Self-Study Report on Bitwise Operators in C#
Day2

1. Introduction to Bitwise Operators

الـ Bitwise Operators هي عوامل بتتعامل مع الأرقام على مستوى الـ Bits يعني بتشتغل على التمثيل الثنائي (البايناري) للأرقام. في C# فيه شوية عوامل بتتعامل مع Bits زي:

AND (&)

OR (|)

XOR (^)

NOT (~)

Left Shift (<<)

Right Shift (>>)

الهدف من كدة:

تعلم استخدام الـ Bitwise Operators دي، سواء في العمليات المنطقية أو في اني احركها لليسار واليمين.

2. Bitwise AND (&) Operator

الـ AND اللي هو & بيقارن كل Bit من الرقمين، وبيرجع قيمة 1 لو الـ Bits اللي اتقارنوا كانوا 1 في الاتنين، غير كده بيرجع 0
مثال:

```
Console.WriteLine(false & true); // Output: False
Console.WriteLine(true | false); // Output: True
```

الـ false & true هيكون النتيجة بتاعتها false عشان في الـ AND لازم الاتنين يكونوا 1 علشان النتيجة تبقى 1. بما إن في واحد منهم false يعني 0، يبقى النتيجة false.
والـ true | false هيكون النتيجة true عشان في الـ OR لو واحدة من الـ Bits كانت 1 فالناتج هيكون true

مثال اخر(بس مع أرقام):

```
int a = 5; // binary: 0101
int b = 3; // binary: 0011
Console.WriteLine(a & b); // Output: 1, binary: 0001
```

الرقم 5 بالـ binary هو 0101 والـ 3 هو 0011. لما نعمل AND ما بينهم هنلاقي النتيجة هي 0001، يعني 1 بالـ decimal

3. Bitwise OR (|) Operator

الـ ORاللي هو | بيقارن كل Bitمن الرقمين، وبيرجع 1 لو على الأقل واحدة من الـ Bitsكانت 1. لو كانوا الاتنين 0 بيرجع 0

مثال:

```
Console.WriteLine(true | false); // Output: True
Console.WriteLine(4 | 5); // Output: 5 (binary: 0100 | 0101 = 0101)
```

في المثال الأول، true | falseالنتيجة هتكون trueعشان الـ ORبيرجع trueلو واحدة من الـ Bitsكانت 1.
في المثال التاني، 4 | 5 بالـ binaryبيبقى 0100 و 0101. فالناتج من الـ ORهيكون 0101، اللي هي 5 بالـ decimal

4. Left Shift (<<) Operator

الـ Left Shiftاللي هو << بيشيل الـ Bitsلليسار بمقدار معين، والمكان اللي بيتشال منه بيتحط مكانه 0. مع كل شيفت لليسار، الرقم بيكبر زي ما لو ضربناه في 2.
مثال:

```
int a = 4; // binary: 0100
Console.WriteLine(a << 1); // Output: 8 (binary: 1000)
```

الرقم 4 بالـ binaryهو 0100. لما نعمل له Shiftلليسار مرة واحدة، النتيجة هتبقى 1000، اللي هي 8 بالـ decimal

5. Right Shift (>>) Operator

الـ Right Shiftاللي هو >> بيشيل الـ Bitsلليمين بمقدار معين. لو الرقم موقعه) signمثلاً، لو كان رقم سالب)، بيبقى فيه بعض العمليات علشان يحافظ على الإشارة المعروفة بالـ Arithmetic Shift

```
int a = 8; // binary: 1000
Console.WriteLine(a >> 1); // Output: 4 (binary: 0100)
```

الرقم 8 بالـ binaryهو 1000 فلما نعمل له Shiftلليمين مرة واحدة، النتيجة هتبقى 0100، اللي هي 4 بالـ decimal

