Advanced Features in OOP

C# AND THE .NET FRAMEWORK

Topics

- Exceptions
- Extension Methods
- Conversions
- File Handling
- Collections
- LINQ (Language Integrated Query)

Codes

Source code

https://github.com/OOP-03376400/WEEK-13

Exceptions

Exceptions

ในการพัฒนาโปรแกรม จะมี error เกิดขึ้นได้ 2 แบบ

1. Compilation error
   - การผิดพลาดลักษณะนี้ Compiler จะแจ้งเตือน
2. Run time error
   - การผิดพลาดลักษณะนี้ Compiler จะไม่แจ้งเตือน
   - จะเกิดการ crashes ของโปรแกรมขณะรัน
   - การ crashes ของโปรแกรมเรียกว่าเกิด Exception

Exceptional Circumstances

Exceptions จะเกิดเมื่อโปรแกรมของเราทำงานผิดพลาด

- พยายามแปลง invalid string
- พยายามเข้าถึงสมาชิกอาเรย์ที่อยู่นอกขอบเขต
- พยายามใช้งานตัวแปร reference ที่เป็น null
- ...และอีกหลายความพยายาม...

ภาษา C# มีกลไกรองรับ เพื่อตรวจจับ exceptions ต่างๆ

What is an exception?

- Exception เป็นข้อมูลจำเพาะชุดหนึ่ง ที่บอกว่าโปรแกรมของเรามีความผิดพลาดตรงไหน
- เรามักเจอในโปรแกรมที่ทำงานผิดพลาด
- เราอาจใช้ข้อมูลเหล่านั้นเพื่อหาที่ผิดพลาดในโปรแกรมขณะทำงาน
- Exceptions สามารถเกิดได้จากโปรแกรม หรือ ระบบปฏิบัติการ (การสั่งให้เกิด exception เรียกว่า "thrown“)
- ถ้าไม่มีการเขียนโปรแกรมรองรับ exception (ทางเทคนิคเรียกว่า "caught“) โปรแกรมเราจะ crashes

ตัวอย่าง สาเหตุของ exceptions

```
static void Main()
{
   int x = 10, y = 0;
   x /= y;                  // Attempt to divide by zero--raises an exception
}
```

- โปรแกรมนี้ มีการหารตัวเลขด้วยค่าศูนย์ ซึ่งคอมพิวเตอร์ไม่สามารถให้คำตอบได้จึงเกิด exception ขึ้น

```
Unhandled Exception: System.DivideByZeroException: Attempted to divide by zero.
       at Exceptions_1.Program.Main() in C:\Progs\Exceptions\Program.cs:line 12
```

Catching exceptions

```
static void Main()
{
   int x = 10;

   try
   {
      int y = 0;
      x /= y;                          // Raises an exception
   }
   catch
   {
         ...                           // Code to handle the exception

      Console.WriteLine("Handling all exceptions - Keep on Running");
   }
}
```

```
Handling all exceptions - Keep on Running
```

ตัวอย่าง สาเหตุของ exceptions

```
int i;
i = int.Parse("One");
```

- โปรแกรมนี้ ต้องการ string ที่ประกอบด้วยตัวเลข 0-9 เพื่อจะไปแปลงเป็นจำนวน integer
- แต่ parameter ที่ส่งให้ Parse กลับเป็นตัวอักษร

Catching exceptions

```
try
{
  i = int.Parse("One");
}
catch
{
  Console.WriteLine("Invalid number");
}
```

- ถ้าโปรแกรมในบล็อก try ทำงานล้มเหลว จะเป็นการสร้าง exception
- โปรแกรมจะกระโดดมาทำงานใน catch
  - โดยโปรแกรมของเราไม่ crashes

Structure of the try statement

`try` block: This block contains the statements being guarded for exceptions.

```
try
{
  statements
}
```

This section is required.

`catch` clauses: This section contains the exception handlers for exceptions thrown in the `try` block.

```
catch( ... )
{
   statements
}
catch( ... )
{
   statements
}
catch ...
```

`finally` block: This block contains code to be executed whether or not an exception is thrown in the `try` block.

```
finally
{
   statements
}
```

One or both of these sections must be present. If both sections are present, the `finally` block must be placed last.

Exception objects

- ส่วนใดๆ ในโปรแกรมที่น่าสงสัยว่าจะเกิด exception เราสามารถล้อมรอบด้วย try
  - เช่นการเขียนอ่านไฟล์, การเข้าถึง network, การใช้งาน database
- เราสามารถเขียน โปรแกรมไว้ดัก exception ต่างๆ ได้ตามต้องการ
- การดัก exception ใช้บล็อกของ catch

Catching exception details

```
try
{
  i = int.Parse("One");
}
catch (Exception e)
{
  Console.WriteLine(e.Message);
  Console.WriteLine(e.StackTrace);
}
```

- หากต้องการดูรายละเอียดของ exception เราสามารถใช้ exception object มาแสดงรายละเอียด

Finally

- บางครั้ง เราอาจต้องการให้โปรแกรมเราทำงานอย่างปกติไม่ว่าจะมี exception เกิดขึ้นหรือไม่
  - การเชื่อมต่อ network ไม่สำเร็จ ต้องทำการยกเลิกการเชื่อมต่อ
  - การเปิดไฟล์แล้วเขียนไม่สำเร็จ ต้องปิดไฟล์นั้น
  - ฯลฯ
- โครงสร้าง try-catch จะมี finally ไว้ให้ทำในสิ่งที่เราต้องการ
  - คำสั่งใน finally จะถูกเรียกเสมอ ไม่ว่าจะเกิด exception หรือไม่

The Finally Clause

```
try
{
  // Code ที่อาจเกิด exception
}

catch
{
  // Code ที่จัดการ exception
}

finally
{
  // Code ที่ต้องทำงานเสมอ ไม่ว่าจะเกิด exception หรือไม่ก็ตาม
}
```

Exception Types

Exception

IndexOutOfRangeException

SEHException

SystemException

AccessViolationException

NullReferenceException

ArgumentException

InvalidOperationException

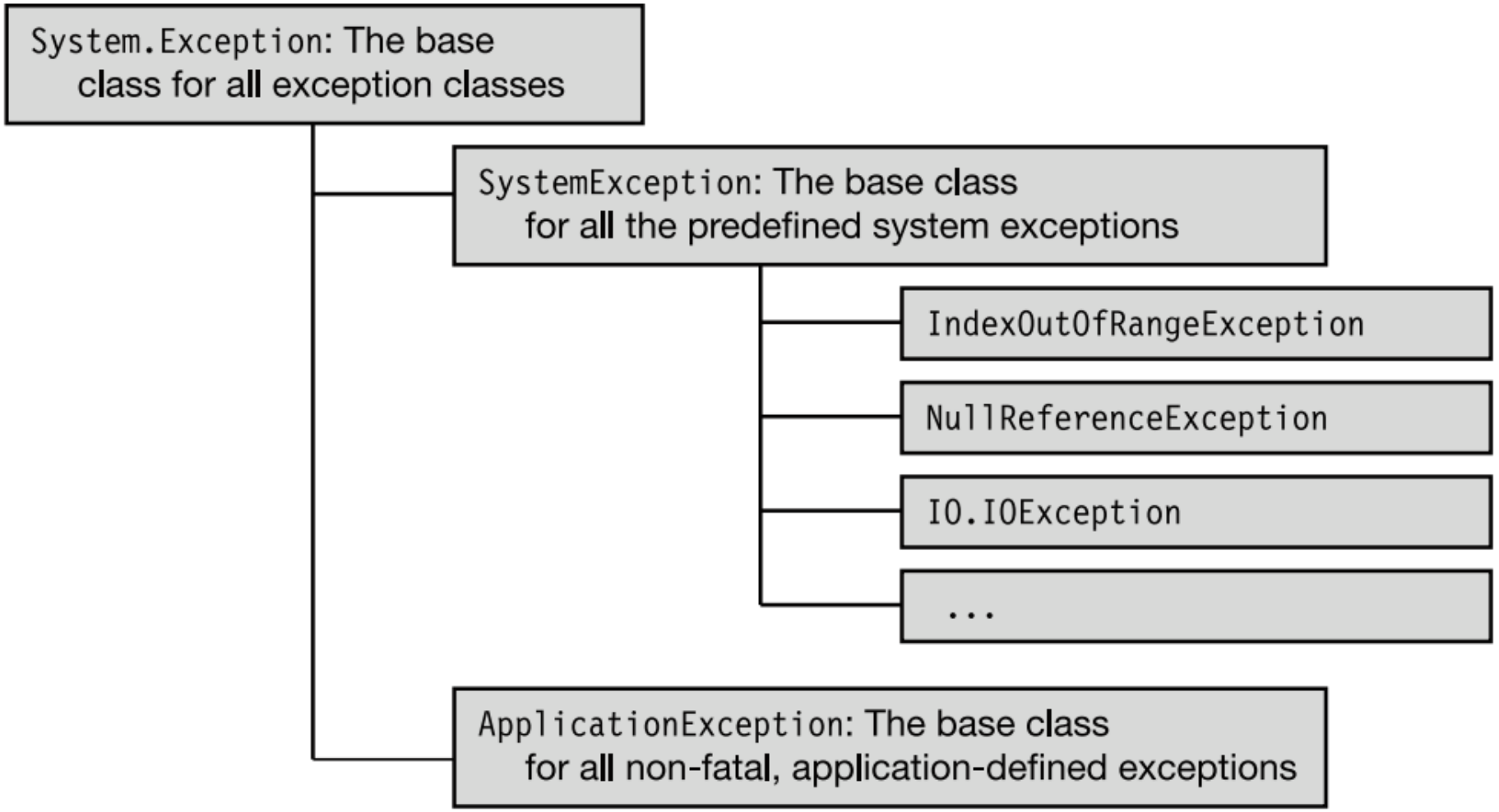
ExternalException

ArgumentNullException

ArgumentOutOfRangeException

frlrfSystemRuntimeInteropServicesCOMExceptionClassTopic

Structure of the exception hierarchy



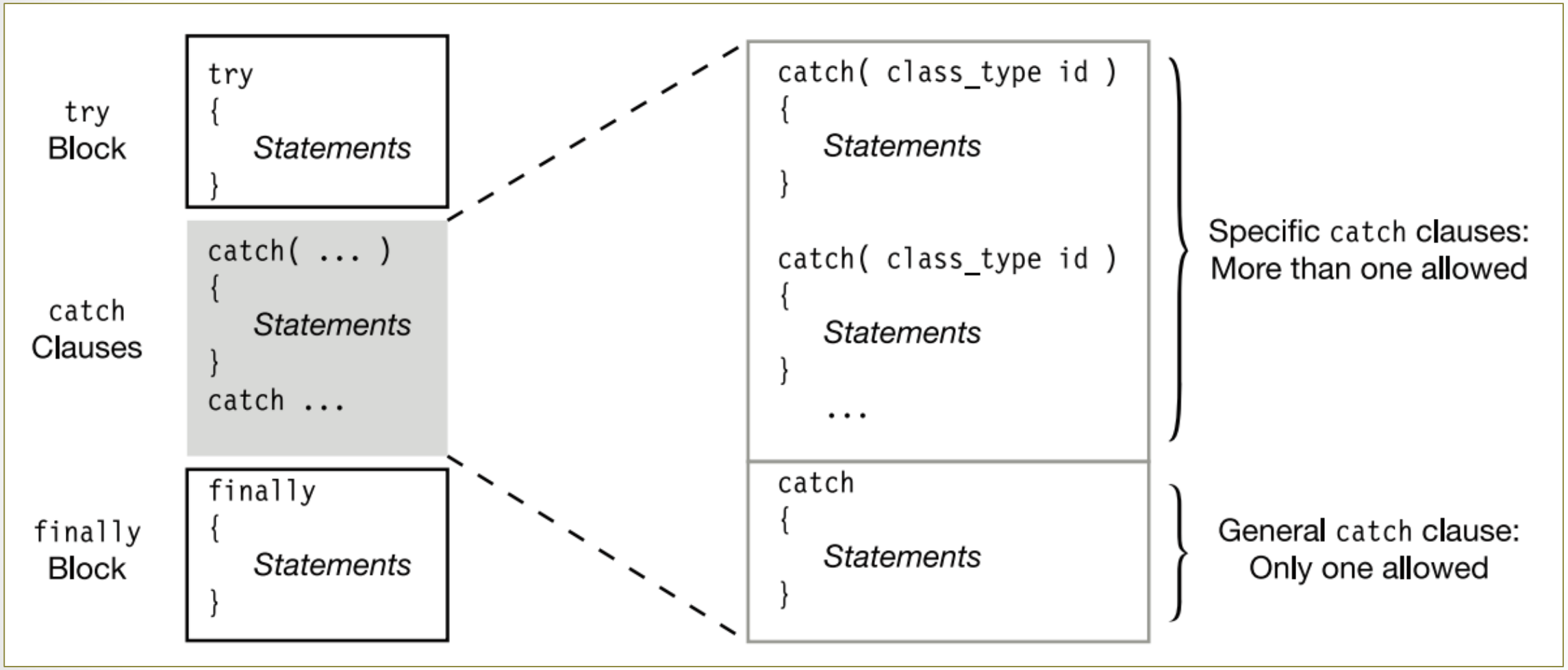
Catching exceptions (modified #1)

```
int x = 10;
try
{
   int y = 0;
   x /= y;                          // Raises an exception
}
                Exception type
                    ↓
catch ( DivideByZeroException )
{
      ...
   Console.WriteLine("Handling an exception.");
}
```

Catching exceptions (modified #2)

```
int x = 10;
try
{
   int y = 0;
   x /= y;                              // Raises an exception
}
                Exception type     Exception variable
                      ↓                  ↓
catch ( DivideByZeroException  e )
{                                  Accessing the exception variables
                                              ↓
   Console.WriteLine("Message: {0}", e.Message );
   Console.WriteLine("Source:  {0}", e.Source );
   Console.WriteLine("Stack:   {0}", e.StackTrace );
}
```

Structure of the catch clauses section of a try statement



สร้าง exceptions เองได้ไหม?

