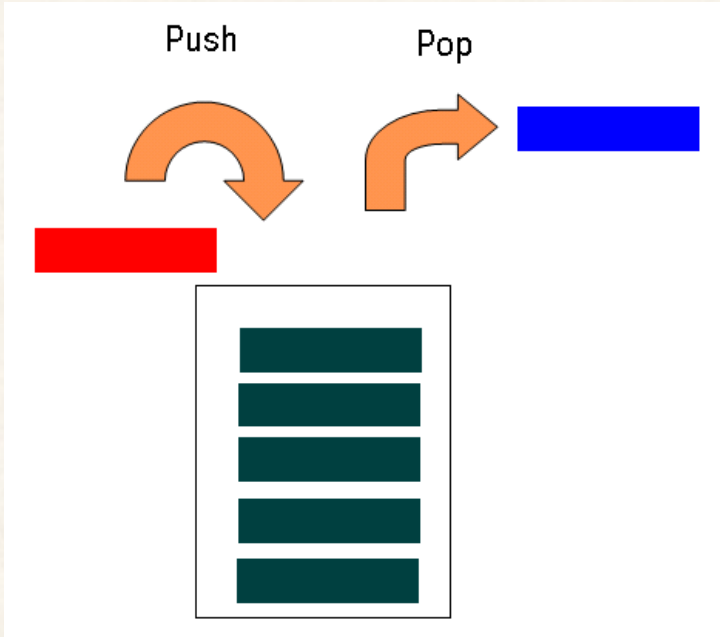
การเขียนโปรแกรม ด้วยภาษา C#

Generics

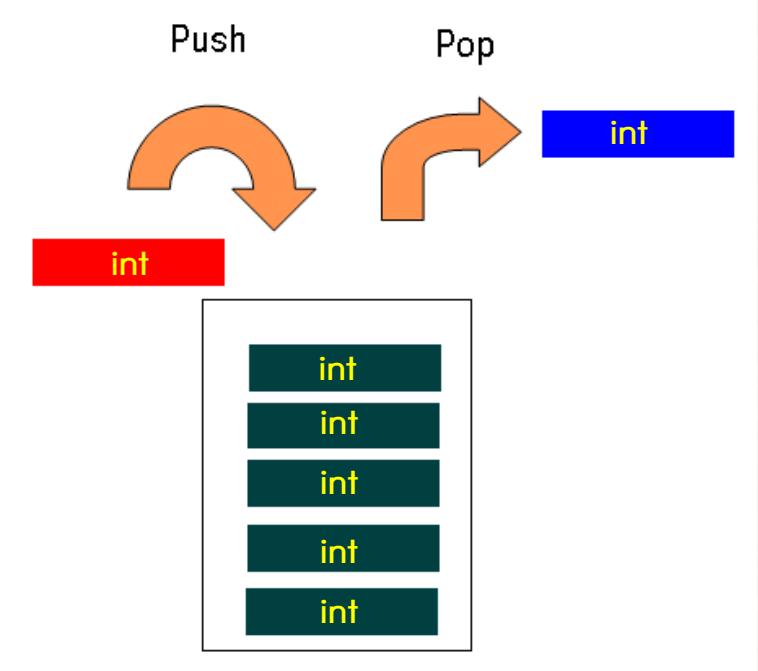
Generics

What Are Generics?

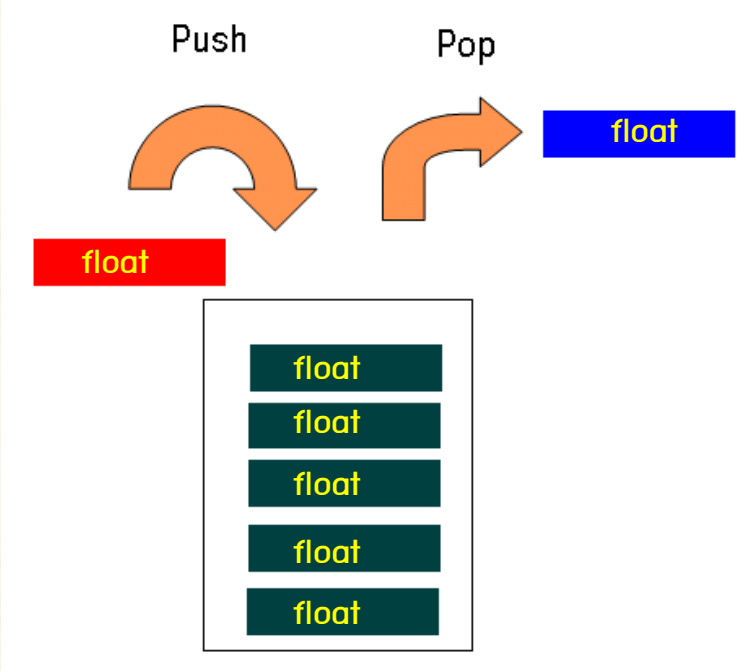
- Generic ช่วยให้ลดภาระในการเขียนโปรแกรม ในลักษณะที่มีอัลกอริทึมเดียวกัน แต่ใช้ data type ต่างกัน
  - ตัวอย่าง Stack



ชนิดข้อมูลต่างกัน แต่ Algorithm เดียวกัน???



