u>double</u>	<u>System.Double</u>	+	$\pm 5,0 \cdot 10^{-324} \dots \pm 1,7 \cdot 10^{308}$	8 байтів, точність 15-17 цифр
<u>float</u>	<u>System.Single</u>	+	$\pm 1,5 \cdot 10^{-45} \dots \pm 3,4 \cdot 10^{38}$	4 байти, точність 6-9 цифр
<u>int</u>	<u>System.Int32</u>	+	-2147483648...2147483647	32-розрядне ціле знакове число
<u>uint</u>	<u>System.UInt32</u>	-	0...4294967295	32-розрядне ціле беззнакове число
<u>long</u>	<u>System.Int64</u>	+	-9 223 372 036 854 775 808 ... 9 223 372 036 854 775 807	64-розрядне ціле знакове число
<u>ulong</u>	<u>System.UInt64</u>	-	0...18 446 744 073 709 551 615	64-розрядне ціле беззнакове число
<u>short</u>	<u>System.Int16</u>	+	-32768...32767	16-розрядне ціле знакове число
<u>ushort</u>	<u>System.UInt16</u>	-	0 ... 65 535	16-розрядне ціле беззнакове число

Значимі типи в CLR



- Типи, породжені від `System.ValueType`, мають спеціальну поведінку в CLR.
 - Змінні містять значення напряду, а не через посилання.
 - Відсутні окреме виділення пам'яті в кучі та збирання сміття.
- Існує лише 2 категорії значимих типів: структури та перелічення
 - Системні числові типи є структурами.

```
...public readonly struct Boolean : IComparable, IComparable<Boolean>, IConvertible, IEquatable<Boolean>
{
    ...public static readonly string FalseString;
    ...public static readonly string TrueString;

    ...public static Boolean Parse(ReadOnlySpan<char> value);
    ...public static Boolean Parse(string value);
    ...public static Boolean TryParse(ReadOnlySpan<char> value, out Boolean result);
    ...public static Boolean TryParse(string? value, out Boolean result);
    ...public int CompareTo(Boolean value);
    ...public int CompareTo(object? obj);
    ...public Boolean Equals(Boolean obj);
    ...public override Boolean Equals(object? obj);
    ...public override int GetHashCode();
    ...public TypeCode GetTypeCode();
    ...public override string ToString();
    ...public string ToString(IFormatProvider? provider);
    ...public Boolean TryFormat(Span<char> destination, out int charsWritten);
}
```

Структури в C#

- Використовуються при проектуванні типів, орієнтованих на дані та з малою кількістю чи навіть відсутністю поведінки.
 - Числа, бульові значення, символи Unicode, дати та ін.

```
namespace System
{
    ...public readonly struct DateTime : IComparable, IComparable<DateTime>, IConvertible, IEquatable<D
    {
        ...public static readonly DateTime MaxValue;
        ...public static readonly DateTime MinValue;
        public static readonly DateTime UnixEpoch;

        ...public DateTime(long ticks);
        ...public DateTime(long ticks, DateTimeKind kind);
        ...public DateTime(int year, int month, int day);
        ...public DateTime(int year, int month, int day, Calendar calendar);
        ...public DateTime(int year, int month, int day, int hour, int minute, int second);
        ...public DateTime(int year, int month, int day, int hour, int minute, int second, DateTimeKind
        ...public DateTime(int year, int month, int day, int hour, int minute, int second, Calendar cal
        ...public DateTime(int year, int month, int day, int hour, int minute, int second, int millisec
        ...public DateTime(int year, int month, int day, int hour, int minute, int second, int millisec
        ...public DateTime(int year, int month, int day, int hour, int minute, int second, int millisec
        ...public static DateTime Now { get; }
        ...public static DateTime Today { get; }
    }
}
```

```
using System;

namespace StructureType
{
    struct User
    {
        public string name;
        public int age;

        public void DisplayInfo()
        {
            Console.WriteLine($"Name: { name} Age: { age}");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            User tom;
            tom.name = "Tom";
            tom.age = 34;
            tom.DisplayInfo();

            Console.ReadKey();
        }
    }
}
```

```
C:\Users\puasson\source\
Name: Tom Age: 34
```

Особливості роботи зі структурами

- Створення структури:

- Конструювання `User tom = new User();` не обов'язкове
- Без конструювання **обов'язково** потрібно проініціалізувати всі поля структури.