- เราสามารถสร้าง exception ขึ้นมาเองในโปรแกรม

```
throw new Exception ("Bang");
```

- เมื่อโปรแกรมทำงานถึงบรรทัด throw และไม่ได้เขียน catch จะเกิดอะไรขึ้น?

มารยาทในการใช้งาน Exception

- ในการใช้งาน Exception ควรใช้เมื่อจำเป็นจริงๆ
  - เช่นเมื่อตรวจพบว่าโปรแกรมมักจะทำงานผิดพลาดในตำแหน่งนั้น
  - เมื่อต้องใช้ทรัพยากรที่คาดว่าอาจจะไม่มีอยู่ หรือสามารถเคลื่อนย้ายได้
    - เช่น ไฟล์บน thumb drive, การเชื่อมต่อฐานข้อมูล, ฯลฯ

Extension Methods

Extension Methods

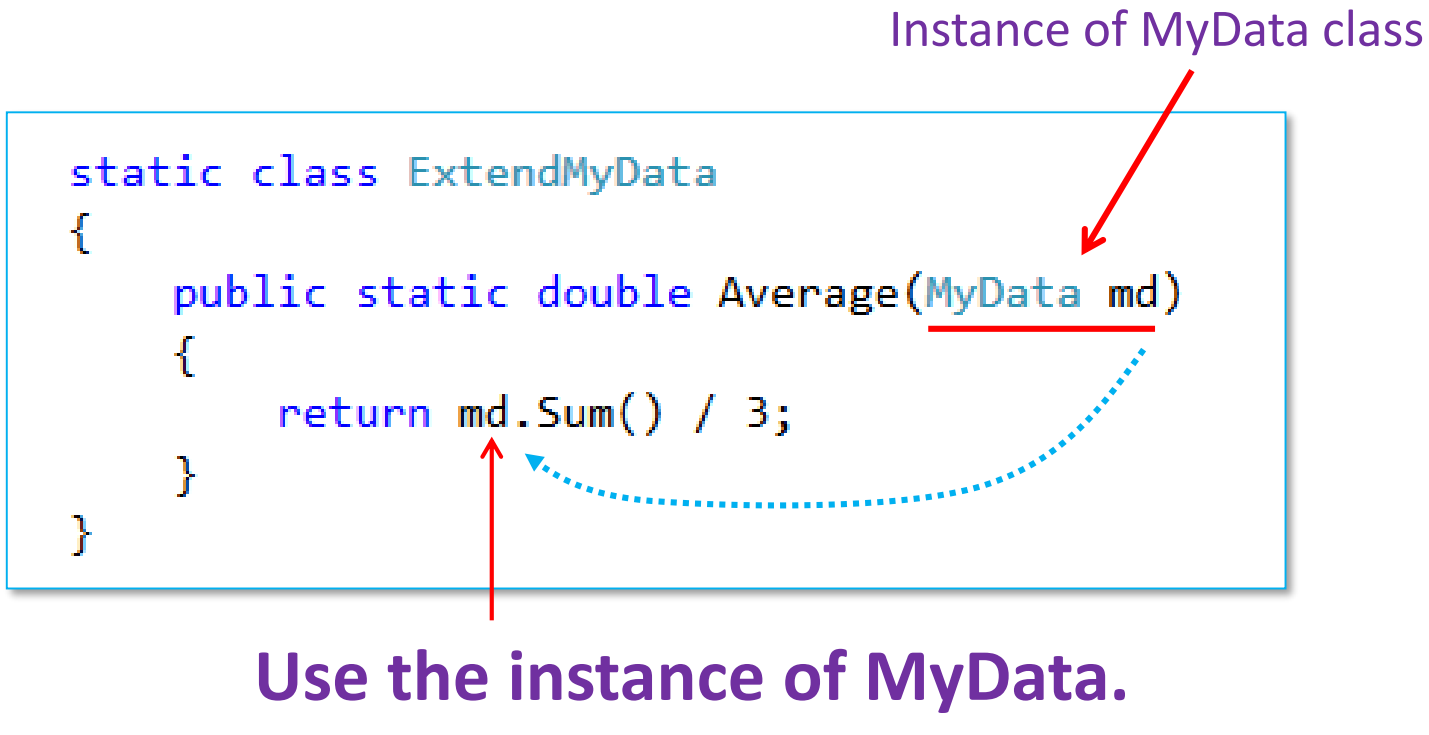
- Extension Methods คือ methods ที่เพิ่มความสามารถของคลาส โดยไม่ต้องสืบทอดคลาสนั้น
- แต่เราต้องสร้างคลาสใหม่ขึ้นมา เพื่อทำการดังกล่าว
- ไม่ต้องทำ extension methods ถ้า
  - สามารถแก้ไข source code ของ class ได้
  - สามารถเข้าถึงข้อมูลในคลาสได้

ตัวอย่างคลาส

```
class MyData
{
    private double D1; // Fields
    private double D2;
    private double D3;
    public MyData(double d1, double d2, double d3) // Constructor
    {
        D1 = d1; D2 = d2; D3 = d3;
    }
    public double Sum() // Method Sum
    {
        return D1 + D2 + D3;
    }
}
```

คลาสนี้มีข้อจำกัด เช่น ไม่สามารถหา**ค่าเฉลี่ย**ของ data ได้

การทำ extension methods

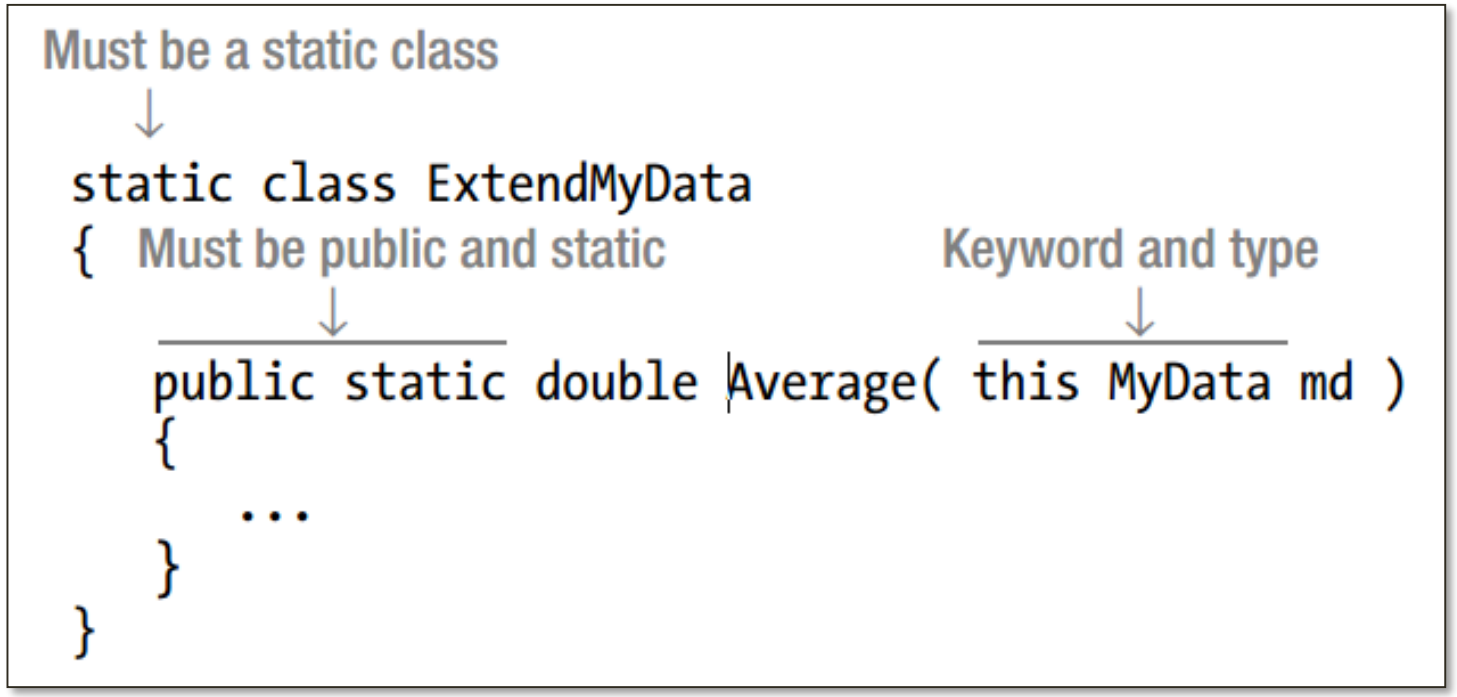


การเรียกใช้ extension class

```
class Program
{
    static void Main(string[] args)
    {
        MyData md = new MyData(3, 4, 5);
        Console.WriteLine("Average: {0}", ExtendMyData.Average(md));
    }
}
```

รูปแบบการเขียนอย่างนี้ ยังไม่เป็นธรรมชาติ

รูปแบบการเขียน extension class



ตัวอย่าง extension method จาก sealed class

```
sealed class MyData
{
    private double D1, D2, D3;
    public MyData(double d1, double d2, double d3)
    { D1 = d1; D2 = d2; D3 = d3; }
    public double Sum() { return D1 + D2 + D3; }
}

static class ExtendMyData
{
    public static double Average(this MyData md)
    {
        return md.Sum() / 3;
    }
}
```

ตัวอย่างโปรแกรม

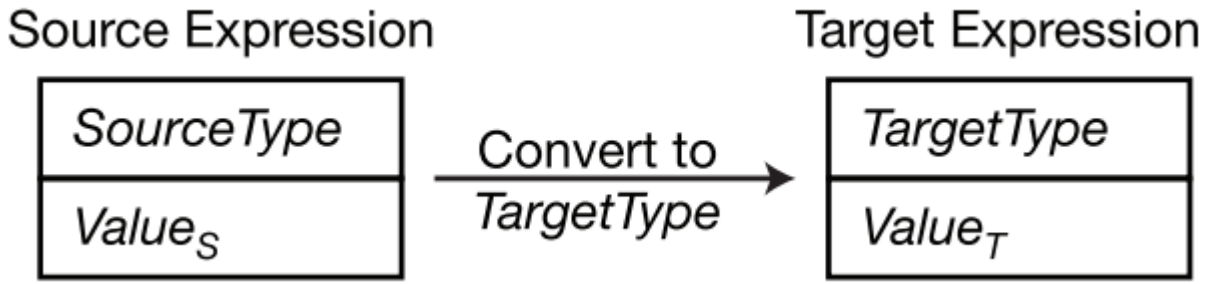
```
static void Main(string[] args)
{
    MyData md = new MyData(3, 4, 5);
    Console.WriteLine("Sum: {0}", md.Sum());
    Console.WriteLine("Average: {0}", md.Average());
}
```

รูปแบบการเขียนอย่างนี้ เป็นธรรมชาติ เหมือนการใช้งานคลาสตามปกติ

Conversions

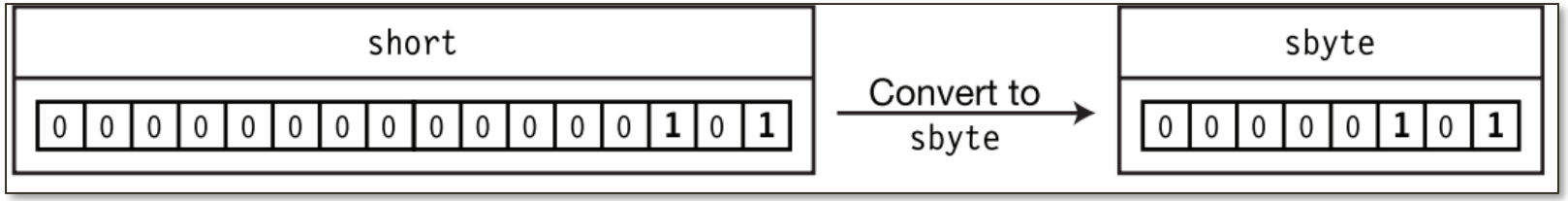
Conversions คืออะไร

- *เป็นกระบวนการสำหรับแปลงข้อมูลชนิดหนึ่ง (ต้นทาง) ให้ดูเสมือนว่าเป็นข้อมูลอีกชนิดหนึ่ง (ปลายทาง)*
- *ถ้าต้นทางและปลายทาง เป็นชนิดข้อมูลเดียวกัน ก็ไม่ต้องผ่านกระบวนการแปลงชนิด*