stack ชนิด int



stack ชนิด float

class MyIntStack

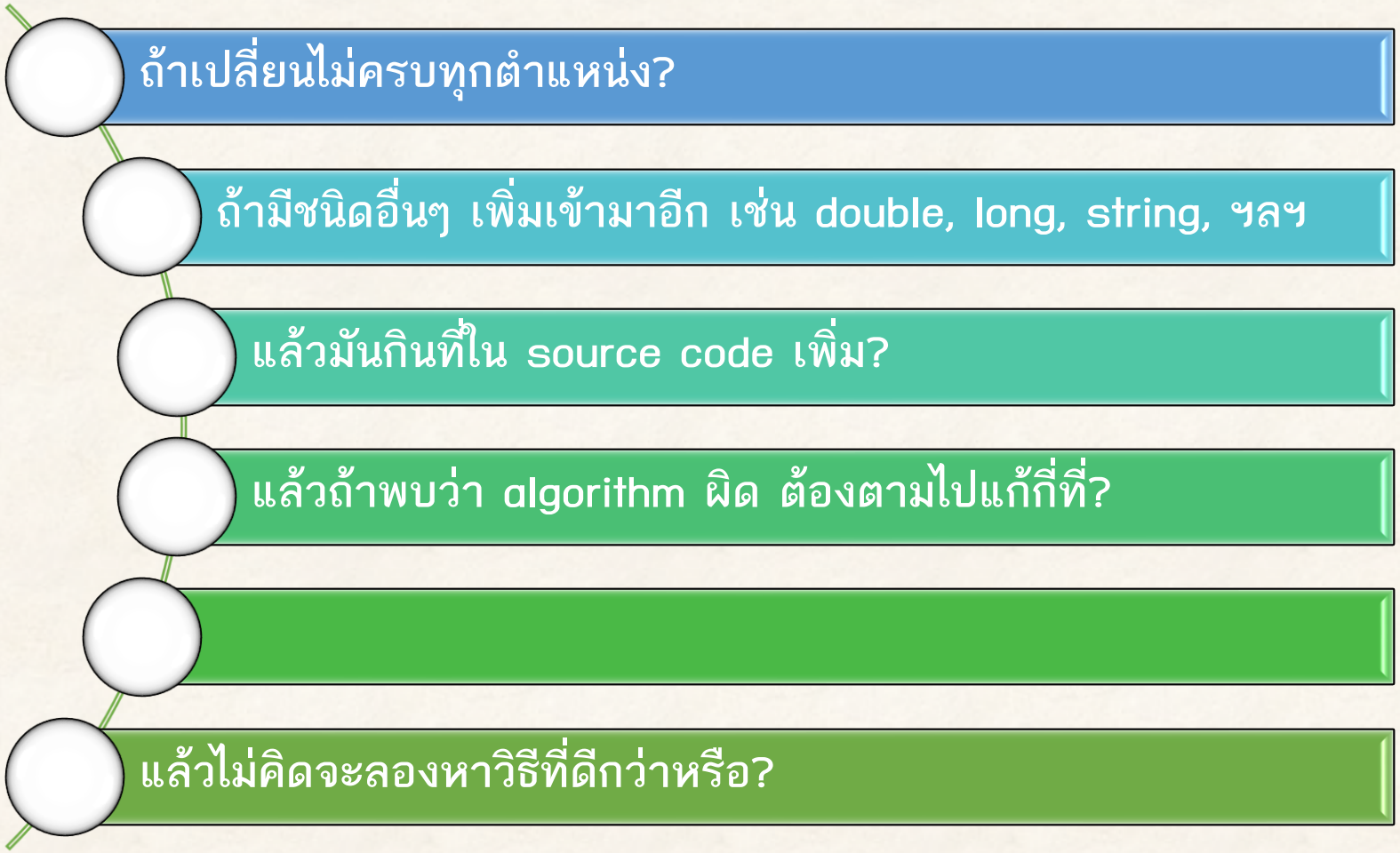
```
class MyIntStack                          // Stack for ints
{
   int   StackPointer = 0;
   int[] StackArray;                      // Array of int
    ↑                  int
   int                 ↓
   public void Push( int x )              // Input type: int
   {
      ...
   }       int
           ↓
   public int Pop()                       // Return type: int
   {
      ...
   }

     ...
}
```