- Ініціалізація структури:

- Через конструктор:

```
public User(string name, int age) {  
    this.name = name;  
    this.age = age;  
}  
User person = new User("Sam", 31);
```
- Через ініціалізатор:

```
User person = new User { name = "Sam", age = 31 };
```
- Поелементно:

```
User sam;  
sam.name = "Sam";  
sam.age = 31;
```

Перелічення (перелічуваний тип, enum type)

```
using System;
```

```
namespace Enums
```

```
{
```

```
    enum CaseColor {  
        Uncolored = 0,  
        Red = 8001,  
        Blue = 9001  
    }
```

```
    class Program {  
        static void Main(string[] args) {  
            // зрозуміліше  
            Console.WriteLine("COLOR: {0}, {1}",  
                CaseColor.Blue,  
                (int)CaseColor.Blue);  
        }
```

```
        // складніше для розуміння  
        Console.WriteLine("COLOR: {0}, {1}",  
            "Blue",  
            9001);  
    }
```

```
}
```

```
}
```

- Значимий тип, визначений набором іменованих констант цілочисельного типу.
 - Використовується ключове слово enum
 - Надає модульний дизайн для роботи з константами, знижує ймовірність появи багів.
 - За умовчанням, числове представлення значень починається з нуля та збільшується відповідно до порядку елемента в переліченні.

```
[Flags]
public enum Days {
    None = 0b_0000_0000,    // 0
    Monday = 0b_0000_0001,   // 1
    Tuesday = 0b_0000_0010,   // 2
    Wednesday = 0b_0000_0100, // 4
    Thursday = 0b_0000_1000,   // 8
    Friday = 0b_0001_0000,     // 16
    Saturday = 0b_0010_0000,   // 32
    Sunday = 0b_0100_0000,     // 64
    Вихідні = Saturday | Sunday
}
```

```
public class FlagsEnumExample {
    public static void Main()
    {
        Console.OutputEncoding = Encoding.UTF8;
        Days meetingDays = Days.Monday | Days.Wednesday | Days.Friday;
        Console.WriteLine(meetingDays);

        Days workingFromHomeDays = Days.Thursday | Days.Friday;
        Console.WriteLine($"Приєднуйтесь до ZOOM конференції в {meetingDays & workingFromHomeDays}");

        bool isMeetingOnTuesday = (meetingDays & Days.Tuesday) == Days.Tuesday;
        Console.WriteLine($"Чи є зустріч у вівторок: {isMeetingOnTuesday}");

        var a = (Days)18;
        Console.WriteLine(a);
    }
}
```

Перелічення як бітові прапорці

- Бажаємо представити комбінацію варіантів вибору за допомогою перелічення.
 - Пов'язані значення цих елементів можуть бути степенями двійки.
 - Потім застосовуються побітові оператори для комбінації обраних елементів.
 - Щоб вказати, що тип перелічення оголошує бітові поля, застосуйте до нього атрибут Flags.

Кортежі (Tuples, починаючи з C# 7.0) – сімейство типів (*)

- (*)-типи `System.ValueTuple` представлені в .NET Framework 4.7, проте можуть використовуватись у раніших версіях при додаванні спеціального пакету `nuget: System.ValueTuple`.
 - Якщо потрібно «склеїти» два значення, щоб повернути їх з функції, або помістити два значення в хеш-набір, можна використовувати типи `System.ValueTuple`.
- Кортеж – набір значень у круглих дужках:
 - `var tuple = (5, 10);`
 - `(int, int) tuple = (5, 10);`
 - `Console.WriteLine(tuple.Item1); // 5`
 - `Console.WriteLine(tuple.Item2); // 10`
 - `(string, int, double) person = ("Tom", 25, 81.23);`
- Також можемо іменувати поля кортежа: `var tuple = (count:5, sum:10);`
 - `Console.WriteLine(tuple.count); // 5`
- Дозволяється не створювати окрему змінну для визначення кортежа:
 - `var (name, age) = ("Tom", 23);`
 - `Console.WriteLine(name); // Tom`
 - `Console.WriteLine(age); // 23`