ตัวอย่างการแปลงชนิดข้อมูล

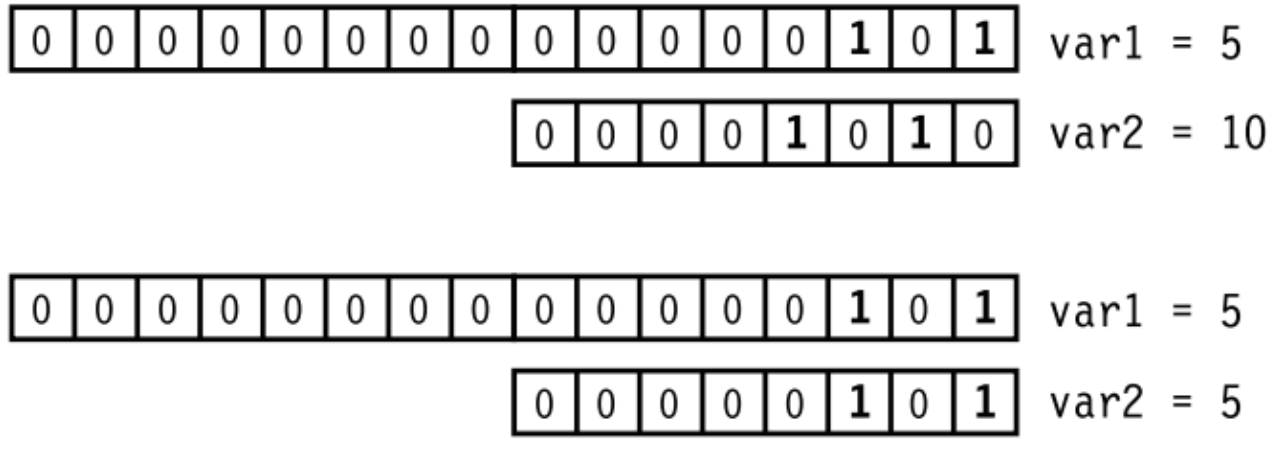
```
short var1 = 5;
sbyte var2;
Var2 = var1;
```



Cast expression

```
short  var1 = 5;
sbyte  var2 = 10;
   ...


var2 = (sbyte) var1;
```

Cast expression, which says to convert the value of var1 to type `sbyte`



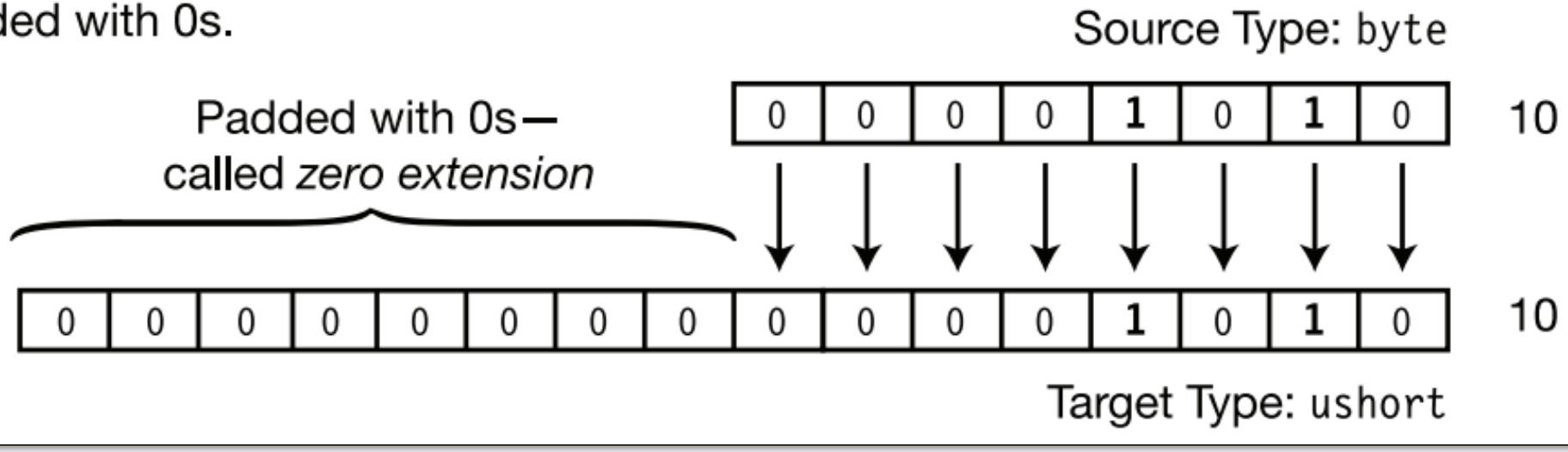
การแปลงโดยปริยาย (Implicit Conversions)

- การแปลงโดยปริยาย เป็นการแปลงที่ ตัวแปลภาษา ทำให้เอง โดยอัตโนมัติ
- เมื่อตัวแปรปลายทาง มีขนาดใหญ่กว่า (มีจำนวนบิตมากกว่า) ก็จะมีการเติมตัวเลข 0 หรือ 1 ลงไปในบริเวณที่เหลือ
- ถ้าเติมเลข 0 ลงไป เรียกว่าการทำ *zero extension*

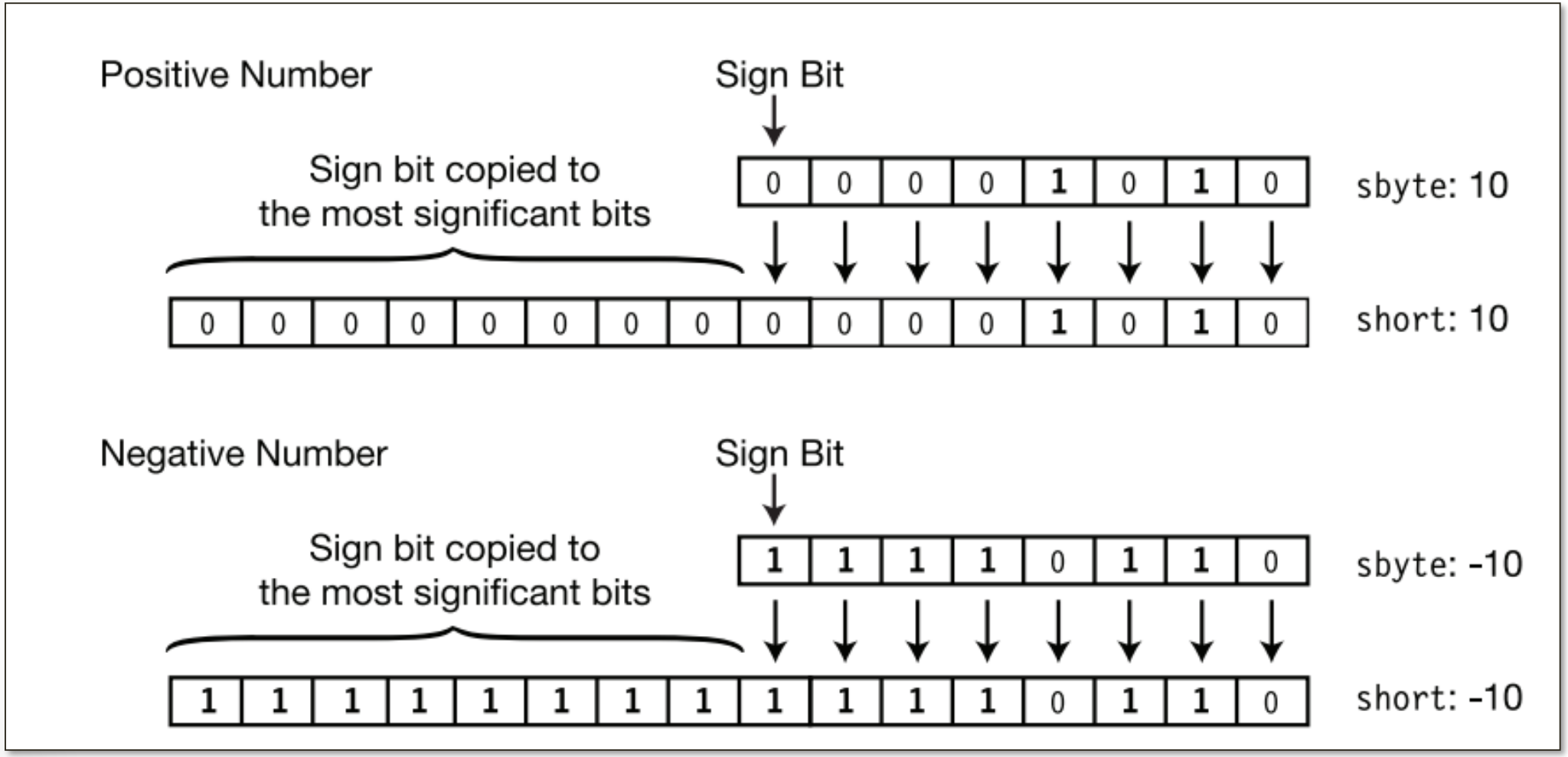
zero extension

zero extension สำหรับ unsigned conversions

The source value fits fine in the target type, and the most significant eight bits of the target are padded with 0s.



Sign extension in signed conversions

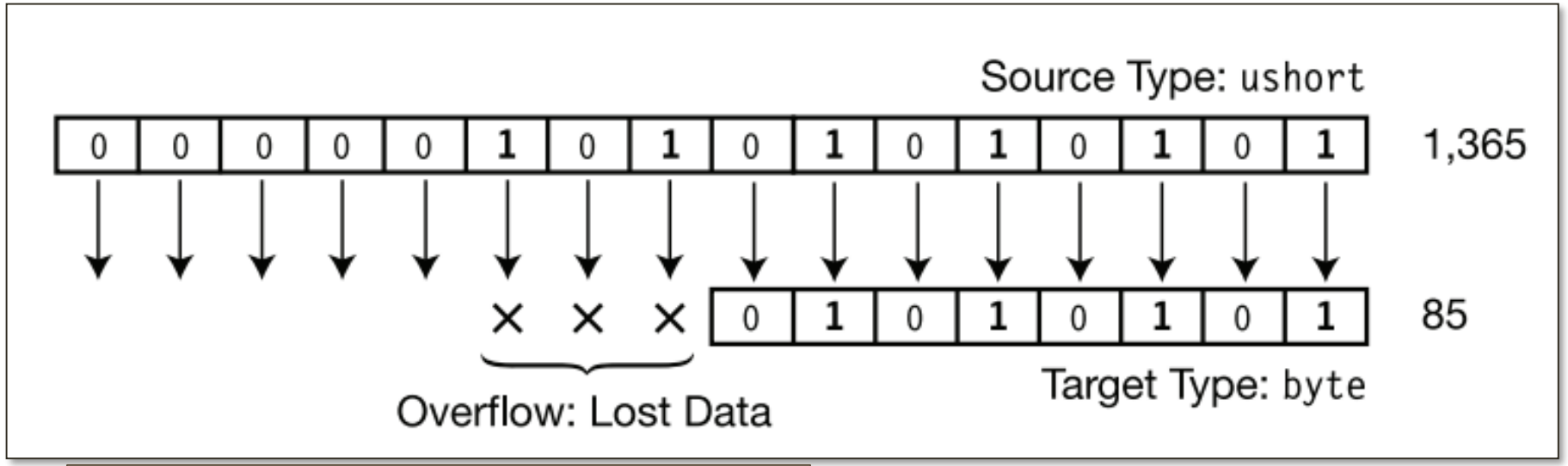


Explicit Conversions and Casting

ushort สามารถเก็บข้อมูลในช่วง 0 และ 65,535

byte สามารถเก็บข้อมูลในช่วง 0 และ 255

จะเป็นอย่างไรถ้านำค่าที่เก็บใน ushort มาใส่ใน byte?