class MyFloatStack

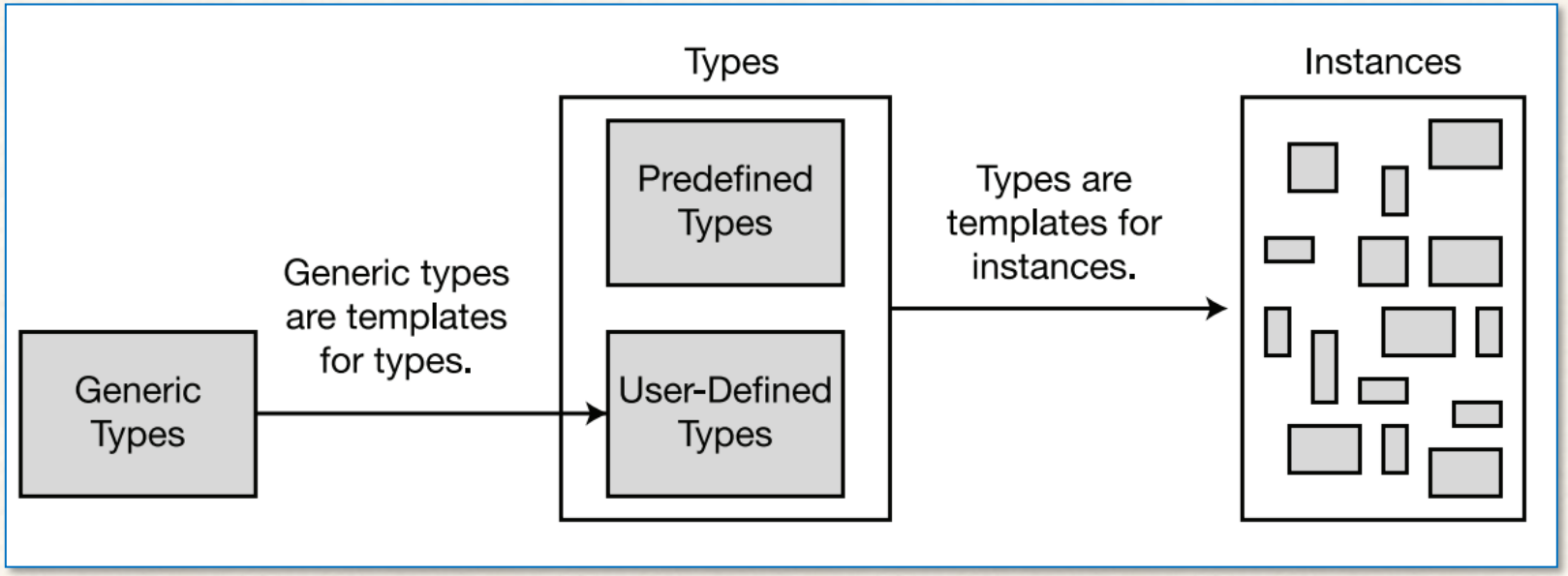
```
class MyFloatStack                          // Stack for floats
{
   int   StackPointer = 0;
   float [] StackArray;                     // Array of float
     ↑                    float
   float                    ↓
   public void Push( float x )              // Input type: float
   {
      ...
   }
            float
              ↓
   public float Pop()                       // Return type: float
   {
      ...
   }

   ...
}
```

Copy-paste แล้วเปลี่ยน int เป็น float

copy-paste, replace... ก็จบ แล้วมีปัญหาตรงไหน?