Використання кортежів

- Кортежі можуть передаватись у якості параметрів у метод, повертатись функцією або використовуватись іншим чином.

```
static void Main(string[] args)
{
    var (name, age) = GetTuple(("Tom", 23), 12);
    Console.WriteLine(name);    // Tom
    Console.WriteLine(age);     // 35
    Console.Read();
}

private static (string name, int age) GetTuple((string n, int a) tuple, int x)
{
    var result = (name: tuple.n, age: tuple.a + x);
    return result;
}
```

Зведення (casting) та перетворення типів (type conversions)

- Види перетворень типів у мові C#:

- Неявні перетворення типів (Implicit conversions): не потребують спеціального синтаксису, неявно виконуються компілятором

```
int num = 2147483647;  
long bigNum = num;
```

- Зведення типів (Explicit conversions, casts): вимагають виразу зведення. Потрібне у випадках, коли при перетворенні типів може втрачатись інформація або будуть причини для невдалого перетворення.

```
double x = 1234.7;  
int a = (int)x;
```

- Користувацькі перетворення типів (User-defined conversions): виконуються спеціальними методами, визначеними розробником для перетворення власних типів даних, які не пов'язані наслідуванням.
- Перетворення типів за допомогою допоміжних класів (helper classes): застосовуються спеціальні допоміжні класи для перетворення двох несумісних типів. Серед таких класів System.BitConverter та System.Convert.

Посилальні системні типи

Ключове слово типу C#	Тип .NET	CLS	Діапазон значень	Короткий опис
<u>object</u>	<u>System.Object</u>	+	Використовується для зберігання будь-якого типу в пам'яті	Базовий клас для всіх типів .NET
<u>string</u>	<u>System.String</u>	+	Обмежений об'ємом доступної пам'яті	Послідовна колекція об'єктів типу System.Char
<u>dynamic</u>	<u>System.Object</u>	+	Динамічне поле зазвичай обробляється компілятором <u>як поле типу object</u>	Тип статичний, проте dynamic-об'єкт обходить статичну перевірку типів

```
using System;
```

```
namespace DynamicType
```

```
{  
    class Student { }
```

```
    class Program
```

```
{
```

```
    static void Main(string[] args)
```

```
{
```

```
        dynamic MyDynamicVar = 100;
```

```
        Console.WriteLine("Value: {0}, Type: {1}",  
            MyDynamicVar, MyDynamicVar.GetType());
```

```
        MyDynamicVar = "Hello World!!";
```

```
        Console.WriteLine("Value: {0}, Type: {1}",  
            MyDynamicVar, MyDynamicVar.GetType());
```

```
        MyDynamicVar = true;
```

```
        Console.WriteLine("Value: {0}, Type: {1}",  
            MyDynamicVar, MyDynamicVar.GetType());
```

```
        MyDynamicVar = DateTime.Now;
```

```
        Console.WriteLine("Value: {0}, Type: {1}",  
            MyDynamicVar, MyDynamicVar.GetType());
```

```
        MyDynamicVar = new Student();
```

```
        // випадки з помилкою виконання, проте не компіляції
```

```
        MyDynamicVar.DisplayStudentInfo(1, "Станіслав");
```

```
        MyDynamicVar.DisplayStudentInfo("1");
```

```
        MyDynamicVar.FakeMethod();
```

```
    }
```

```
}
```

```
}
```