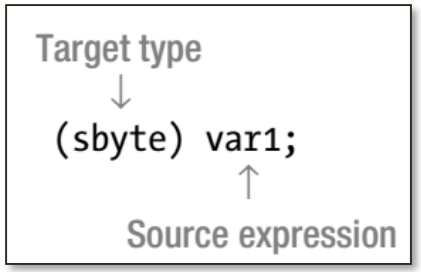


$$\frac{255}{65535} = 0.00389 \approx 0.4\%$$

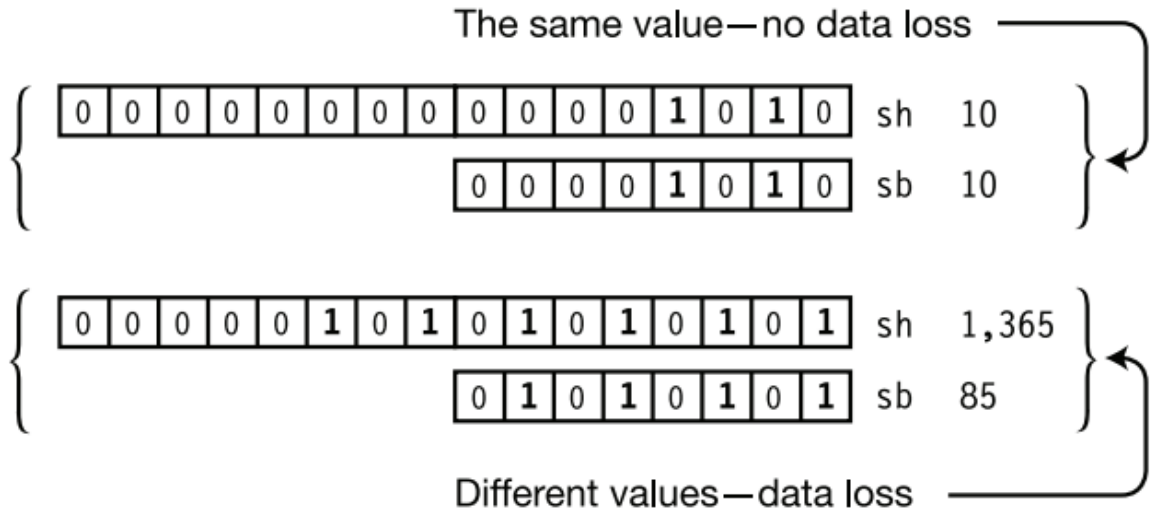
⇐ แปลงได้โดยปลอดภัย

Casting

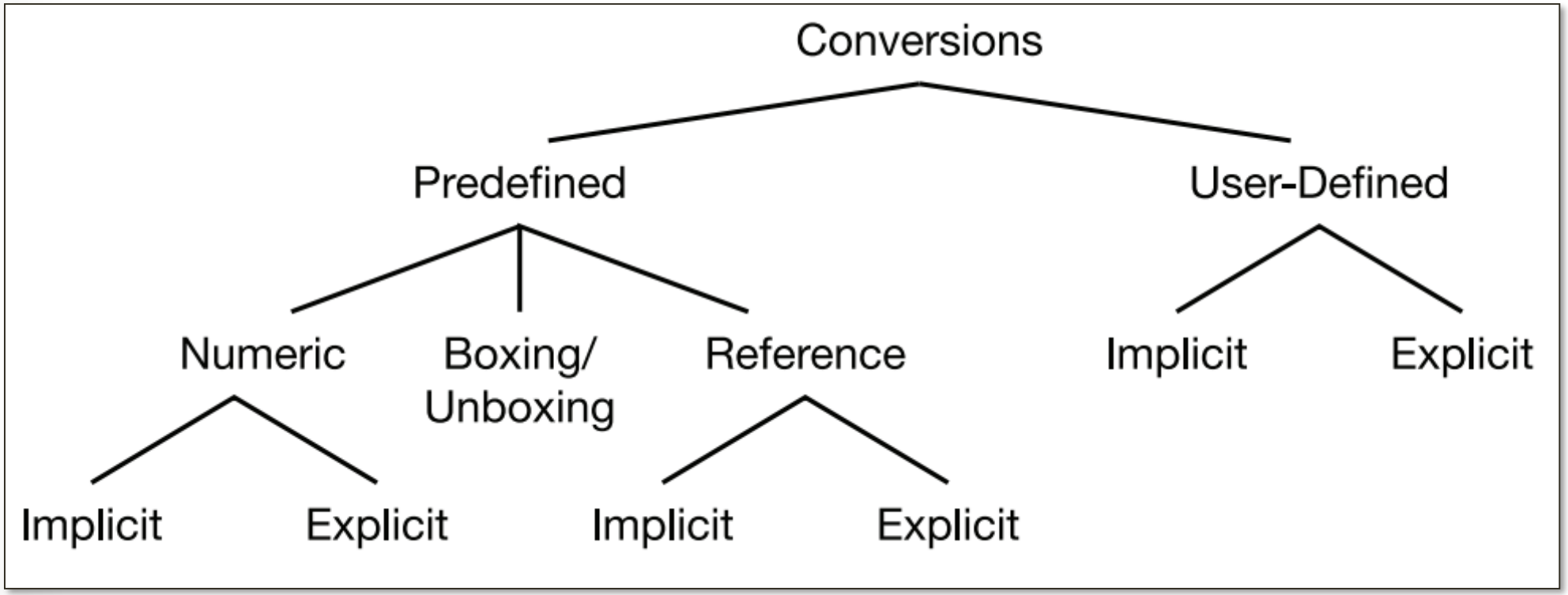
- ทำได้โดยกการระบุชนิดของตัวแปรปลายทาง ไว้ในวงเล็บ หน้าตัวแปรต้นทาง



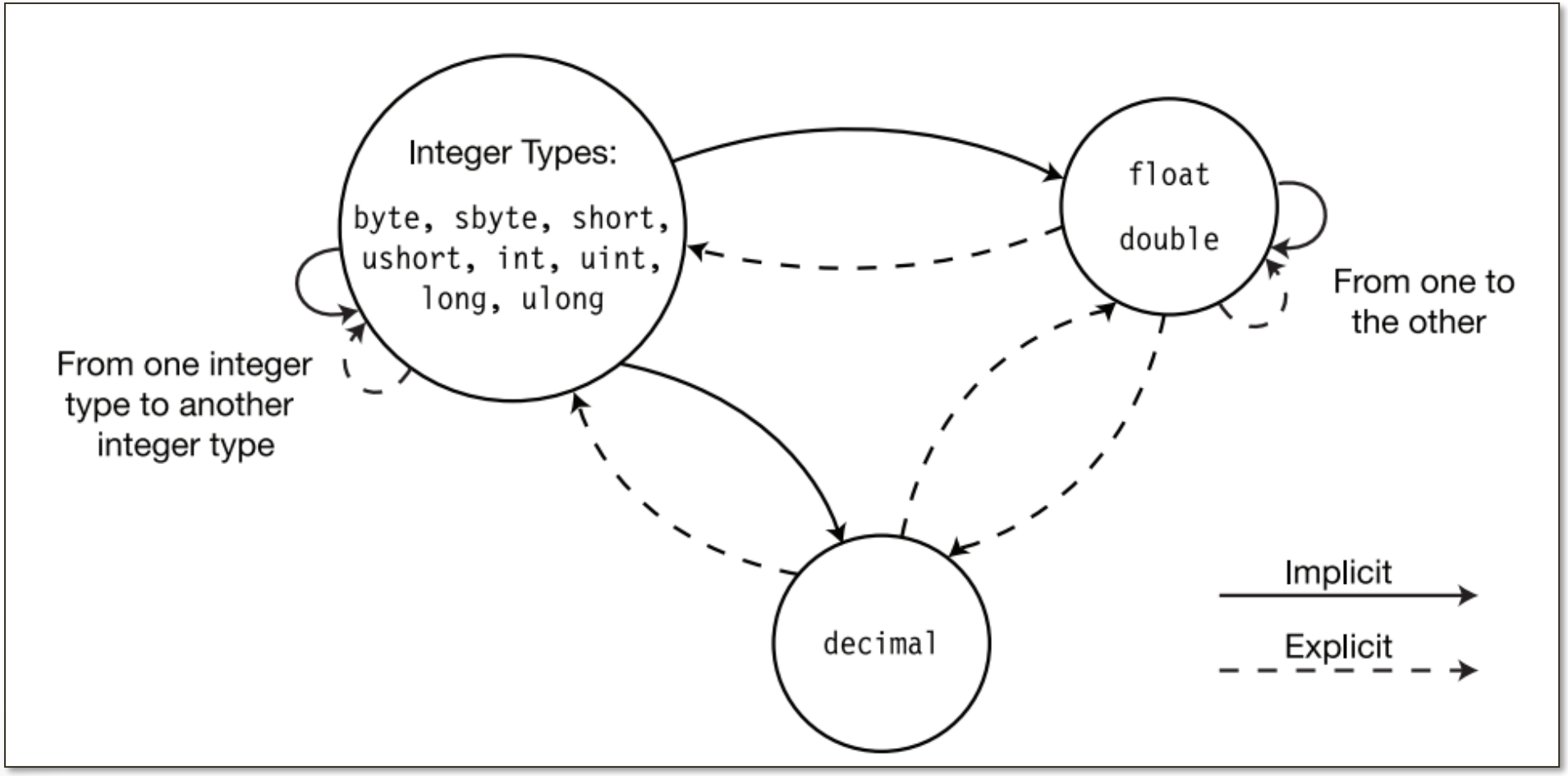

```
ushort sh = 10;
byte sb = (byte) sh;
Console.WriteLine
     ("sb:  {0} = 0x{0:X}", sb);

sh = 1365;
sb = (byte) sh;
Console.WriteLine
     ("sb:  {0} = 0x{0:X}", sb);
```



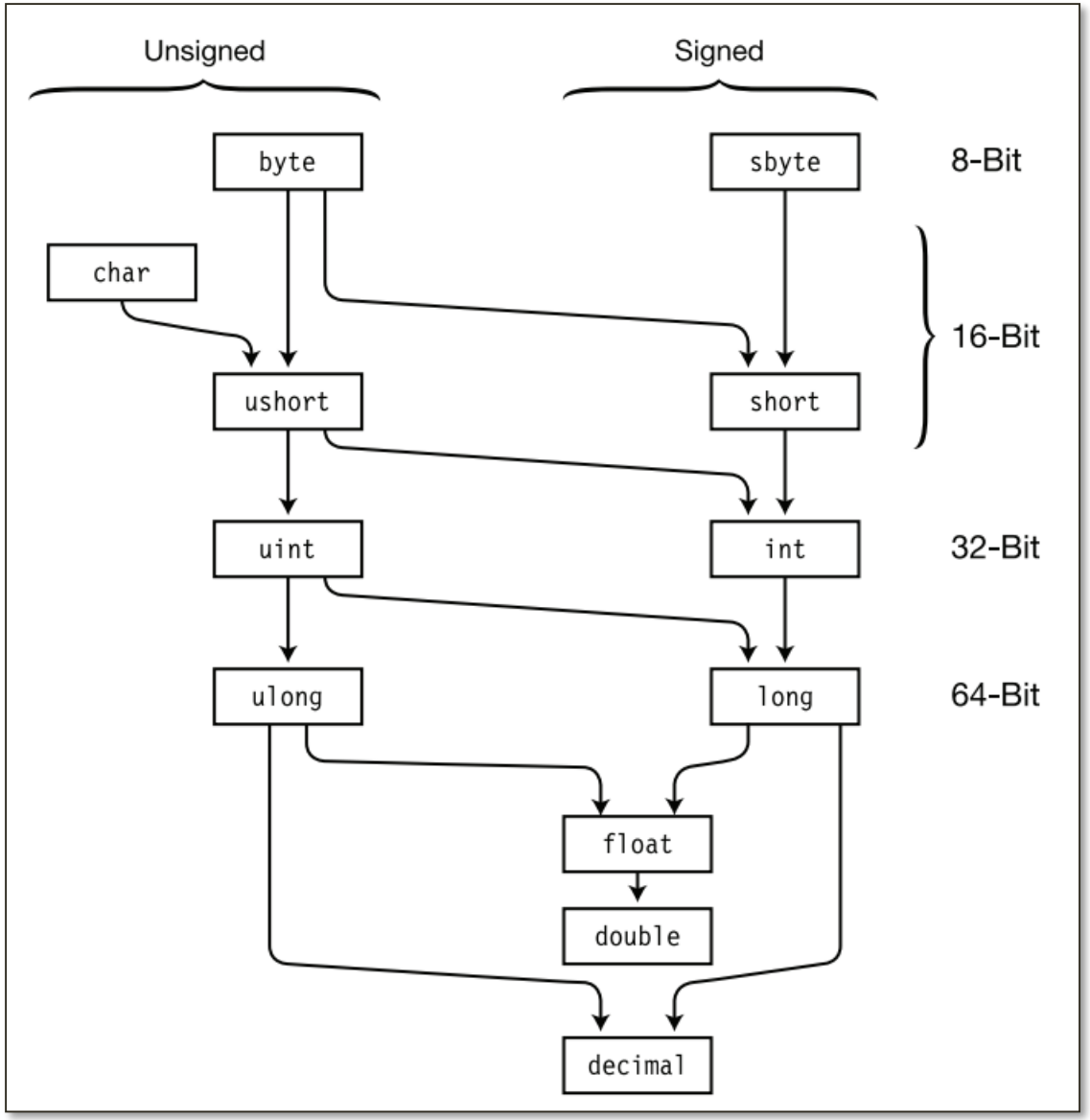
Types of Conversions



Numeric Conversions



Implicit Numeric Conversions



The checked and unchecked Operators

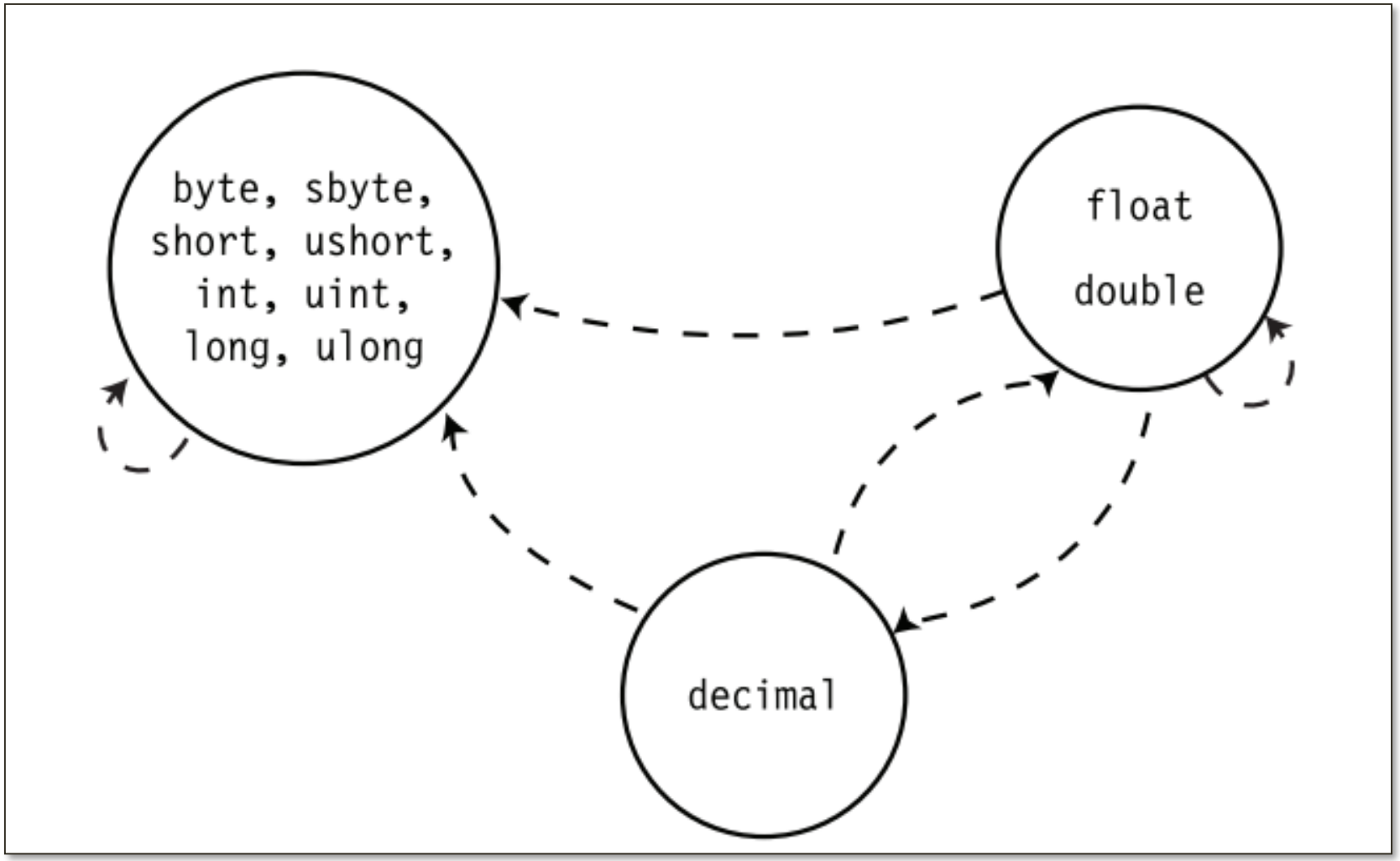
```
checked   ( Expression )
unchecked ( Expression )
```

```
ushort sh = 2000;
byte   sb;

sb = unchecked ( (byte) sh );          // Most significant bits lost
Console.WriteLine("sb: {0}", sb);

sb =   checked ( (byte) sh );          // OverflowException raised
Console.WriteLine("sb: {0}", sb);
```

