В.В. Подбельский

Иллюстрации к курсу лекций по дисциплине «Программирование на C#»

C#

Использованы материалы пособия Daniel Solis, Illustrated C#

Enumerables, Enumerators and Iterators

Перечислимые, перечислители и итераторы

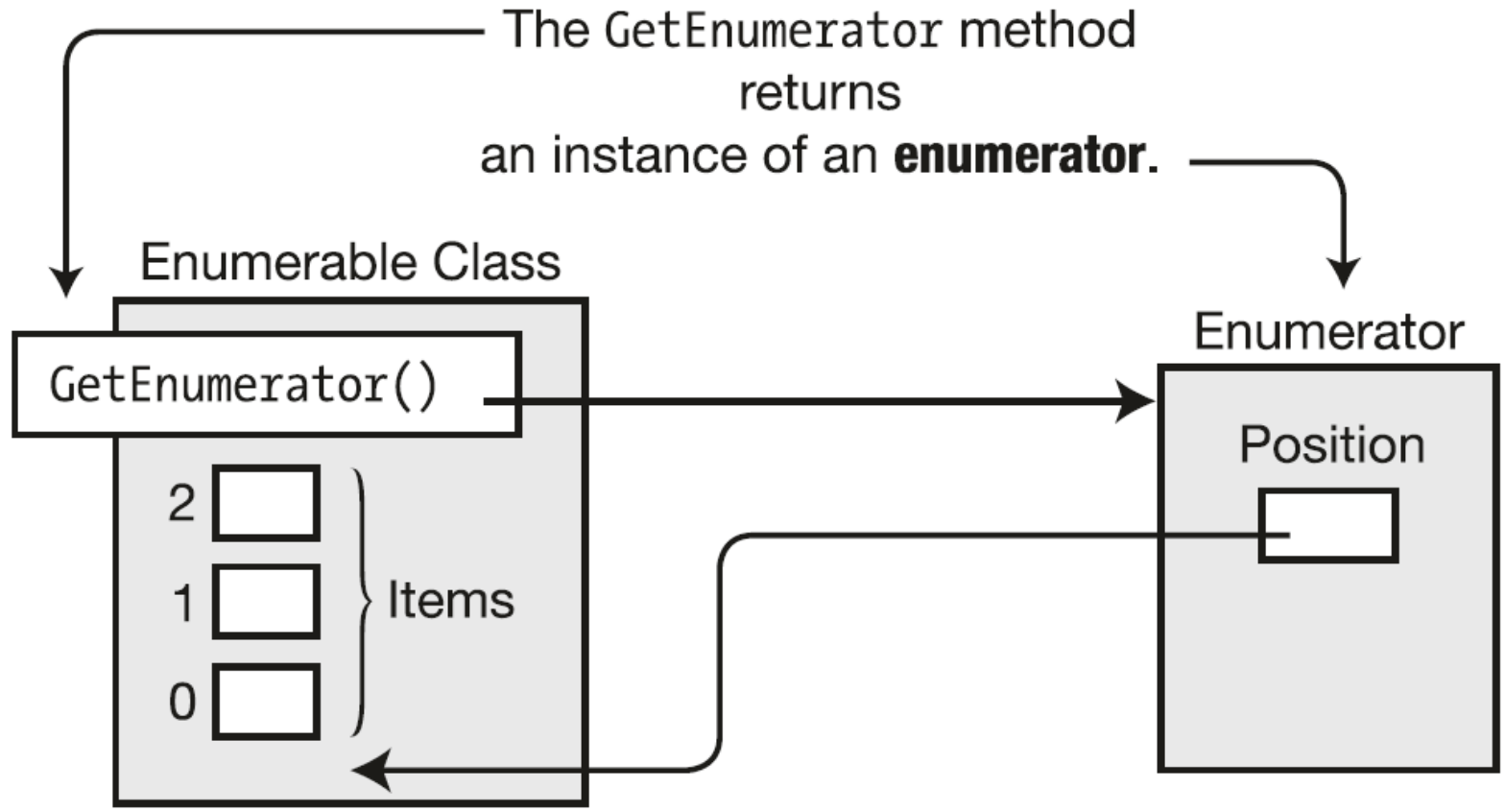
Использование оператора foreach

```
int[] arr = { 10, 11, 12, 13 };
foreach (int item in arr)          // перечисление элементов
    Console.WriteLine($"Item value: {item}");
```

Результат работы:

Item value: 10
Item value: 11
Item value: 12
Item value: 13

Перечислители и перечисляемые (*enumerators and enumerables*)



An **enumerable** is a type that has a method called `GetEnumerator` that returns an enumerator for its items.