28.08.2020

Тип dynamic

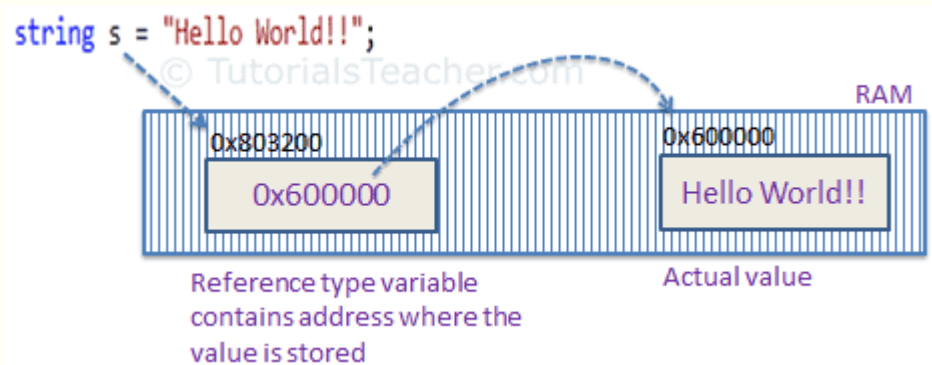
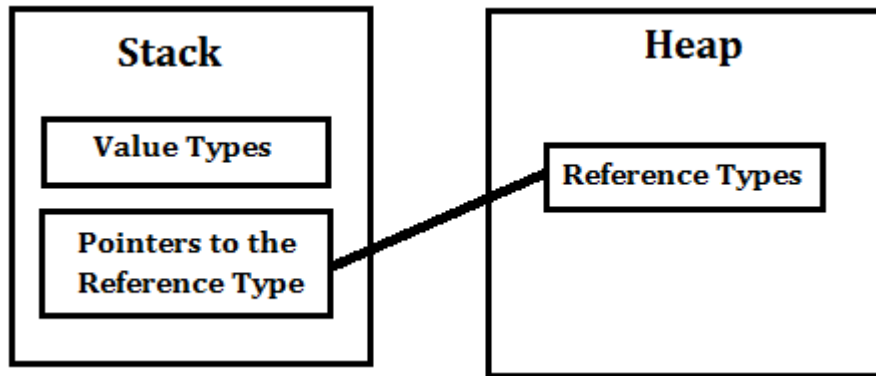
- Під час компіляції припускається, що динамічний тип підтримує будь-які операції.
 - Тоді не потрібно задумуватись, звідки об'єкт отримує значення: з API COM, з динамічної мови на зразок IronPython, з моделі DOM HTML, з відображення або іншої частини програми.
 - Якщо код недопустимий, помилки перехоплюються під час виконання.
- Довідка по dynamic.

Компіляція проектів

- Для компілювання коду без запуску використовується комбінація клавіш Ctrl+Shift+B.
 - Рішення має ще кілька проектів, проте помилок при компіляції не знайдено.

```
Вывод
Показать выходные данные из: Сборка
1>----- Сборка начата: проект: TimeStamp, Конфигурация: Debug Any CPU -----
2>----- Сборка начата: проект: AssemblyVsNamespace, Конфигурация: Debug Any CPU -----
3>----- Сборка начата: проект: Numbers, Конфигурация: Debug Any CPU -----
4>----- Сборка начата: проект: ValueVsReference, Конфигурация: Debug Any CPU -----
4>C:\Users\puasson\source\repos\TimeStamp\ValueVsReference\Program.cs(13,20,13,21): warning CS0168: Переменная "j" объявлена, но ни разу не использована.
3>Numbers -> C:\Users\puasson\source\repos\TimeStamp\Numbers\bin\Debug\netcoreapp3.1\Numbers.dll
5>----- Сборка начата: проект: DynamicType, Конфигурация: Debug Any CPU -----
4>ValueVsReference -> C:\Users\puasson\source\repos\TimeStamp\ValueVsReference\bin\Debug\netcoreapp3.1\ValueVsReference.dll
4>Сборка проекта "ValueVsReference.csproj" завершена.
1>TimeStamp -> C:\Users\puasson\source\repos\TimeStamp\TimeStamp\bin\Debug\netcoreapp3.1\TimeStamp.dll
2>AssemblyVsNamespace -> C:\Users\puasson\source\repos\TimeStamp\AssemblyVsNamespace\bin\Debug\netcoreapp3.1\AssemblyVsNamespace.dll
5>DynamicType -> C:\Users\puasson\source\repos\TimeStamp\DynamicType\bin\Debug\netcoreapp3.1\DynamicType.dll
===== Сборка: успешно: 5, с ошибками: 0, без изменений: 3, пропущено: 0 =====
```