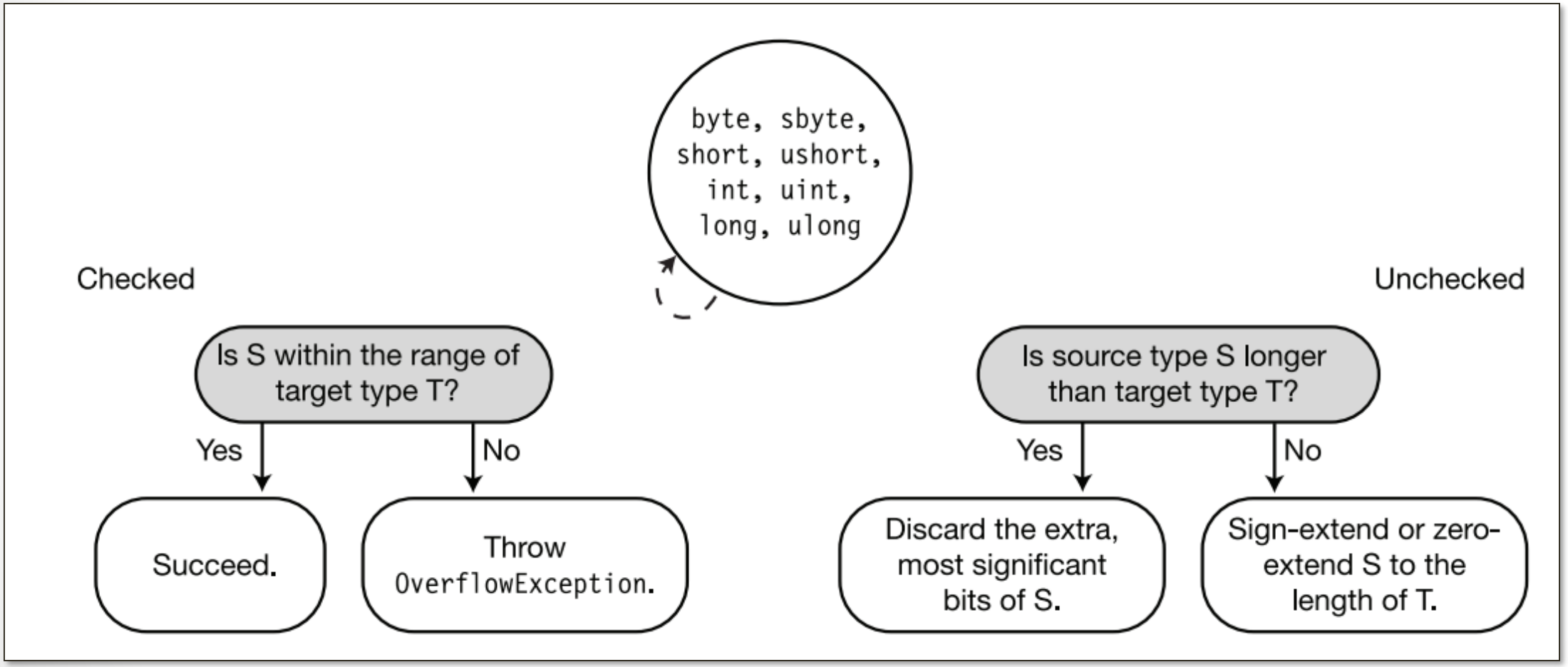
The checked and unchecked Statements

```
byte   sb;
ushort sh = 2000;

unchecked                                          // Set unchecked
{
   sb = (byte) sh;
   Console.WriteLine("sb: {0}", sb);

   checked                                         // Set checked
   {
      sb = (byte) sh;
      Console.WriteLine("sb: {0}", sh);
   }
}
```