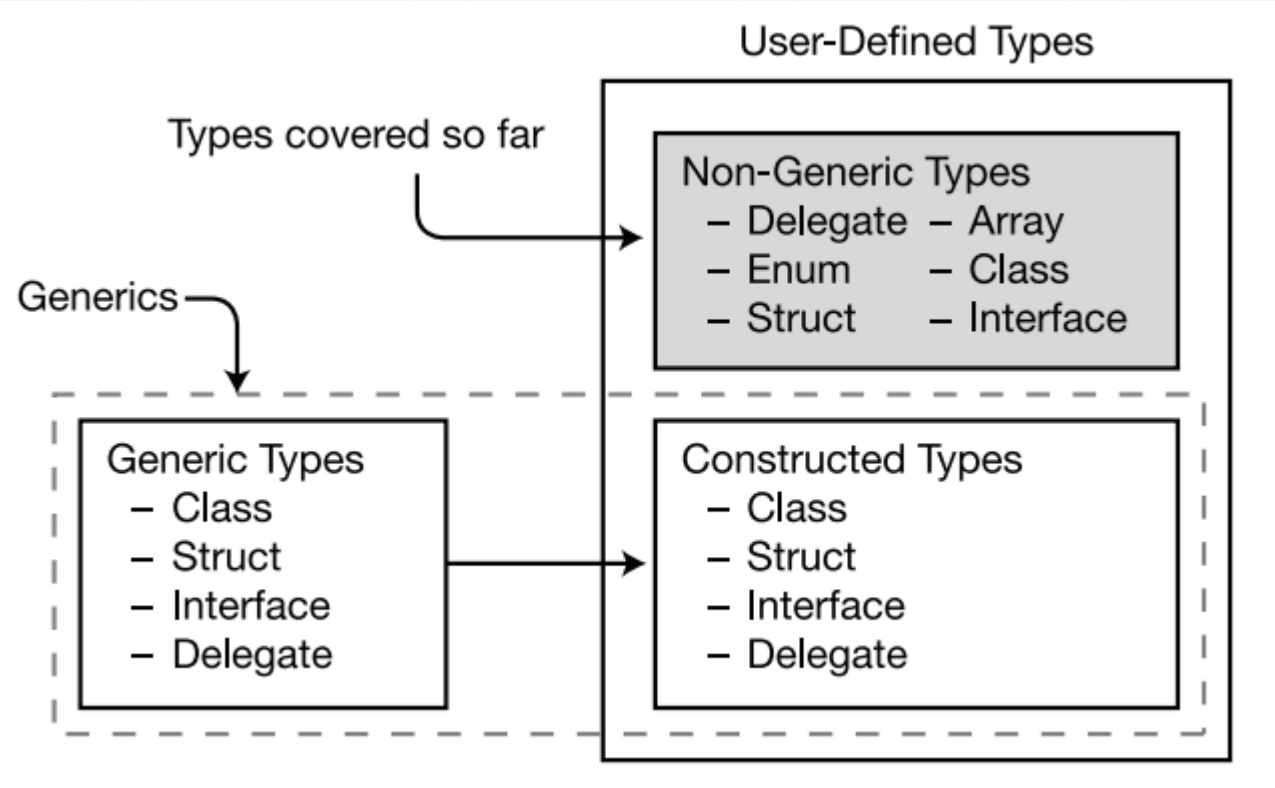
Generics ใน C#



อะไรเป็น Generic ได้บ้าง

- classes
- structs
- interfaces
- delegates
- methods

Generics ใน C#



แก้คลาส stack ให้เป็น Generic

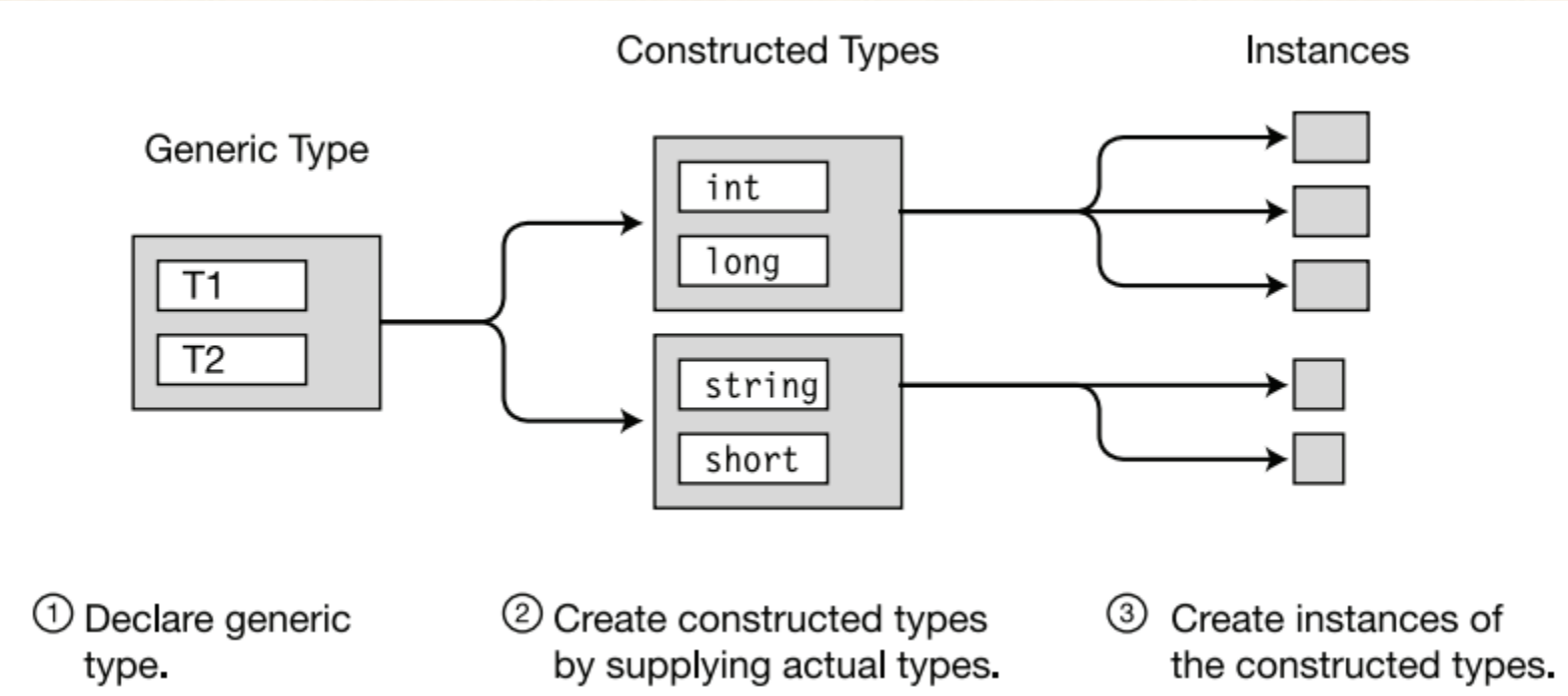
- ในคลาส MyIntStack ให้เปลี่ยน int ทุกตำแหน่งเป็นตัวอักษร T
- เปลี่ยนชื่อ MyIntStack เป็น MyStack
- ใส่ <T> หลังชื่อคลาส

```
class MyStack <T>
{
    int StackPointer = 0;
    T [] StackArray;

    public void Push(T x ) {...}

    public T Pop() {...}
        ...
}
```