Відмінності між значимими та посилальними типами



Розташування в пам'яті:

- Значимий тип зберігає дані у власній виділеній пам'яті. Змінні значимих типів знаходяться в сегменті стеку.
- Посилальний тип містить посилання на місце в пам'яті, яке зберігає реальні значення. Змінні посилальних типів знаходяться в сегменті кучі.

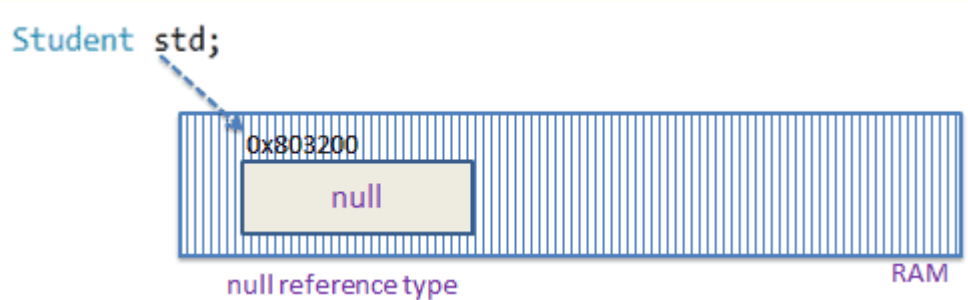
Виділення пам'яті:

- При створенні змінної значимого типу виділяється єдине місце в пам'яті для зберігання значення. При присвоєнні цього значення іншій змінній воно копіюється, й обидві змінні працюють із значеннями незалежно.
- Посилальні типи зберігають посилання (адресу) на об'єкт, тому присвоєння однієї змінної посилального типу іншій копіює не значення, а адресу об'єкта.

Збирання сміття:

- Змінні значимих типів можуть створюватись під час компіляції в стеку, тому збирач сміття не може отримувати доступ до стеку.
- Змінні посилальних типів, якщо не використовуються, можуть бути відмічені на збирання сміття.

Відмінності між значимими та посилальними типами



```
static void Main(string[] args)
{
    Console.OutputEncoding = Encoding.UTF8;
    // значимі типи
    int i = 1;
    double j;

    // посилальні типи
    object obj = null;
    string str = String.Empty;

    // нулабельні типи
    int? k = null;
    int? f = 23;

    Console.WriteLine($"Значимі типи: i = {i}");
    //Console.WriteLine("j = {j}"); // помилка
    Console.WriteLine($"Посилальні типи: obj = {obj}, str = {str}");
    Console.WriteLine($"Нулабельні типи: k = {k}, f = {f}");
}
```

■ Значення *null* – пам'ять, яка не належить жодному об'єкту:

- Для змінних посилальних типів – значення за умовчанням до моменту ініціалізації.
- Змінні значимих типів не можуть отримувати значення `null`, оскільки це адреса пам'яті, а не значення.
- У C# 2.0 введено **нулабельні типи**, яким можна присвоювати значення

Консоль отладки Microsoft Visual Studio

Значимі типи: i = 1
Посилальні типи: obj = , str =
Нулабельні типи: k = , f = 23

Значення null та нулабельні типи

- Значимі типи, на відміну від посилальних, не можуть приймати значення null.

```
0 references
static void Main(string[] args)
{
    int x = null;
    string p = null;
}
```

Cannot convert null to 'int' because it is a non-nullable value type

- Існує 2 вирішення проблеми:

- (1) Огорнути значення в *об'єкт* класу System.Nullable:
- (2) Застосувати скорочений синтаксис (1):

```
System.Nullable<int> x = null;
int? x = null;
```

- Застосування нулабельних типів підвищує безпеку коду, оскільки NPE-помилки відстежуються ще на етапі компіляції.