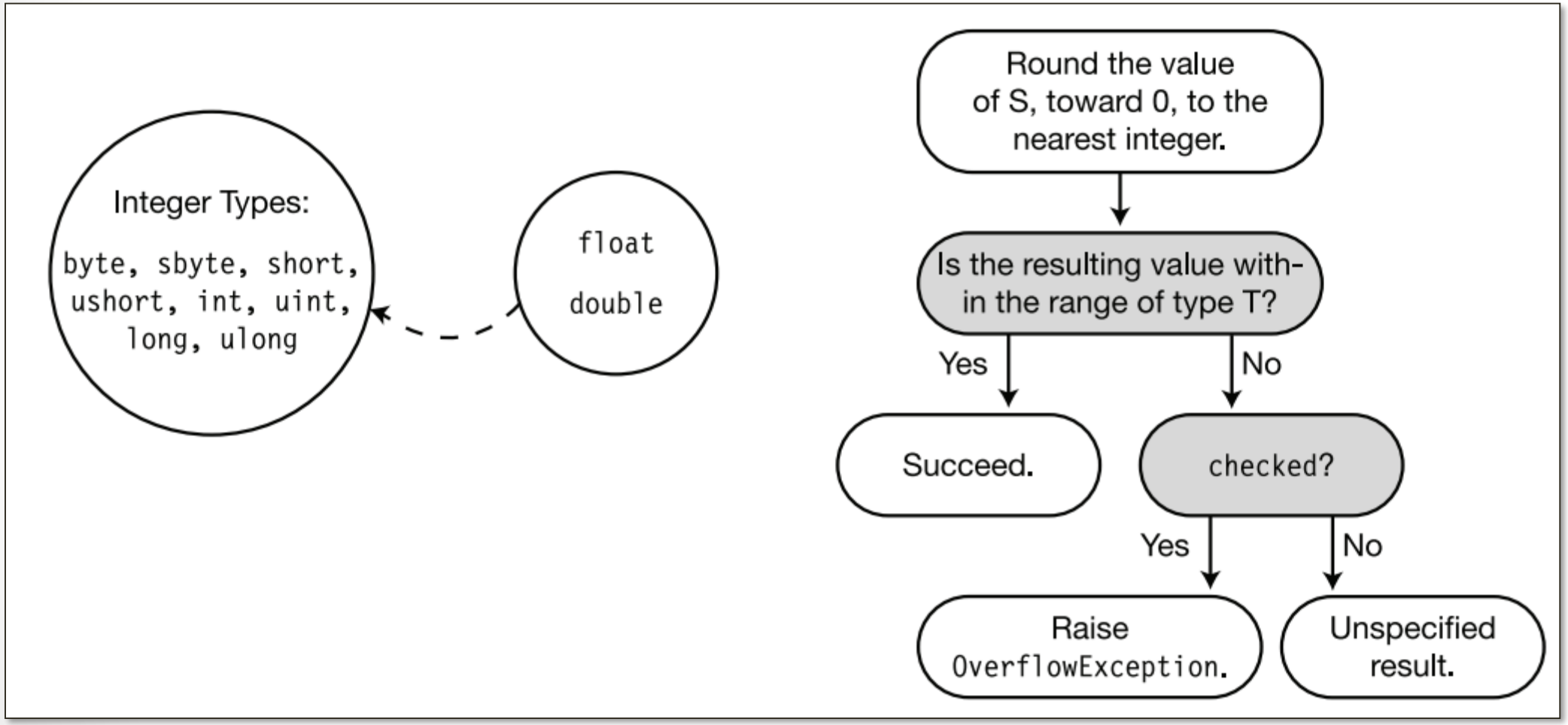
Explicit Numeric Conversions



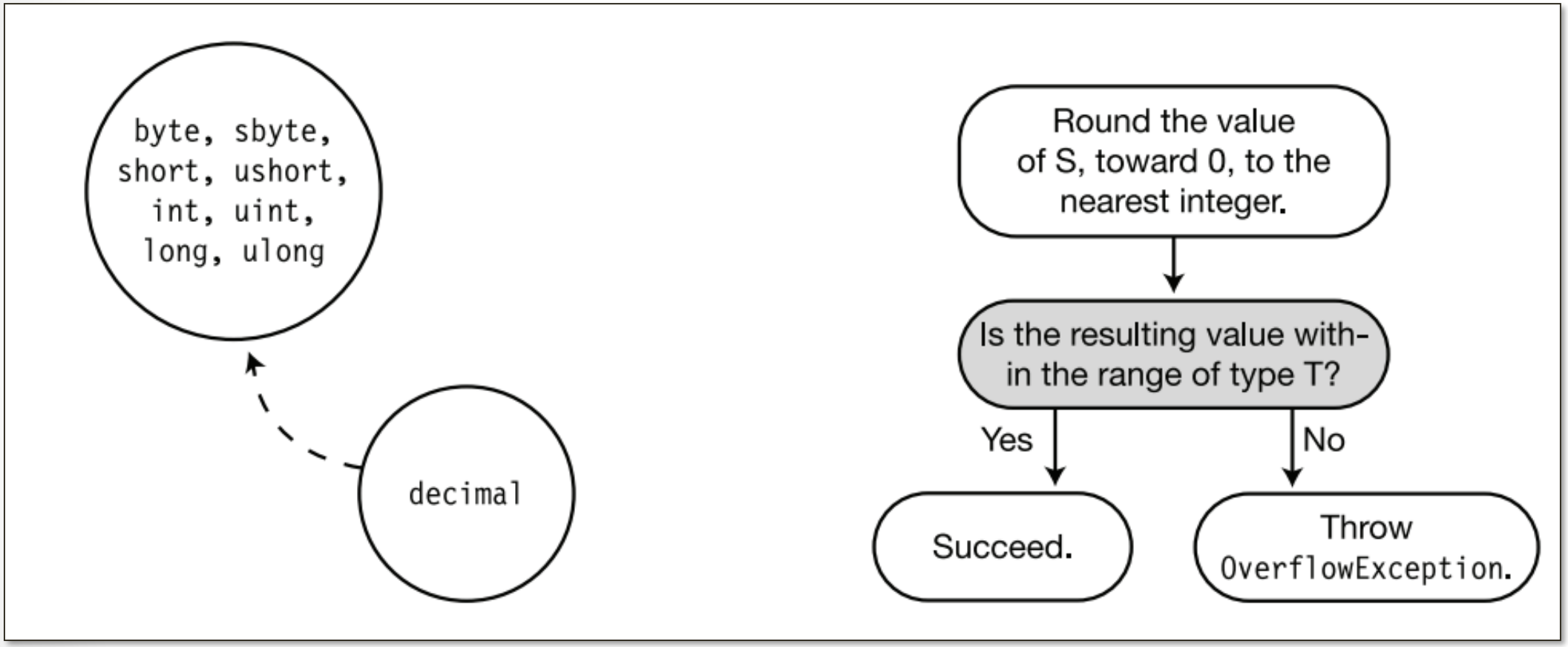
Integer Type to Integer Type



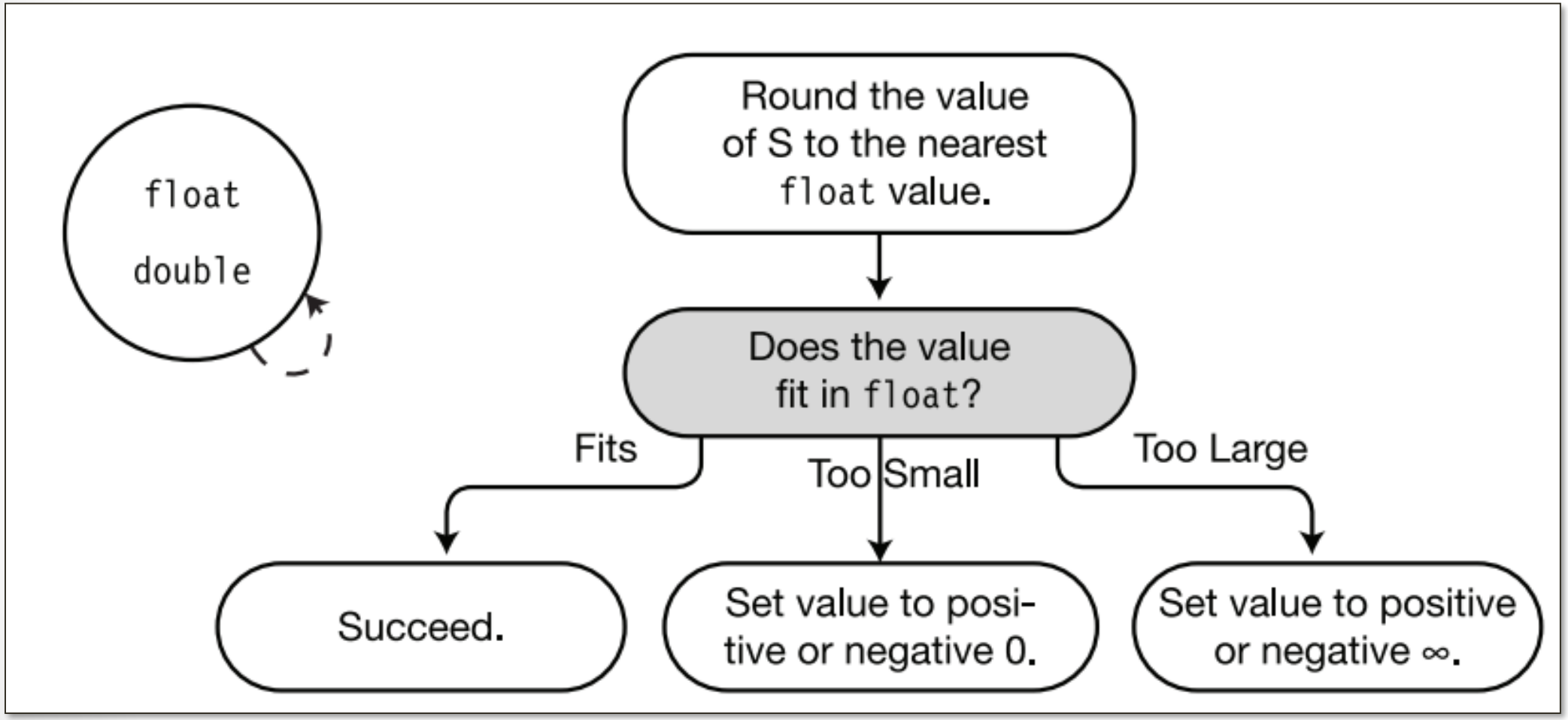
float or double to Integer Type



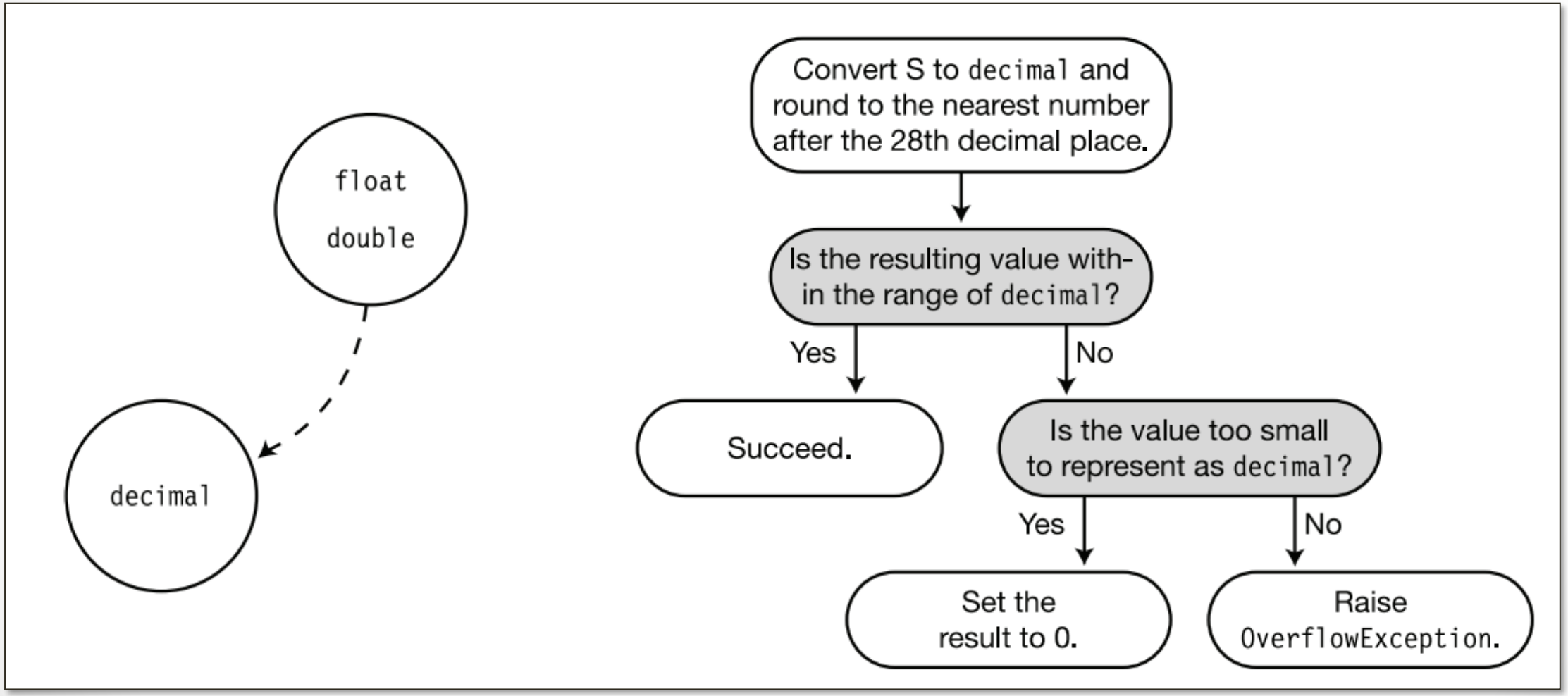
decimal to Integer Type



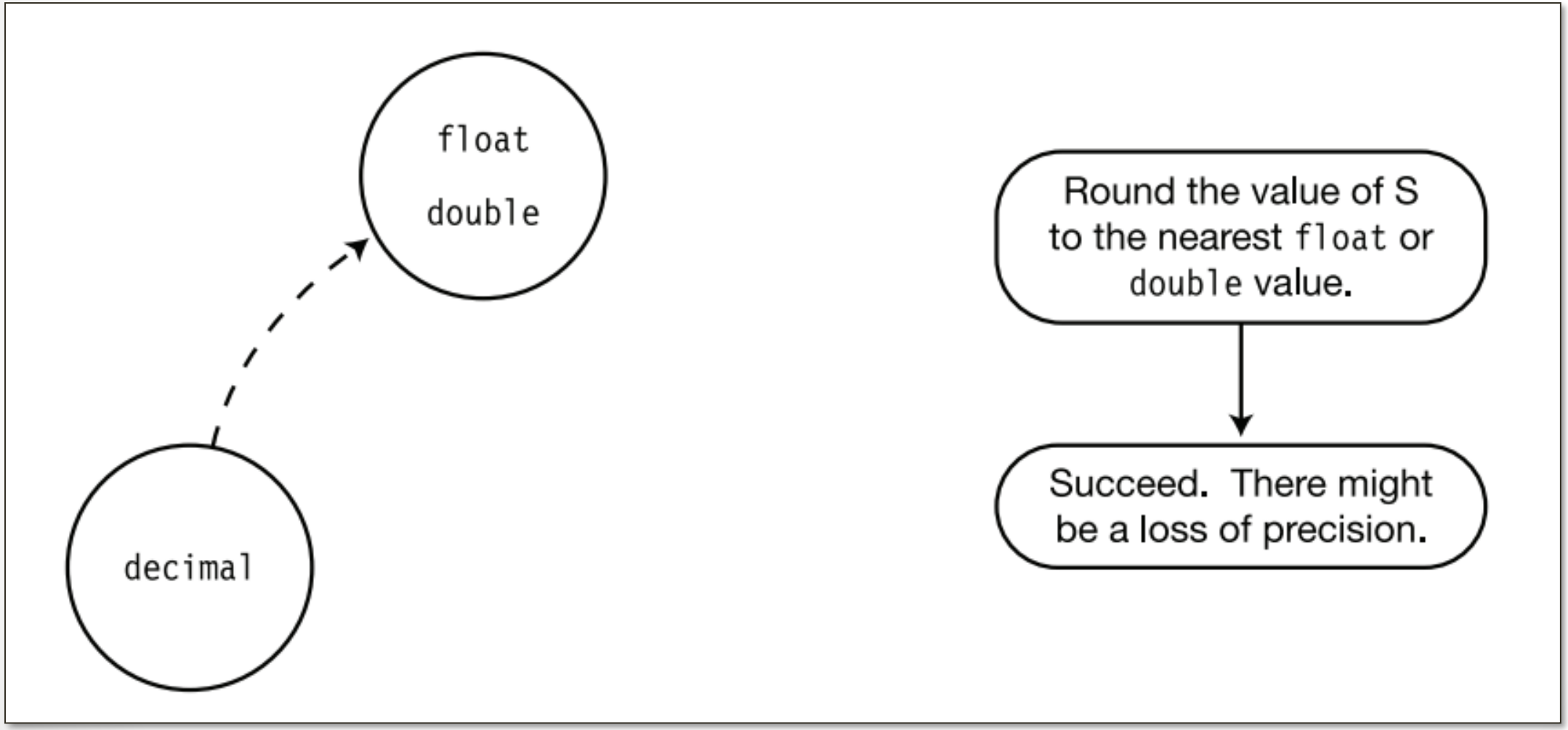
double to float



float or double to decimal



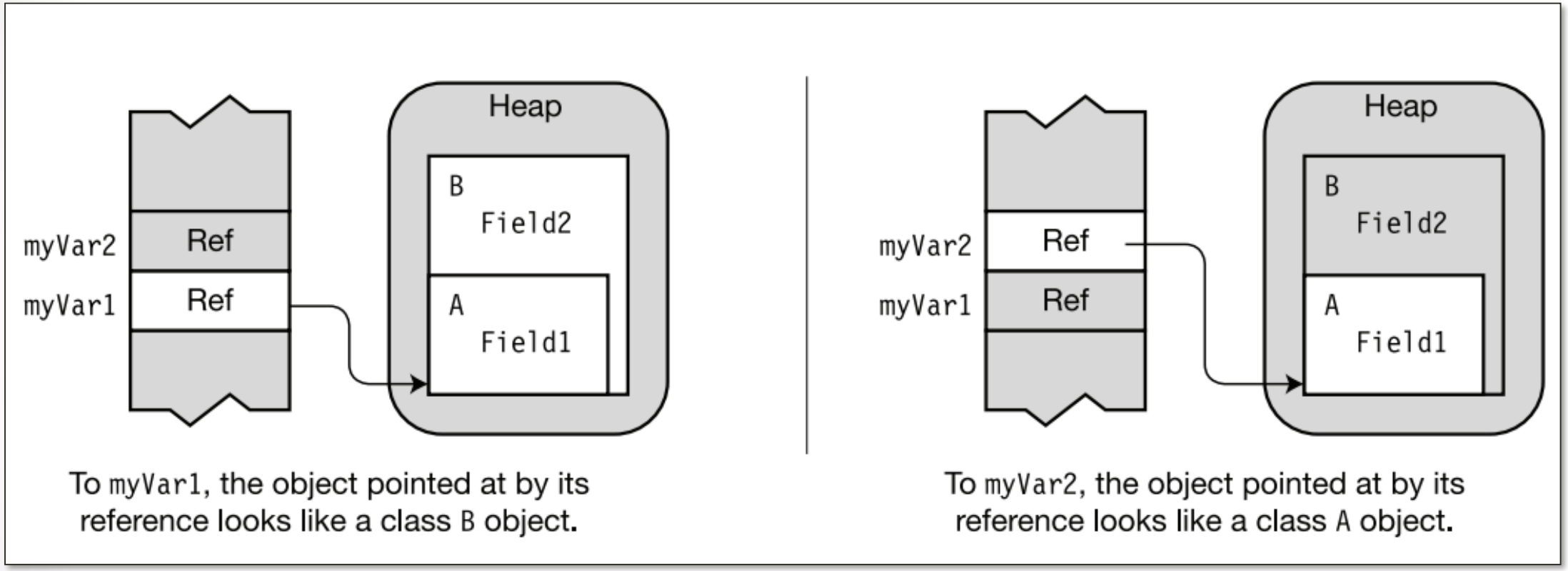
decimal to float or double



Reference Conversions

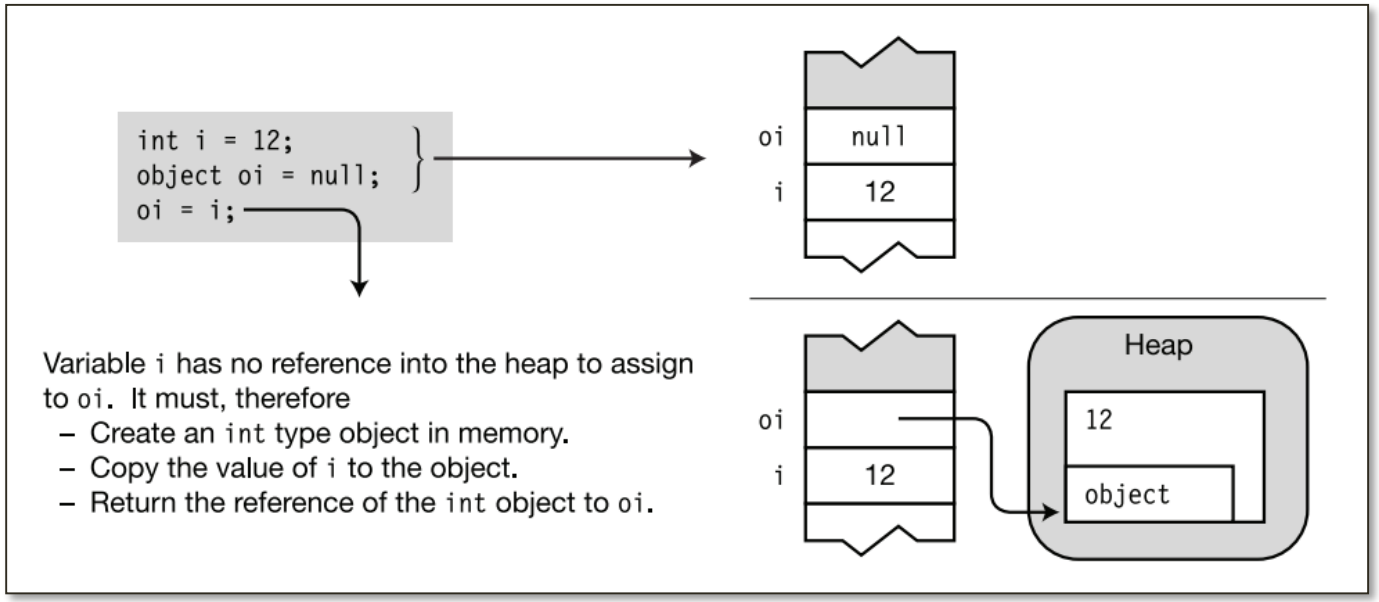
- ตัวแปร Reference คือ ตัวแปรที่มีองค์ประกอบ 2 ส่วน โดยส่วนแรก เป็นตัวแปรบน stack และส่วนที่สองเป็นวัตถุที่อยู่ใน heap
- reference conversion เป็นการแปลงตัวแปรบน stack ให้เป็นชนิดอื่น ในขณะที่ยังคงชี้ไปยังวัตถุเดิมใน heap

```
class A    { public int Field1; }

class B: A { public int Field2; }

class Program
{
   static void Main( )
   {
      B myVar1 = new B();
Return the reference to myVar1 as a reference to a class A.
                  ↓
      A myVar2 = (A) myVar1;

      Console.WriteLine("{0}", myVar2.Field1);          // Fine
      Console.WriteLine("{0}", myVar2.Field2);          // Compile error!
   }                                    ↑
}                                  myVar2 can't see Field2.
```


Heap
B
Field2
A
Field1
myVar2
Ref
myVar1
Ref
To myVar1, the object pointed at by its reference looks like a class B object.
Heap
B
Field2
A
Field1
myVar2
Ref
myVar1
Ref
To myVar2, the object pointed at by its reference looks like a class A object.