An **enumerator** is an object that can return each item in a collection, in order.

Способы создания перечислителей

Способы создания перечислителей (нумераторов):

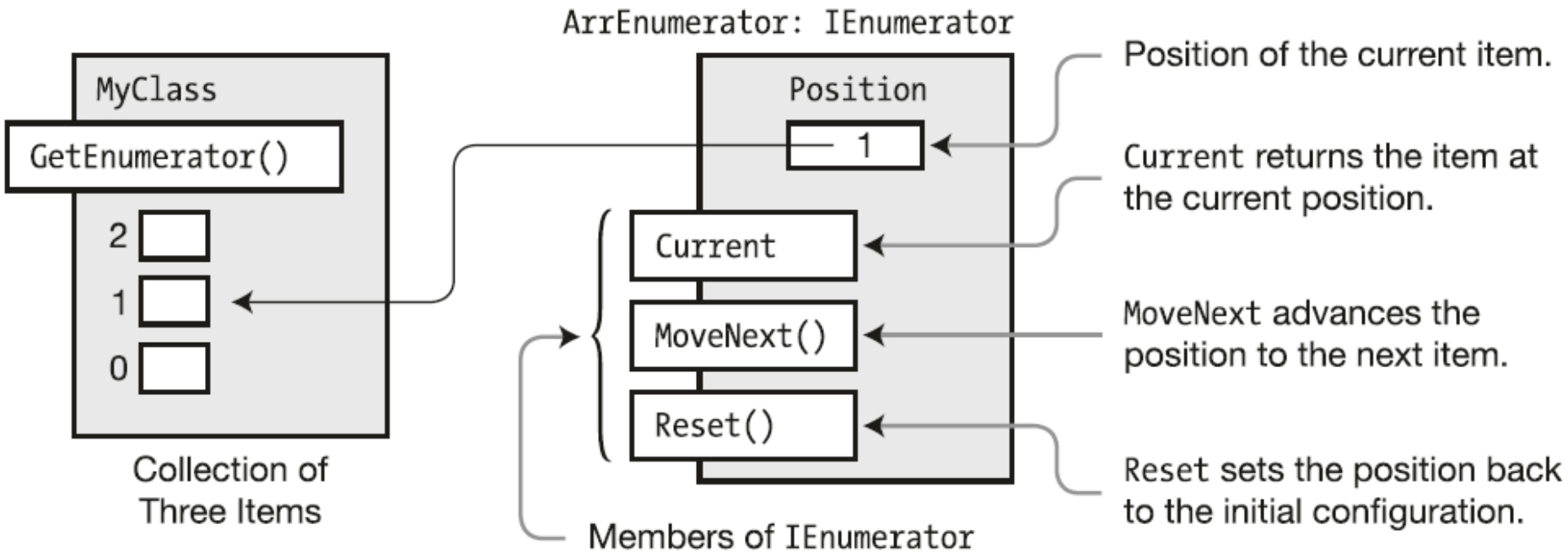
- Интерфейсы **IEnumerator** / **IEnumerable**;
- Интерфейсы **IEnumerator<T>** / **IEnumerable<T>**;
- Конструкция, в которой интерфейсы не применяются (утиная типизация).