- Також корисно при прив'язуванні даних у WPF, коли дані для відображення на формі відсутні.
- Нулабельні типи можуть використовуватись *лише* для значимих типів.
- Нулабельні типи *не можна вкладати*:

```
Nullable<Nullable<bool>> value;
```

struct System.Nullable<T> where T : struct
Represents a value type that can be assigned null.

Т является bool

IDE0001: Имя может быть упрощено.

CS0453: Для использования в качестве параметра "T" в универсальном типе или методе "Nullable<T>" тип "bool?" должен быть типом значения, не допускающим значения Null.

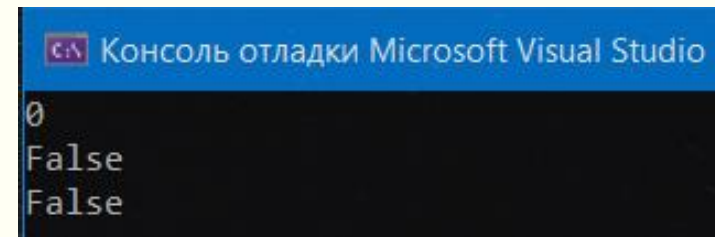
Особливості роботи з нулабельними типами

```
using System;

namespace NullableTypes
{
    class Program
    {
        static void Main(string[] args)
        {
            int? i = null;
            int j = i ?? 0;
            int k = -1000;

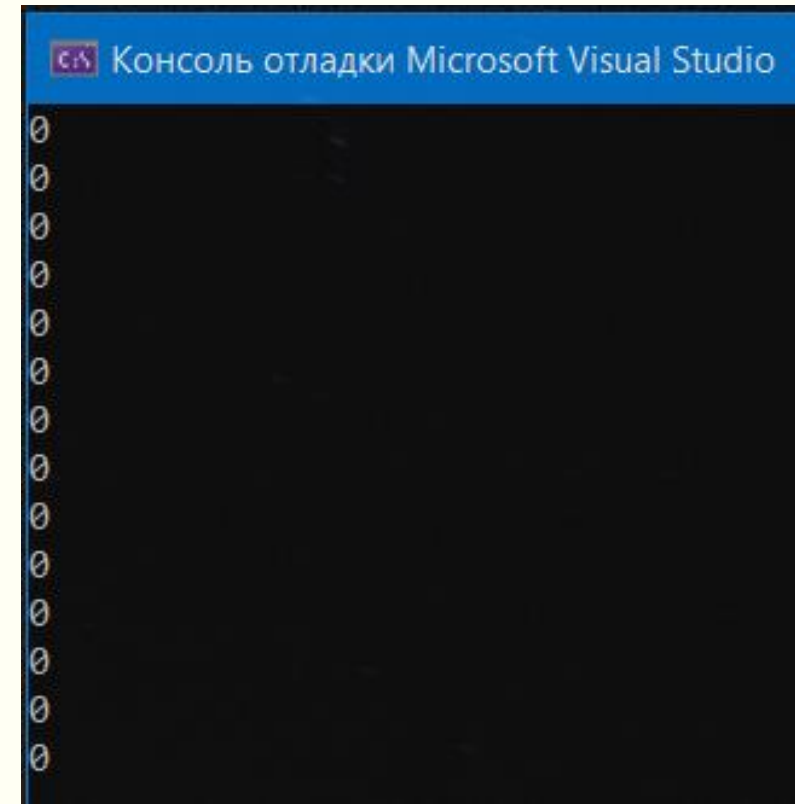
            Console.WriteLine(j);
            // null менший за інші значення
            Console.WriteLine(k < i);
            Console.WriteLine(Nullable.Compare<int>(k, i) < 0);
        }
    }
}
```

- null-значення вважається меншим за будь-яке інше значення, тому можуть логічно некоректно спрацьовувати умовні оператори.
 - Застосовується спеціальний метод Compare з класу `Nullable`.
 - Повертає значення для змінних *a* і *b* (з хешем):
 - `< 0`: *a* < *b* за значенням
 - `0`: *a* = *b* за значенням
 - `> 0`: *a* > *b* за значенням
- Оператор об'єднання з NULL (null-coalescing operator) `??` дозволяє замінити null-значення на безпечне для виконання.