Boxing Conversions

- การแปลงจาก value type ไปเป็น reference type เรียกว่า boxing conversion
- เนื่องจาก reference type จะเก็บค่าของข้อมูลใน heap ดังนั้น ในการแปลงจะมีกระบวนการดังนี้
  - 1. สร้าง “วัตถุ” ใน heap
  - 2. คัดลอกค่าตัวแปร value type ไปยัง heap
  - 3. ส่งที่อยู่ของ “วัตถุ” ไปเก็บในตัวแปร reference type



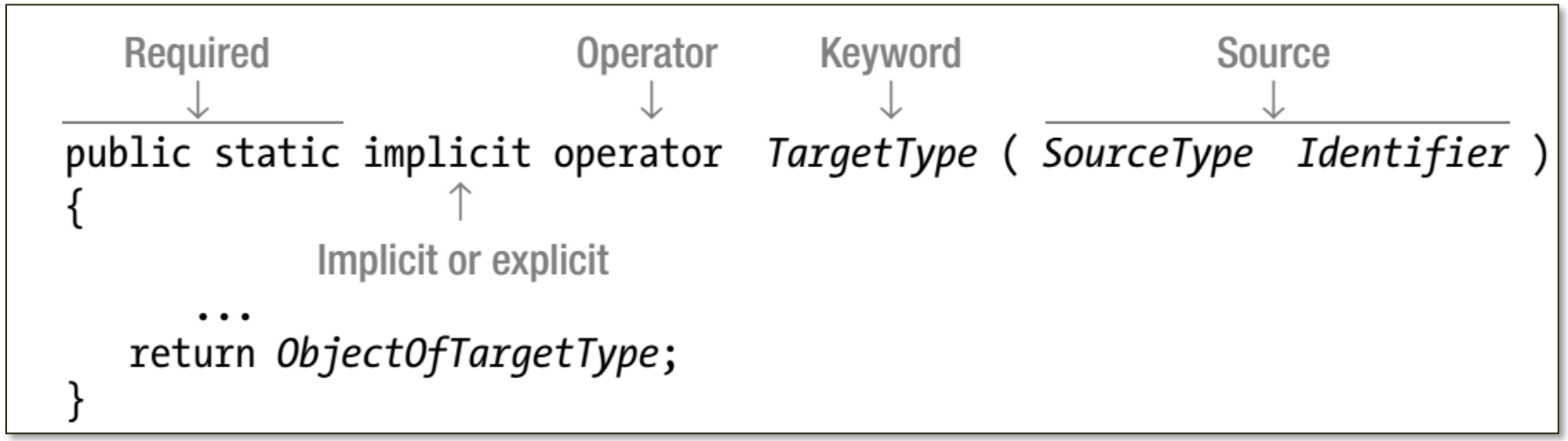
Unboxing Conversions

- เป็นกระบวนการแปลง boxed object ไปเป็น value type

```
static void Main()
{
    int i = 10;
      Box i and assign its reference to oi.
              ↓
    object oi = i;
         Unbox oi and assign its value to j.
              ↓
    int j = (int) oi;
    Console.WriteLine("i: {0},   oi: {1},   j: {2}", i,  oi, j);
}
```

User-Defined Conversions

รูปแบบ



ตัวอย่างการใช้

```
public static implicit operator int(Person p)
{
    return p.Age;
}
```

Example of a User-Defined Conversion

แบบปริยาย (Implicit)

```
class Person
{
   public string Name;
   public int    Age;
   public Person(string name, int age)
   {
      Name = name;
      Age = age;
   }

   public static implicit operator int(Person p)   // Convert Person to int.
   {
      return p.Age;
   }

   public static implicit operator Person(int i)   // Convert int to Person.
   {
      return new Person("Nemo", i);        // ("Nemo" is Latin for "No one".)
   }
}
```

```
class Program
{
   static void Main( )
   {
      Person bill = new Person( "bill", 25);

      int age = bill;
      Console.WriteLine("Person Info: {0}, {1}", bill.Name, age);

      Person anon = 35;
      Console.WriteLine("Person Info: {0}, {1}", anon.Name, anon.Age);
   }
}
```

Convert a Person object to an int.

Convert an int to a Person object.

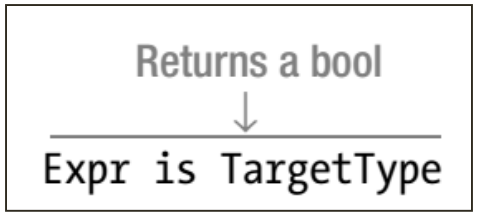
Example of a User-Defined Conversion

แบบแจ้งชัด (Explicit)

```
                        Explicit
                           ↓
        ...
public static explicit operator int( Person p )
{
   return p.Age;
}

...

static void Main( )
{
        ...    Requires cast expression
                           ↓
   int age = (int) bill;
        ...
```

The is Operator

- เนื่องจากในบางครั้ง เราไม่แน่ใจว่า การ conversion จะประสบผลสำเร็จหรือไม่ ซึ่งการ conversion ที่ไม่สำเร็จ อาจทำให้เกิด exception ได้
- ทางออกคือการใช้ is operator



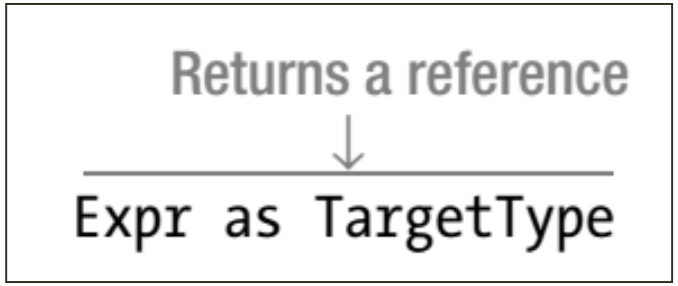
- is operator ใช้ได้กับ
  - reference conversion
  - boxing conversion
  - unboxing conversion

ตัวอย่าง is operator

```
class Employee : Person { }
class Person
{
   public string Name = "Anonymous";
   public int Age     = 25;
}

class Program
{
   static void Main()
   {
      Employee bill = new Employee();
      Person p;

      // Check if variable bill can be converted to type Person
      if( bill is Person )
      {
         p = bill;
         Console.WriteLine("Person Info: {0}, {1}", p.Name, p.Age);
      }
   }
}
```

The as Operator

- ใช้งานคล้ายกับ cast operator แต่จะมีกลไกป้องกันไม่ให้เกิด exception
- ถ้าไม่สามารถแปลงได้ จะส่งค่า null กลับมา



ตัวอย่าง as Operator

```
class Employee : Person { }

class Person
{
   public string Name = "Anonymous";
   public int Age     = 25;
}

class Program
{
   static void Main()
   {
      Employee bill = new Employee();
      Person p;

      p = bill as Person;
      if( p != null )
      {
         Console.WriteLine("Person Info: {0}, {1}", p.Name, p.Age);
      }
   }
}
```

ข้อจำกัดของ is และ as operator

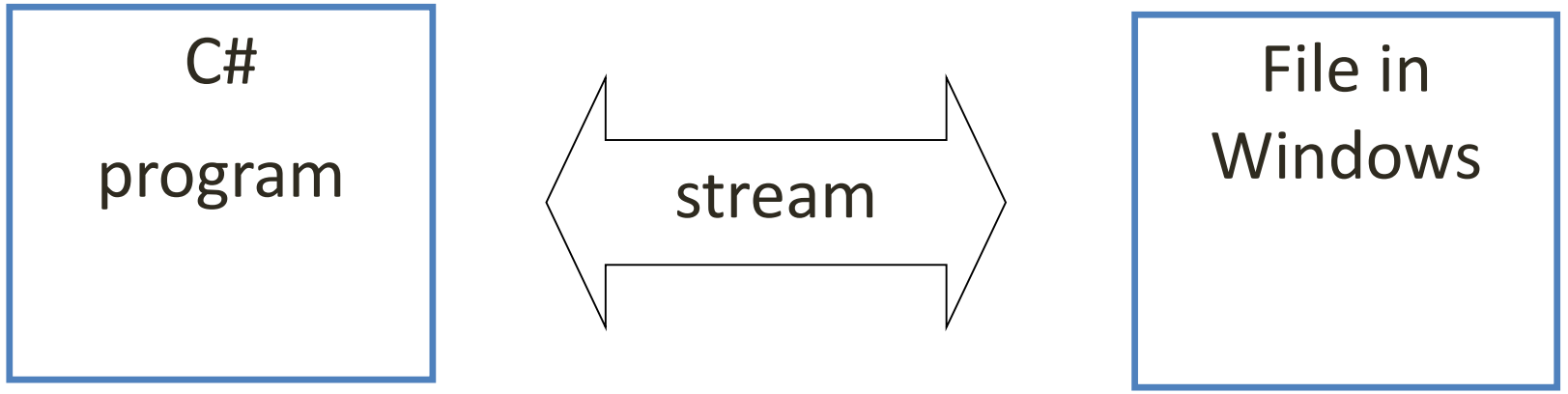
- ใช้ได้กับ
  - Reference conversion
  - Boxing conversion
  - Unboxing conversion
- ใช้ไม่ได้กับ user-defined conversion

File Handling

Files

- ข้อมูลต่างๆ จะหายไป เมื่อปิดโปรแกรม
- เราสามารถบันทึกลงแฟ้มข้อมูลก่อนที่จะปิดโปรแกรม เพื่อรักษาข้อมูลไว้บนสื่อเก็บข้อมูล
- ระบบปฏิบัติการ จะรับผิดชอบเรื่องการเขียนข้อมูลลงสื่อต่างๆ
- ในภาษา C# จะมี library รองรับเรื่องการทำงานกับแฟ้มข้อมูลไว้ให้แล้ว

Files and Streams



- ในการเขียนข้อมูลลงในไฟล์ เราใช้ stream
- stream จะช่วยให้เขียนข้อมูลลงในสื่อทุกชนิดในรูปแบบเดียวกันหมด (files, ports, network)

Creating a Stream

```
StreamWriter writer ;
writer = new StreamWriter(“hello.txt");
```

- บรรทัดแรก เป็นการสร้าง StreamWriter reference ชื่อ writer
  - StreamWriter มีความสามารถเขียนข้อมูลลงไฟล์
- บรรทัดที่สอง เป็นการสร้างวัตถุ stream แล้วอ้างถึงโดย writer
  - วัตถุ stream จะเชื่อมไปยังไฟล์ที่ชื่อ hello.txt
  - ไฟล์ชื่อ hello.txt ตั้งอยู่ที่ไหน?

Writing to a Stream

```
writer.WriteLine("hello world");
```

- StreamWriter มีเมธอดต่างๆ
  - จากตัวอย่าง จะเห็นว่ามี WriteLine เหมือนกับใน Console.WriteLine
- **โปรแกรมตัวอย่างนี้จะทำงานอย่างไร?**

Closing a Stream

```
writer.Close();
```

- ทำไมต้องปิด stream ทุกครั้งที่เลิกใช้?
  - อาจเขียนข้อมูลลงไฟล์ไม่ครบ
  - อาจเรียกใช้ไฟล์จากโปรแกรมอื่นๆ ไม่ได้

คลาส Stream

- คลาส Straem เป็นส่วนหนึ่งของ C# Library
- อยู่ใน namespace ที่ชื่อ System.IO

```
using System.IO;
```

- หรือถ้าไม่ใช้ using เราต้องใช้แบบนี้

```
System.IO.StreamWriter writer;
```

File and Directories/Folders

- ระบบปฏิบัติการมีการเก็บระบบไฟล์เป็น directory
  - ไม่ได้เก็บทั้งหมดในที่เดียวกัน... ทำไม?
- เราเรียกว่า directory หรือ folder
  - ต่างกันตรงไหน?