Использование интерфейса IEnumerator

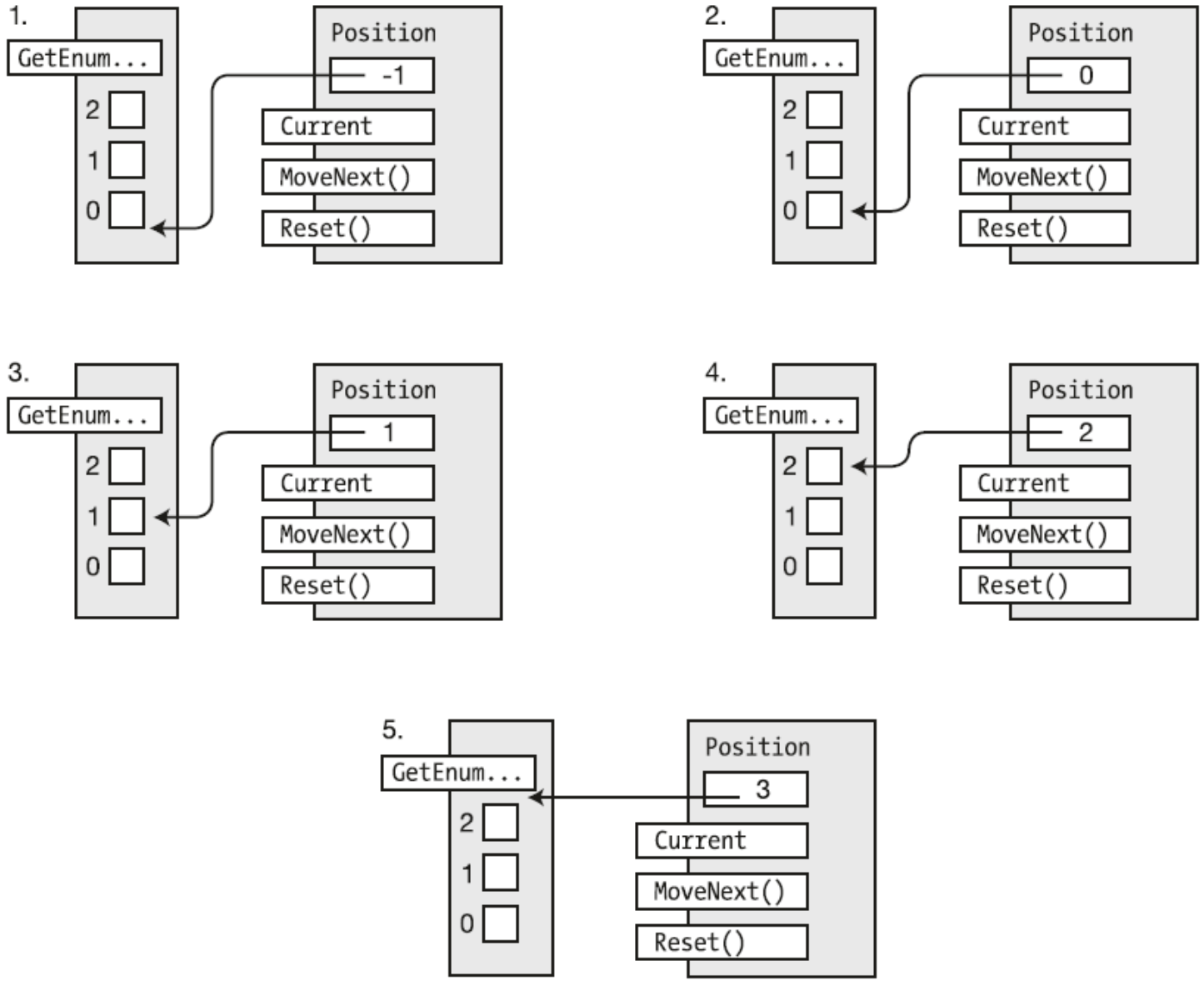
Члены интерфейса IEnumerator:

- object **Current** { get; };
- bool **MoveNext**();
- void **Reset**().

Перечислитель для коллекции



Состояния перечислителя



Моделирование оператора foreach

```
using System;
using System.Collections;

public static void Main()
{
    int[] arr1 = { 10, 11, 12, 13 };

    IEnumerator ie = arr1.GetEnumerator();  // получаем перечислитель System.Collections;
    // IEnumerator<int> ie = (arr1 as IEnumerable<int>).GetEnumerator(); // ...Generic;

    while (ie.MoveNext()) { // к следующему элементу
        int item = (int)ie.Current; // получаем текущий элемент
        Console.WriteLine($"Item value: { item }");
    }
}
```