Особливості null-арифметики

```
namespace NullArithmetics
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine($"{10 / null ?? 0}");
            Console.WriteLine($"{10 * null ?? 0}");
            Console.WriteLine($"{10 + null ?? 0}");
            Console.WriteLine($"{10 - null ?? 0}");
            Console.WriteLine($"{10 % null ?? 0}");
            Console.WriteLine($"{null / 10 ?? 0}");
            Console.WriteLine($"{null * 10 ?? 0}");
            Console.WriteLine($"{null + 10 ?? 0}");
            Console.WriteLine($"{null - 10 ?? 0}");
            Console.WriteLine($"{null % 10 ?? 0}");
            Console.WriteLine($"{null / null ?? 0}");
            Console.WriteLine($"{null * null ?? 0}");
            //Console.WriteLine($"{null + null ?? 0}"); // помилка
            Console.WriteLine($"{null - null ?? 0}");
            Console.WriteLine($"{null % null ?? 0}");
            //Console.WriteLine($"{null == null ?? 0}"); // помилка
        }
    }
}
```



```
namespace NullConditional
```

```
{
```

```
class Program
```

```
{
```

```
struct Person
```

```
{
```

```
public string FirstName { get; set; }
```

```
public string LastName { get; set; }
```

```
public static void DisplayPerson(Person? person)
```

```
{
```

```
    Console.OutputEncoding = Encoding.UTF8;
```

```
    Console.WriteLine(person?.FirstName ?? "Ім'я за умовчанням");
```

```
    Console.WriteLine(person?.LastName ?? "Прізвище за умовчанням");
```

```
}
```

```
}
```

```
static void Main(string[] args)
```

```
{
```

```
    Person? person1 = null;
```

```
    Person? person2 = new Person {
```

```
        FirstName = "Станіслав", LastName = null };
```

```
    Person.DisplayPerson(person1);
```

```
    Person.DisplayPerson(person2);
```

```
    Person?[] persons = new Person?[2] { person1, person2 };
```

```
    Console.WriteLine(persons?[1].Value.FirstName ?? "Порожнє ім'я");
```

```
    Console.WriteLine(persons?[1].Value.LastName ?? "Порожнє прізвище");
```