Path

```
c:\data\2007\november\sales.txt
```

- Path คืออะไร?
- ในภาษา C# มีปัญหาเรื่องการเขียน \
  - เนื่องจาก C# มอง \ เป็น escape character
  - เราต้องใช้

```
string path;
path = “c:\\data\\2007\\november\\sales.txt”
```

A Path as a string

```
string path;
path = @"c:\data\2007\november\sales.txt";
```

- ใน C# เราสามารถใช้ @ เพื่อไม่ต้องเขียน \\

File reading

```
TextReader reader =
                    new StreamReader("Test.txt");
string line = reader.ReadLine();
Console.WriteLine (line);
reader.Close();
```

- อ่านไฟล์จาก Test.txt ทีละบรรทัด
- แล้วพิมพ์ออกทาง console

Reading an Entire File

```
StreamReader reader;
reader = new StreamReader("Test.txt");
while (reader.EndOfStream == false)
{
    string line = reader.ReadLine();
    Console.WriteLine(line);
}
reader.Close();
```

- EndOfStream ใช้ตรวจสอบว่าจบไฟล์หรือยัง

สรุป (files)

- เราใช้ stream เพื่อเชื่อมต่อระหว่างโปรแกรมและไฟล์บนสื่อเก็บข้อมูลต่างๆ
- stream ใช้ได้ทั้งเขียนและอ่าน
- การใช้งาน stream ถ้าไม่ระบุ path จะได้ไฟล์อยู่ที่เดียวกับโปรแกรมที่เราเขียน

Collections

Collection classes

- Collection classes เป็นคลาสที่ถูกออกแบบมาสำหรับการเก็บและเรียกใช้ข้อมูลในลักษณะต่างๆ
  - คลาสเหล่านี้สามารถใช้ได้กับ stacks, queues, lists, และ hash tables
- Collection classes แทบทั้งหมด จะ implement มาจาก interfaces เดียวกัน
- Collection classes ช่วยให้เราจัดการ memory แบบ dynamic โดยไม่ต้องจองพื้นที่เก็บข้อมูลล่วงหน้า
- การเข้าถึงข้อมูลใน Collection classes ทำได้ในลักษณะเดียวกับการเข้าถึงข้อมูลในอาร์เรย์ (โดยใช้ index)
- Collection classes สามารถใช้ได้กับทุกชนิดข้อมูลในภาษา C#

ArrayList

- เก็บ object ต่างๆ ในลักษณะอาร์เรย์ที่สามารถเข้าถึงโดยการใช้ index แบบอิสระ
- นิยมใช้แทนอาร์เรย์ แต่จะดีกว่าตรงที่ มันสามารถยืดหดขนาดของตัวเอง ได้ตามปริมาณ object ที่เราใส่ลงไป
- สามารถเพิ่มหรือเรียกใช้สมาชิกของอาร์เรย์ได้ในตำแหน่งที่ต้องการ โดยการระบุ index
- สามารถเพิ่มเติม ค้นหา หรือจัดเรียง object ภายใน list ได้

Hashtable

- ใช้ key เพื่อเข้าถึงสมาชิก (elements) ใน collection
- hash table จะถูกนำมาใช้ เมื่อเราต้องการเข้าถึงสมาชิกใน collection โดยการใช้ key ที่กำหนด
- แต่ละ item ใน hash table จะมี key/value pair
- ดังนั้นเมื่อเรากำหนดความสัมพันธ์เรียบร้อยแล้ว เราสามารถใช้งาน hash table ได้คล้ายๆ กับ dictionary

SortedList

- ในการเข้าถึงข้อมูล เราสามารใช้ key หรือ index ก็ได้
- sorted list เป็นลูกผสมของ array และ hash table
- มันประกอบด้วย list ของ items ที่สามารถอ้างถึงโดยการใช้ key หรือ index
- ถ้าเข้าถึงข้อมูลโดยใช้ index มันจะมีสถานะเป็น ArrayList
- ถ้าเข้าถึงข้อมูลโดยใช้ key มันจะมีสถานะเป็น Hashtable.
- collection ของ items จะถูกจัดเรียงโดย key value เสมอ

Stack

- เป็น collection ที่จัดเก็บ object แบบ last-in, first out
- Stack จะถูกนำมาใช้ในงานที่ต้องการจัดการการเคลื่อนไหวของข้อมูลเข้า-ออกแบบ last-in, first-out
- การนำข้อมูลเข้าไปยัง collection เรียกว่า pushing
- การนำข้อมูลออกจาก collection เรียกว่า popping

Queue

- เป็น collection ที่มีลักษณะการจัดการข้อมูลแบบ first-in, first out
- จะถูกนำมาใช้ในงานที่ต้องการจัดการการเคลื่อนไหวของข้อมูลเข้า-ออกแบบ first-in, first-out
- การเพิ่มข้อมูลไปยัง collection เรียกว่า enqueue
- การดึงข้อมูลออกจาก collection เรียกว่า deque

BitArray

- ใช้แทน array ของเลขฐานสอง ซึ่งมีค่าได้ 2 ค่าคือ 1 และ 0
- ใช้เมื่อต้องการเก็บข้อมูลชนิดบิต ที่ไม่ทราบจำนวนล่วงหน้า
- เราสามารถเข้าถึงบิตที่ตำแหน่งใดๆ โดยการใช้ index ไปยังตำแหน่งนั้น
- ตำแหน่งเริ่มต้นของ index คือ 0

LINQ

Language Integrated Query

LINQ

- LINQ ย่อมาจาก Language Integrated Query
- ใช้งานคล้ายกับการ query ข้อมูลจาก database แต่ใช้งานได้กับ
  - objects ในหน่วยความจำ
  - ฐานข้อมูล
  - เอกสาร XML
  - และอื่นๆ
- เป็นส่วนขยายของ C# และ VB.NET

ทั่วไปเกี่ยวกับ LINQ

- การเขียนโปรแกรมเพื่อจัดการกับข้อมูล ที่ใช้เทคโนโลยีที่แตกต่างกันมากๆ เช่น SQL, Web Services, XQuery เป็นต้น ทำให้ programmer พัฒนาได้ช้า หรือไม่ก็ต้องใช้ทีมงานที่มีความชำนาญในหลายๆ ภาษา
- Visual studio 2008 ได้แนะนำส่วนขยายของ C# และ VB.NET เรียกว่า LINQ
- ทำให้ programmer ไม่จำเป็นต้องรู้ภาษา เช่น SQL, Web Services, XQuery อย่างลึกซึ้ง

ตัวอย่าง LINQ ในภาษา C#

```
1   using System;
2   using System.Linq;
3
    0 references | 0 changes | 0 authors, 0 changes
4   class Program
5   {
        0 references | 0 changes | 0 authors, 0 changes
6       static void Main()
7       {
8           string[] words = { "hello", "wonderful", "LINQ", "beautiful", "world" };
9           //Get only short words
10          var shortWords = from word in words
11                           where word.Length <= 5
12                           select word;
13
14          //Print each word out
15          foreach (var word in shortWords)
16          {
17              Console.WriteLine(word);
18          }
19          Console.ReadLine();
20      }
21  }
```

```
hello
LINQ
world
```

รูปแบบของ LINQ

- Lamda (Method) Syntax

```
var longWords = words.Where(w => w.length > 10);
```

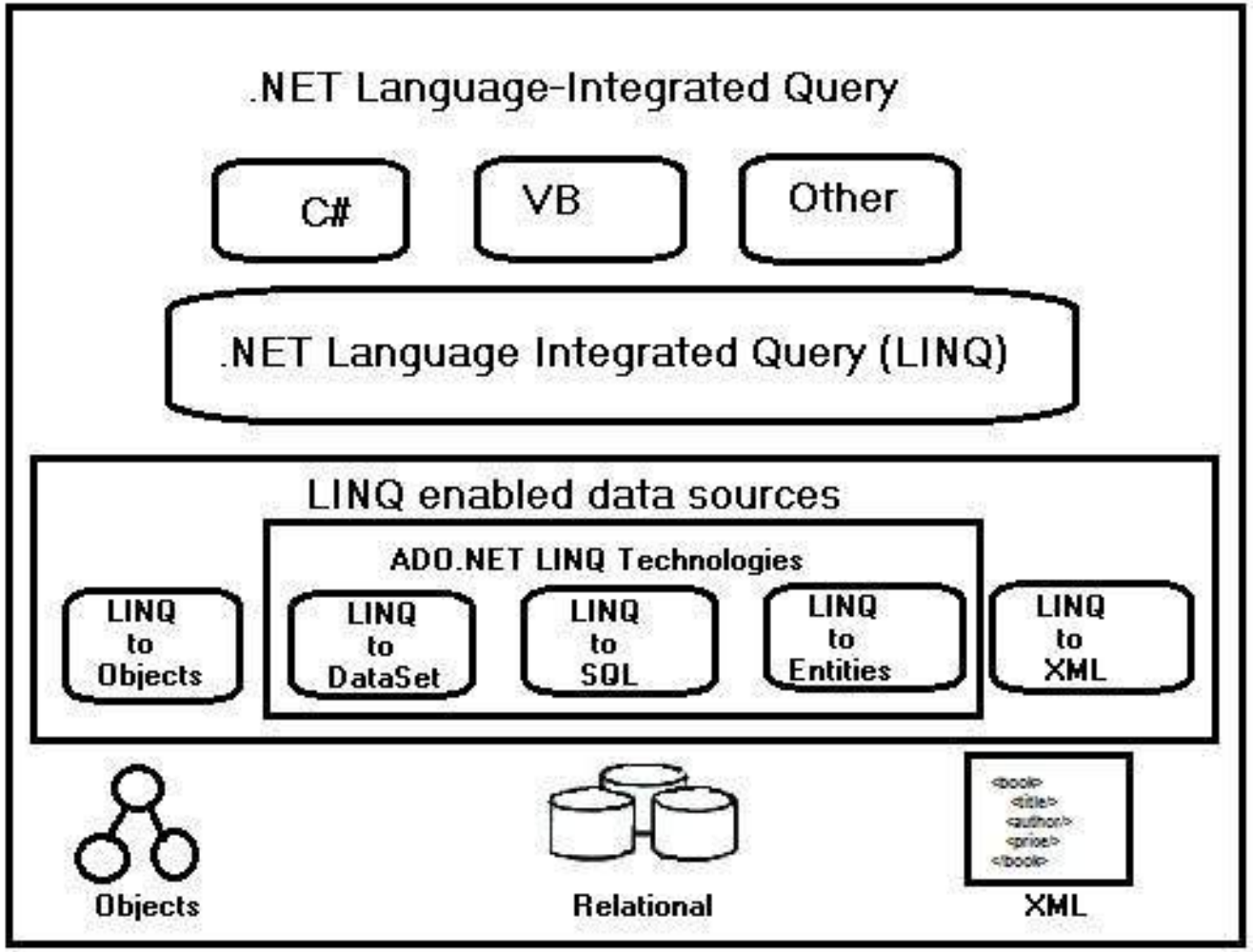
- Query (Comprehension) Syntax

```
var longwords = from w in words where w.length > 10;
```

ชนิดของ LINQ

- LINQ to Objects
- LINQ to XML(XLINQ)
- LINQ to DataSet
- LINQ to SQL (DLINQ)
- LINQ to Entities

สถาปัตยกรรมของ LINQ ใน .NET



ตัวอย่างการประยุกต์ใช้ LINQ

SQL CONNECTION

```
SqlConnection sqlConnection = new SqlConnection(connectString);
SqlConnection.Open();
System.Data.SqlClient.SqlCommand sqlCommand = new SqlCommand();
sqlCommand.Connection = sqlConnection;
sqlCommand.CommandText = "Select * from Customer";
return sqlCommand.ExecuteReader (CommandBehavior.CloseConnection)
```

LINQ

```
Northwind db = new Northwind(@"C:\Data\Northwnd.mdf");

var query = from c in db.Customers select c;
```

ข้อดีของ LINQ

- syntax highligh ทำให้เขียน code ง่าย
- IntelliSense ช่วยเขียน code เร็วขึ้น
- easy debugging เนื่องจากเป็นส่วนขยายของ C#
- ใช้รูปแบบการเขียนเพียงแบบเดียว กับ datasource ทุกชนิด
- สามารถเชื่อมหลายๆ datasource เป็น single query
- สามารถ transform จาก data type ไปยังต่างชนิด เช่น จาก SQL เป็น XML data

LINQ - Query Operators

- **Filtering Operators**
  - Where
  - OfType
- **Join Operators**
  - Join
  - GroupJoin
- **Projection Operations**
  - Select
  - SelectMany
- **Sorting Operators**
  - OrderBy
  - OrderByDescending
  - ThenBy
  - ThenByDescending
  - Reverse
- **Grouping Operators**
  - GroupBy
  - ToLookup

LINQ - Query Operators

- **Conversions**
  - AsEnumerable
  - AsQueryable
  - Cast
  - OfType
  - ToArray
  - ToDictionary
  - ToList
  - ToLookup
- **Concatenation**
  - Concat
- **Aggregation**
  - Aggregate
  - Average
  - Count
  - LonCount
  - Max
  - Min
  - Sum
- **Quantifier Operations**
  - All
  - Any
  - Contains

LINQ - Query Operators

- **Partition Operators**
  - Skip
  - SkipWhile
  - Take
  - TakeWhile
- **Generation Operations**
  - DefaultIfEmpty
  - Empty
  - Range
  - Repeat
- **Set Operations**
  - Distinct
  - Except
  - Intersect
  - Union
- **Equality**
  - SequenceEqual

LINQ - Query Operators

- **Element Operators**
  - ElementAt
  - ElementAtOrDefault
  - First
  - FirstOrDefault
  - Last
  - LastOrDefault
  - Single
  - SingleOrDefault
  - DefaultIfEmpty