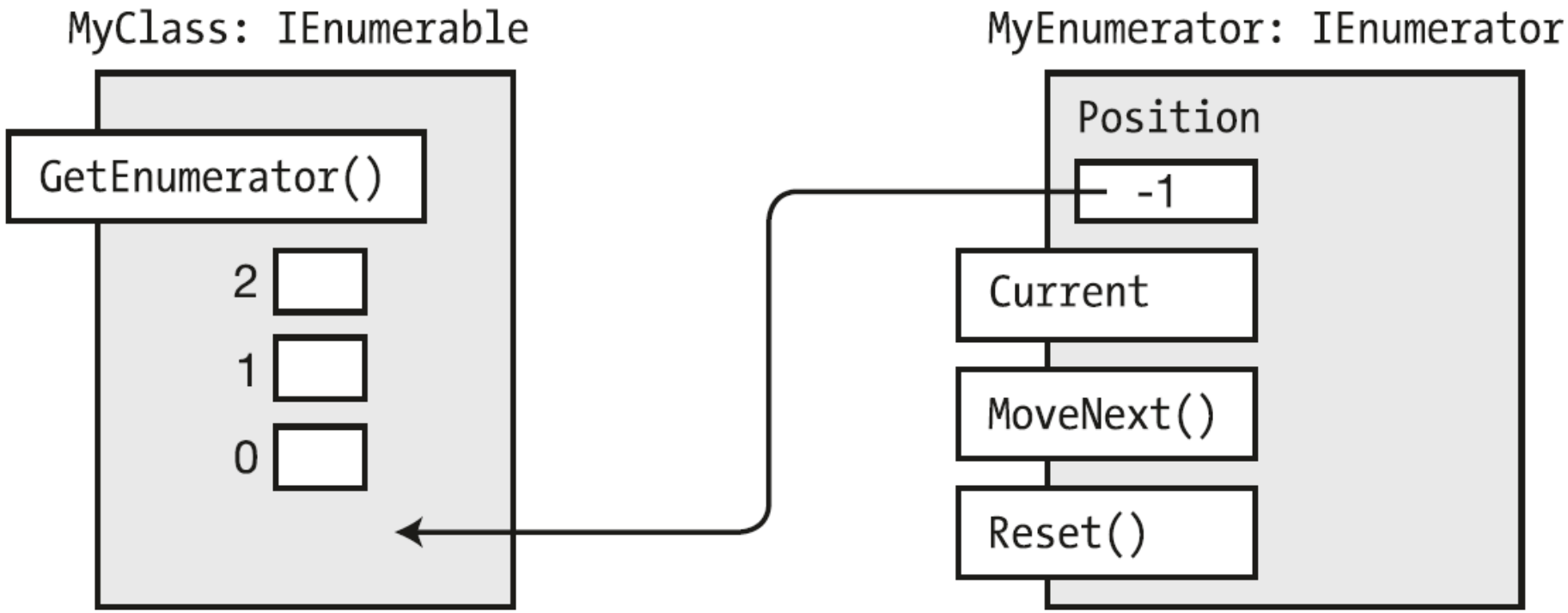
Результат работы:

10
11
12
13

Определение перечислителя IEnumerator

```
using System.Collections;          // подключаем пространство имен

class MyEnumerator: IEnumerator
{ возвращает ссылку на объект
                ↓
  public object Current { get; }   // текущий элемент
  public bool MoveNext() { ... }   // перемещение к следующему элементу
  public void Reset() { ... }      // сброс (возврат к началу)
  ...
}
```

Интерфейс IEnumerable



Способ определения перечислимого класса

```
using System.Collections;
Реализация интерфейса IEnumerable
                        ↓
class MyClass : IEnumerable
{
    public IEnumerator GetEnumerator { ... }
...                 ↑
}      возврат объекта типа IEnumerator
```

Пример «Enumerator»

```
using System.Collections;
class ColorEnumerator: IEnumerator     {
            string[ ] Colors;    int Position = -1;
public ColorEnumerator(string[ ] theColors)          // конструктор
{           Colors = new string[theColors.Length];
            for (int i = 0; i < theColors.Length; i++)
                        Colors[i] = theColors[i];
}

public object Current  { get { return Colors[Position]; } }   // Current
public bool MoveNext( )                          // MoveNext
{           if (Position < Colors.Length - 1) {  Position++;    return   true;  }
            else   return   false;
}
public void Reset()  { Position = -1; }        // Reset
}
```

Пример «Enumerable»