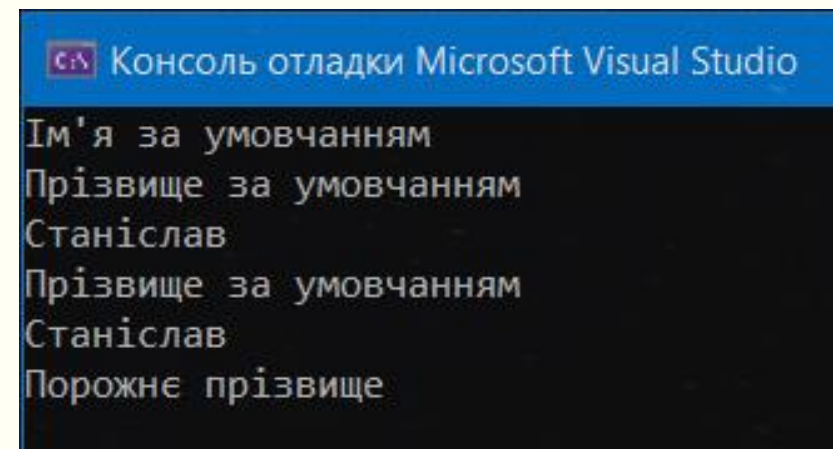
```
}
```

```
}
```

```
}
```

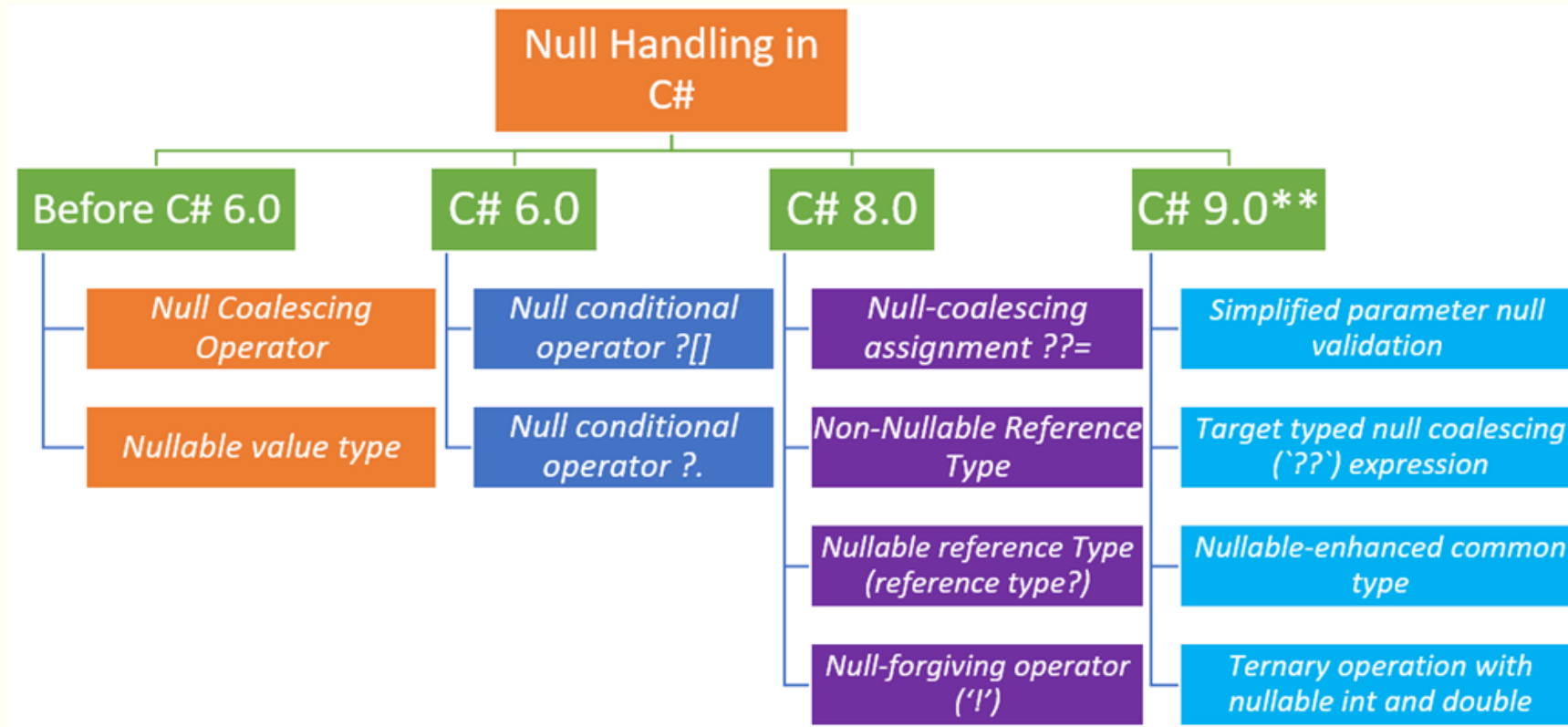
NULL-умовні оператори: ?. та ?[] (починаючи з C# 6.0)

- ?. - NULL-умовний оператор для доступу до членів об'єкта
- ?[] - NULL-умовний оператор для доступу до елементів



```
Консоль отладки Microsoft Visual Studio
Ім'я за умовчанням
Прізвище за умовчанням
Станіслав
Прізвище за умовчанням
Станіслав
Порожнє прізвище
```

Загальна обробка Null у мові C#



- Сценарії застосування нулабельних типів



ДЯКУЮ ЗА УВАГУ!

Наступна тема: Знайомство з базовим синтаксисом мови програмування C#