Generic Classes

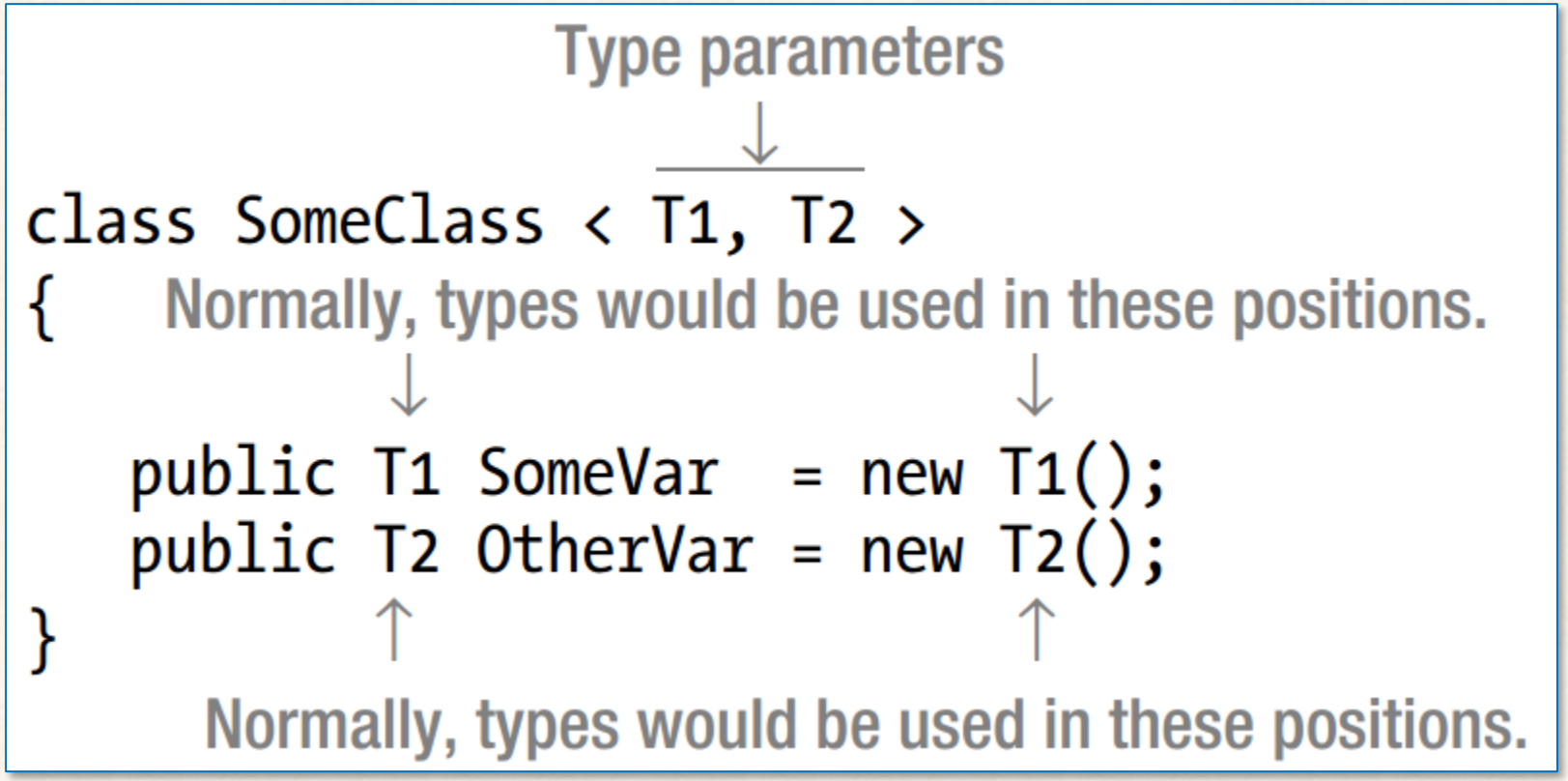
- ในการสร้าง object ปกติจะมี 2 ขั้นตอนคือ
  1. สร้าง class
  2. สร้าง instance ของ class (ก็คือ object)
- แต่ Generic Class ใช้สร้าง object โดยตรงไม่ได้
  - เพราะมันเป็นเพียง template ไม่ใช่ แบบแปลน
- เราต้องระบุชนิดข้อมูลให้กับ Generic Class ก่อน จึงมี 3 ขั้นตอน
  1. สร้าง Generic class (ได้ template ของ class)
  2. ระบุชนิดที่ต้องการให้กับ place holder ของคลาส
  3. สร้าง instance ของ class ในข้อ 2.