6. Combining Operators (Left & Right Shift with Bitwise AND/OR)

ممكن أحيانًا تستخدم Shift operators زي << و >> مع (&) AND و (|) OR علشان تتحكم في البيانات بشكل أدق، زي إنك تنضف Bits معينة أو تتحقق من الـ Bit في مكان معين.

```
int a = 5;  // binary: 0101
Console.WriteLine((a << 1) & 3);  // Output: 2 (binary: 0010)
```

الأول بنعمل Shift لليسار على 5 فتبقي 10 (binary 1010)
بعد كده بنعمل AND مع 3 (binary 0011) فالناتج بيبقى 2 (binary 0010)

لو حصل overflowزيادة القيمة عن الحد المسموح بيه في العمليات الحسابية، في الوضع العادي بيظهر خطأ OverflowExceptionلكن باستخدام unchecked، البرنامج يتجاهل الخطأ ده ويكمل.
إزاي نستخدم unchecked؟

لو عندي عملية زي جمع أو ضرب والنتيجة ممكن تكون أكبر من الحد المسموح، تقدر تستخدم uncheckedعلشان تتجاهل الخطأ:

```
int a = int.MaxValue;  // Largest possible value of int
int b = 1;
int result = unchecked(a + b);  // Result will wrap around without throwing an error
Console.WriteLine(result);  // Output: -2147483648
```

الفرق بين checkedو unchecked
Checked لو حصل overflow، البرنامج هيرمي .OverflowException
Unchecked يتجاهل overflowويكمل بدون ما يرمي خطأ.

```
int a = int.MaxValue;
int b = 1;

try
{
   int resultChecked = checked(a + b);  // Throws an exception if overflow occurs
   Console.WriteLine(resultChecked);
}
catch (OverflowException ex)
{
   Console.WriteLine("Overflow with checked: " + ex.Message);  // This will print
the overflow error
}

int resultUnchecked = unchecked(a + b);  // Ignores overflow
Console.WriteLine(resultUnchecked);  // The result will wrap around and be
negative
```

Garbage Collector (GC) هو جزء من CLR (Common Language Runtime) وبيساعد في إدارة الذاكرة بشكل تلقائي. مهمته الأساسية هي التأكد من أن الذاكرة اللي مش مستخدمة (الـ Objects اللي ما بقاش ليها مرجعية تتجمع وتحرر عشان تقدر تستخدمها تاني.

إزاي بيشتغل Garbage Collector؟
تتبع الـ Objects: الـ GC بيتتبع الـ objects في الذاكرة، وبيشوف إذا كان في object لسه ليه مرجعية في الكود (يعني لسه بيتم استخدامه) أو لو بقى مش مستخدم (مفيش حاجة بترجع له).
جمع الذاكرة (Collecting): لما يلاقي أن الـ object مش مستخدم، الـ GC بيقوم بتحرير المساحة بتاعته في الذاكرة علشان تكون جاهزة للاستخدام تاني.
التنظيف التلقائي: الـ GC بيشتغل في خلفية البرنامج بشكل تلقائي، يعني مش لازم تكتب كود خاص بيه، هو بيقوم بالعملية دي تلقائيًا في الوقت المناسب.

متى يبدأ Garbage Collector؟
الـ Garbage Collector بيبدأ في جمع الـ objects اللي مش مستخدمة عندما:
الذاكرة بتقل: لو النظام عنده ضغط في الذاكرة وعايز يحرر مساحات جديدة.
عند الحاجة: الـ GC ممكن يبدأ تنظيف تلقائيًا بعد تنفيذ عمليات معينة أو بشكل دوري حسب الحاجة.
مراحل الـ Garbage Collection:
الـ GC في C# مش بيجمع كل الـ objects مرة واحدة، لكن بيقوم بعمليه جمع بالتدريج على 3 مراحل Generations
Generation 0 أصغر جيل، بيحصل فيه جمع بشكل متكرر.
Generation 1 إذا كان الـ object عاش لفترة أطول من Generation 0
Generation 2 أكبر جيل، وبيشمل الـ objects اللي عاشوا لفترة طويلة جدًا.