Generic Classes



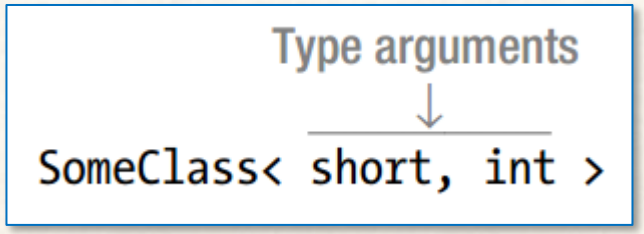
การสร้าง Generic Class

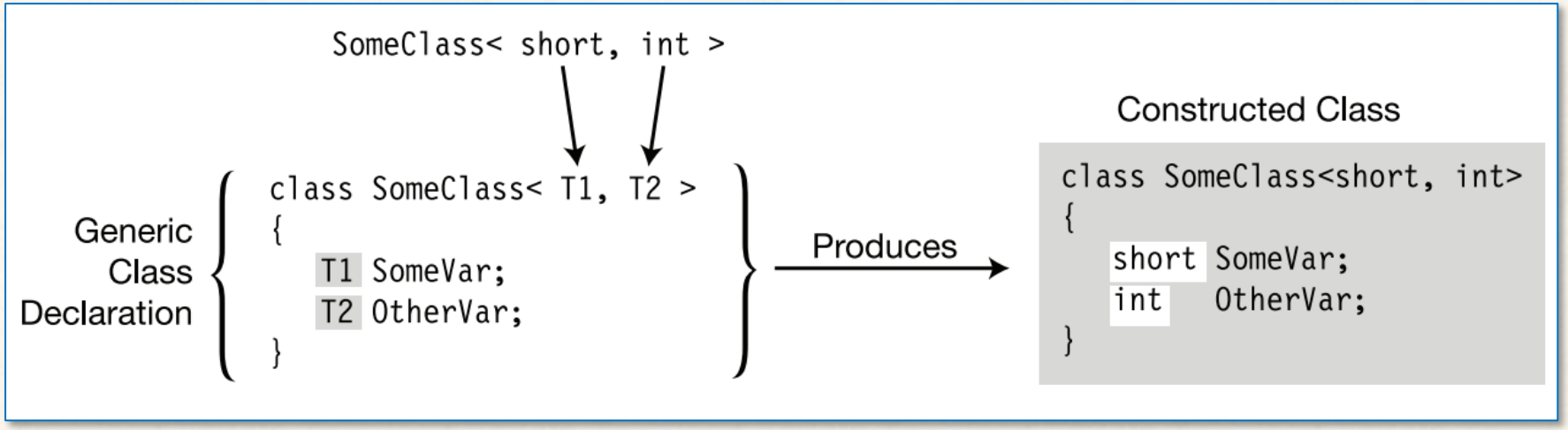
1 **Declaring a Generic Class**



การสร้าง Generic Class

2 **Creating a Constructed Type**





การสร้าง Generic Class

3 Creating Variables and Instances

การสร้าง object จาก class ปกติ

```
MyNonGenClass          myNGC = new MyNonGenClass          ();
     Constructed class                    Constructed class
     ↓                                    ↓
SomeClass<short, int>  mySc1 = new SomeClass<short  int>();
var                    mySc2 = new SomeClass<short, int>();
```

การสร้าง object จาก Generic class

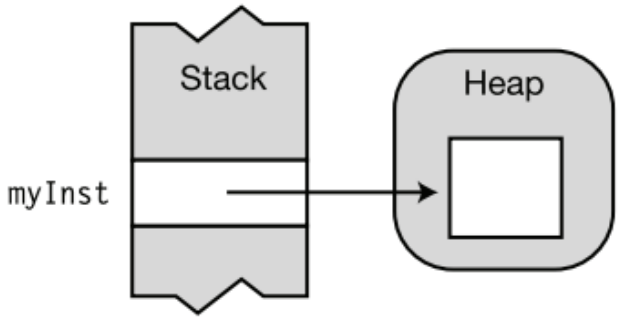
การสร้าง reference และ object

```
class SomeClass< T1, T2 >
{
   public T1 SomeVar;
   public T2 OtherVar;
}
   ...
SomeClass< short, int >  myInst;

myInst = new SomeClass< short, int >( );
```

Generic Class Declaration

Allocate class variable of the constructed type

Allocate instance



ตัวอย่าง

```
class SomeClass< T1, T2 >                                   // Generic class
{
   ...
}

class Program
{
   static void Main()
   {
      var first  =  new SomeClass<short, int >();    // Constructed type
      var second =  new SomeClass<int,   long>();    // Constructed type

         ...
```

```
class SomeClass< T1, T2 >
{
   T1 SomeVar;
   T2 OtherVar;
}
   ...
var first = new SomeClass<short, int> ( );

var second = new SomeClass<int, long> ( );
   ...
```

```
class SomeClass <short,int>
{
   short SomeVar;
   int   OtherVar;
}
```

```
class SomeClass <int,long>
{
   int   SomeVar;
   long  OtherVar;
}
```

ตัวอย่าง Stack ใช้ Generics (1)

```
class MyStack<T>
{
   T[] StackArray;
   int StackPointer = 0;

   public void Push(T x)
   {
      if ( !IsStackFull )
         StackArray[StackPointer++] = x;
   }

   public T Pop()
   {
      return ( !IsStackEmpty )
         ? StackArray[--StackPointer]
         : StackArray[0];
   }
```

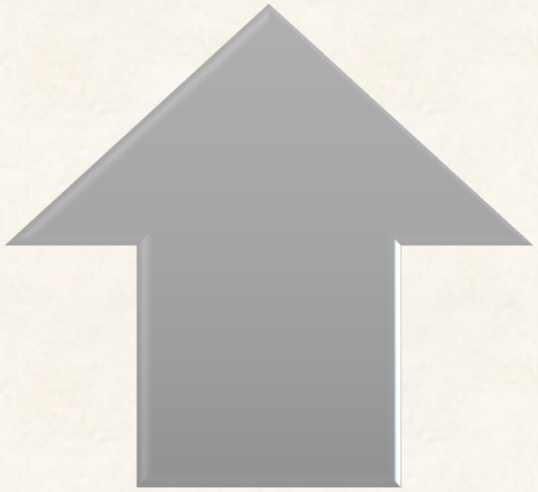
ตัวอย่าง Stack ใช้ Generics (2)

```
    const int MaxStack = 10;
    bool IsStackFull  { get{ return StackPointer >= MaxStack; } }
    bool IsStackEmpty { get{ return StackPointer <= 0; } }

    public MyStack()
    {
       StackArray = new T[MaxStack];
    }

    public void Print()
    {
       for (int i = StackPointer-1; i >= 0 ; i--)
          Console.WriteLine("   Value: {0}", StackArray[i]);
    }
}
```

ตัวอย่าง Stack ใช้ Generics (3)

```
class Program
{
   static void Main( )
   {
      MyStack<int>    StackInt    = new MyStack<int>();
      MyStack<string> StackString = new MyStack<string>();

      StackInt.Push(3);
      StackInt.Push(5);
      StackInt.Push(7);
      StackInt.Push(9);
      StackInt.Print();

      StackString.Push("This is fun");
      StackString.Push("Hi there!  ");
      StackString.Print();
   }
}
```

Generic vs Nongeneric Stack



Generic

- source code เล็กกว่า
- Executable Size เล็กกว่า
- เขียนง่ายกว่า
- บำรุงรักษาง่าย

nongeneric

- source code ใหญ่
- Executable Size ใหญ่กว่า
- เขียนยากกว่า
- บำรุงรักษายาก

ส่วนของ class

```
class MyIntStack
{
   int[] StackArray;
   int   StackPointer = 0;

   public void Push( int x )
   { ... }
   public int Pop()
   { ... }
}

class MyStringStack
{
   string[] StackArray;
   int   StackPointer = 0;

   public void Push( string x )
   { ... }
   public string Pop()
   { ... }
}
```

Nongeneric

Generic

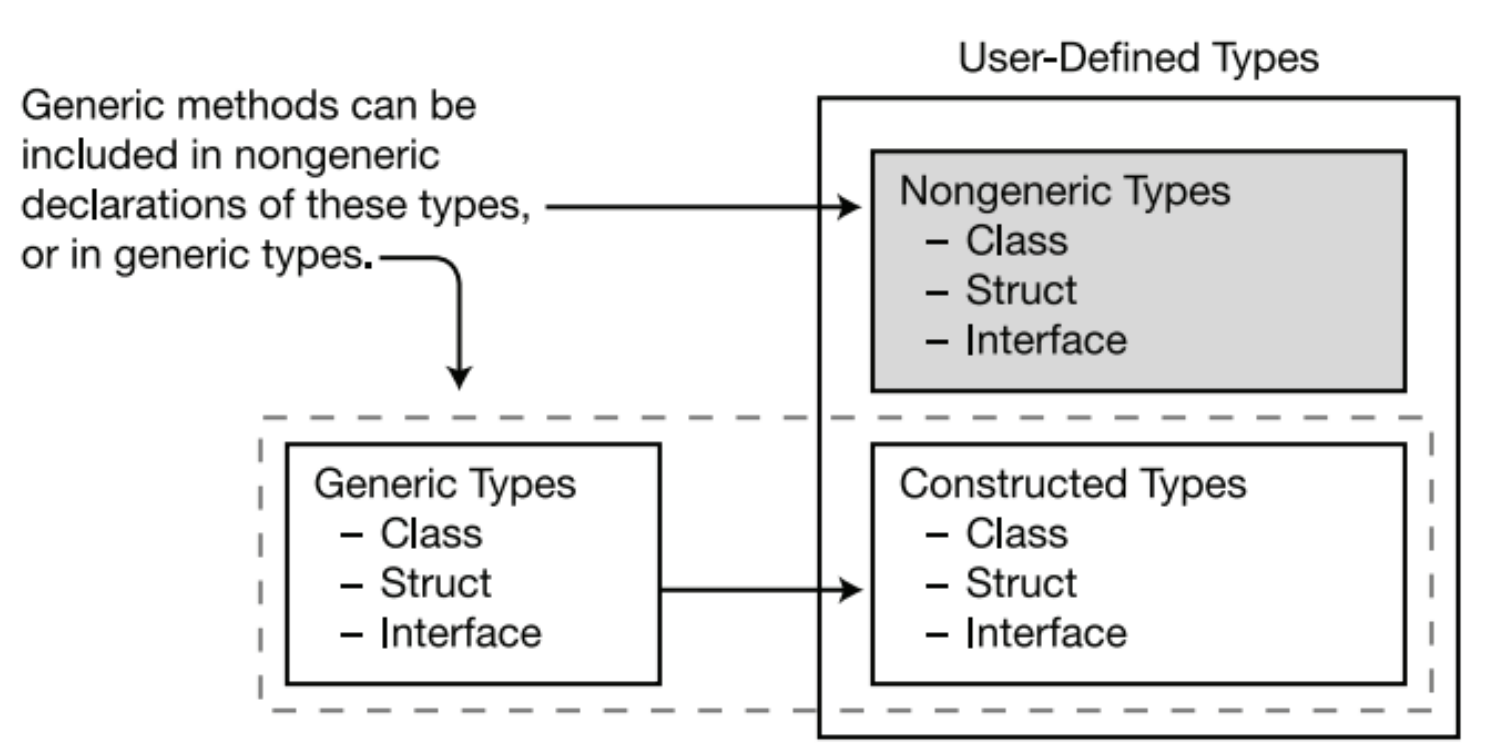
```
class MyStack < T >
{
   T[] StackArray;
   int        StackPointer = 0;

   public void Push(T x )
   {...}
   public T Pop()
   {...}
}
```

ส่วนของการเรียกใช้

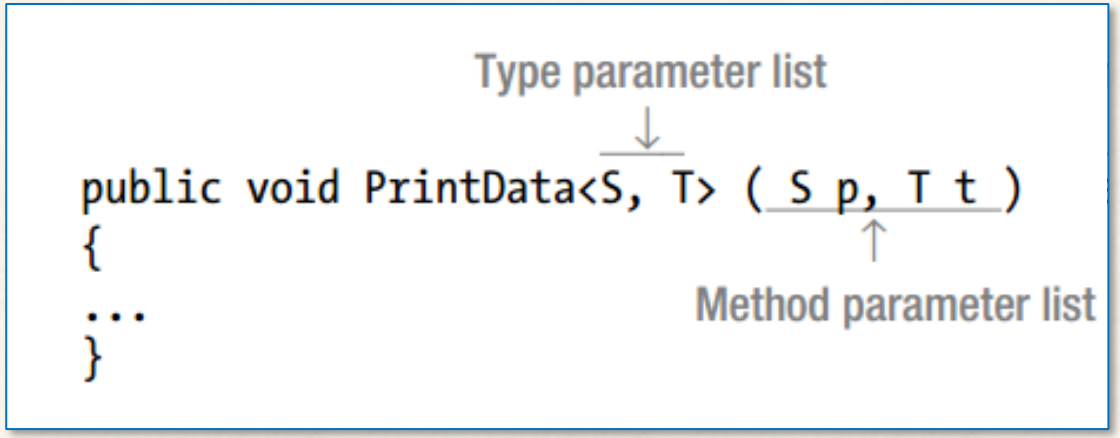
```
static void Main()
{
    var intStack =
                new MyintStack();
    var stringStack =
                new MyStringStack();
        ...
}
```

```
static void Main()
{
    var intStack =
                new MyStack<int>();
    var stringStack =
                new MyStack<string>();
        ...
}
```

Generic Methods



You can declare generic methods in both generic and nongeneric classes


Type parameter list
public void PrintData<S, T> (S p, T t)
{
...
}
Method parameter list

Example of a Generic Method

```
class Simple                                          // Non-generic class
{
   static public void ReverseAndPrint<T>(T[] arr)     // Generic method
   {
      Array.Reverse(arr);
      foreach (T item in arr)                         // Use type argument T.
         Console.Write("{0}, ", item.ToString());
      Console.WriteLine("");
   }
}
```

Example of a Generic Method

```
class Program
{
   static void Main()
   {
      // Create arrays of various types.
      var intArray    = new int[]    { 3, 5, 7, 9, 11 };
      var stringArray = new string[] { "first", "second", "third" };
      var doubleArray = new double[] { 3.567, 7.891, 2.345 };

      Simple.ReverseAndPrint<int>(intArray);         // Invoke method.
      Simple.ReverseAndPrint(intArray);              // Infer type and invoke.

      Simple.ReverseAndPrint<string>(stringArray);   // Invoke method.
      Simple.ReverseAndPrint(stringArray);           // Infer type and invoke.

      Simple.ReverseAndPrint<double>(doubleArray);   // Invoke method.
      Simple.ReverseAndPrint(doubleArray);           // Infer type and invoke.
   }
}
```

Generic Structs

```
struct PieceOfData<T>                                    // Generic struct
{
    public PieceOfData(T value) { _data = value; }
    private T _data;
    public  T Data
    {
        get { return _data; }
        set { _data = value; }
    }
}
```

Generic Structs

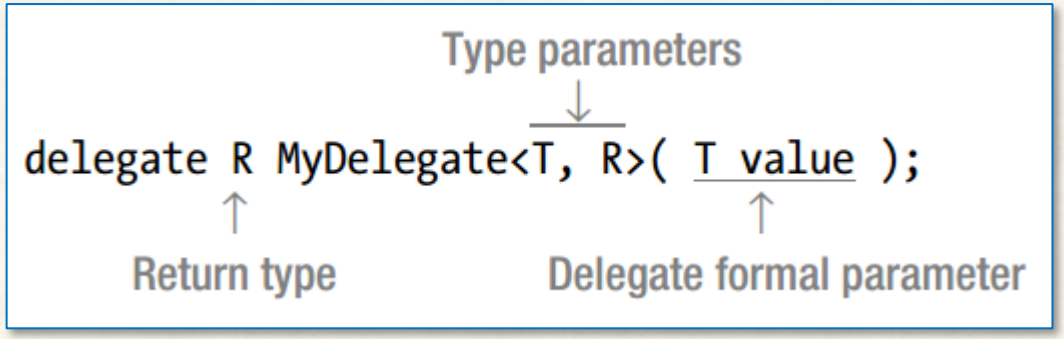
```
class Program
{
   static void Main()
   {
      var intData    = new PieceOfData<int>(10);
      var stringData = new PieceOfData<string>("Hi there.");

      Console.WriteLine("intData    = {0}", intData.Data);
      Console.WriteLine("stringData = {0}", stringData.Data);
   }
}
```

Constructed type

Constructed type

Generic Delegates

- declare a generic delegate



Generic Delegates

```
delegate void MyDelegate<T>(T value);                  // Generic delegate

class Simple
{
   static public void PrintString(string s)        // Method matches delegate
   {
      Console.WriteLine(s);
   }

   static public void PrintUpperString(string s)  // Method matches delegate
   {
      Console.WriteLine("{0}", s.ToUpper());
   }
}
```

Generic Delegates

```
class Program
{
   static void Main( )
   {
      var myDel =                                    // Create inst of delegate.
         new MyDelegate<string>(Simple.PrintString);
      myDel += Simple.PrintUpperString;              // Add a method.

      myDel("Hi There.");                            // Call delegate.
   }
}
```

Generic Interfaces

Type parameter

```
interface IMyIfc<T>                          // Generic interface
{
   T ReturnIt(T inValue);
}
```

Type parameter Generic interface

```
class Simple<S> : IMyIfc<S>                  // Generic class
{
   public S ReturnIt(S inValue)              // Implement generic interface.
   { return inValue; }
}
```

Generic Interfaces

```
class Program
{
   static void Main()
   {
      var trivInt    = new Simple<int>();
      var trivString = new Simple<string>();

      Console.WriteLine("{0}", trivInt.ReturnIt(5));
      Console.WriteLine("{0}", trivString.ReturnIt("Hi there."));
   }
}
```