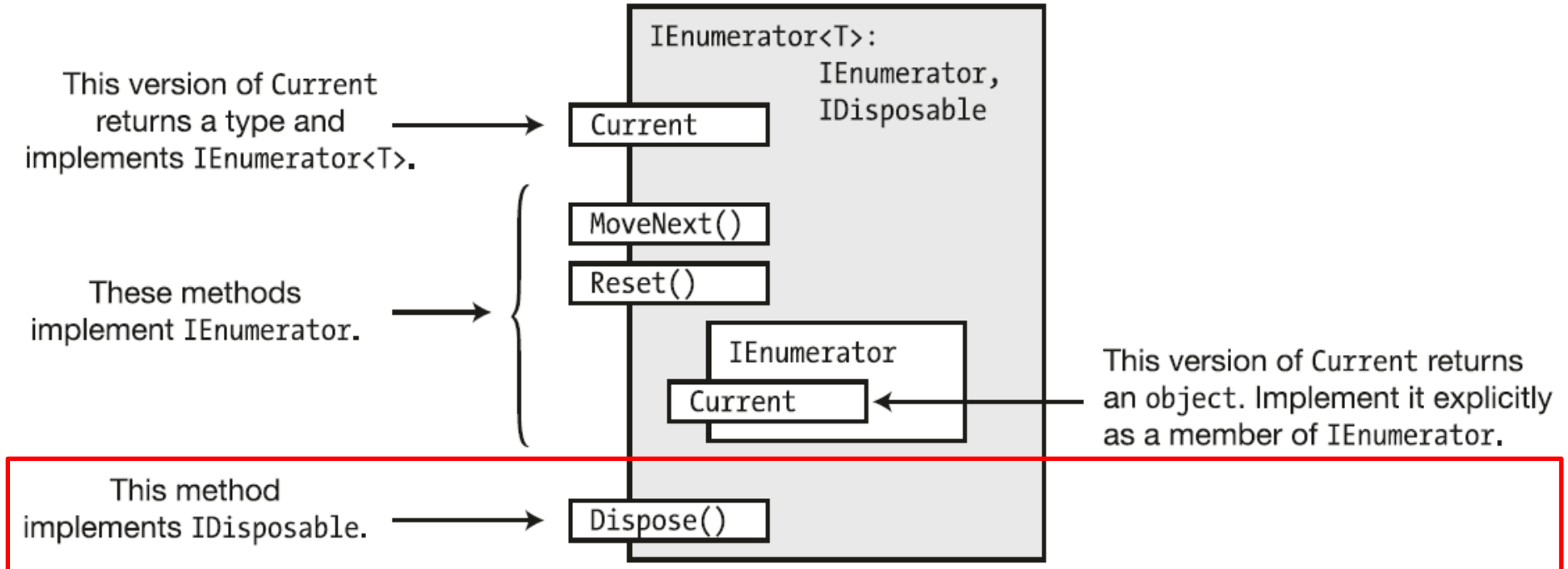
```
class MyColors: IEnumerable  {
        string[ ] Colors = { "Red", "Yellow", "Blue" };
public IEnumerator GetEnumerator()
        { return new ColorEnumerator(Colors); }

}  //  class MyColors

class Program  {
static void Main()  {
   MyColors  mc  = new MyColors();
   foreach (string  color  in  mc)
       Console.WriteLine("{0}", color);
   }
}
```

Перечислитель без интерфейса

```
class SibEnumerator : IEnumerator
{
   ...
   public object Current
      { get { ... } }

   public bool MoveNext()
   { ... }

   public void Reset()
   { ... }
}

class Siblings : IEnumerable
{
   ...
   public IEnumerator GetEnumerator()
   { ... }
}
```

```
class SibEnumerator
{
   ...
   public string Current
      { get { ... } }

   public bool MoveNext()
   { ... }



}

class Siblings
{
   ...
   public SibEnumerator GetEnumerator()
   { ... }
}
```

Реализация интерфейса IEnumerator<T>



Обязательно для обобщенной версии (часть IEnumerator<T>…)
Опционально для необобщенной версии (не часть IEnumerator…)

public interface IEnumerable
public interface IEnumerator

public interface IEnumerable<out T> : System.Collections.IEnumerable
public interface IEnumerator<out T> : IDisposable, System.Collections.IEnumerator

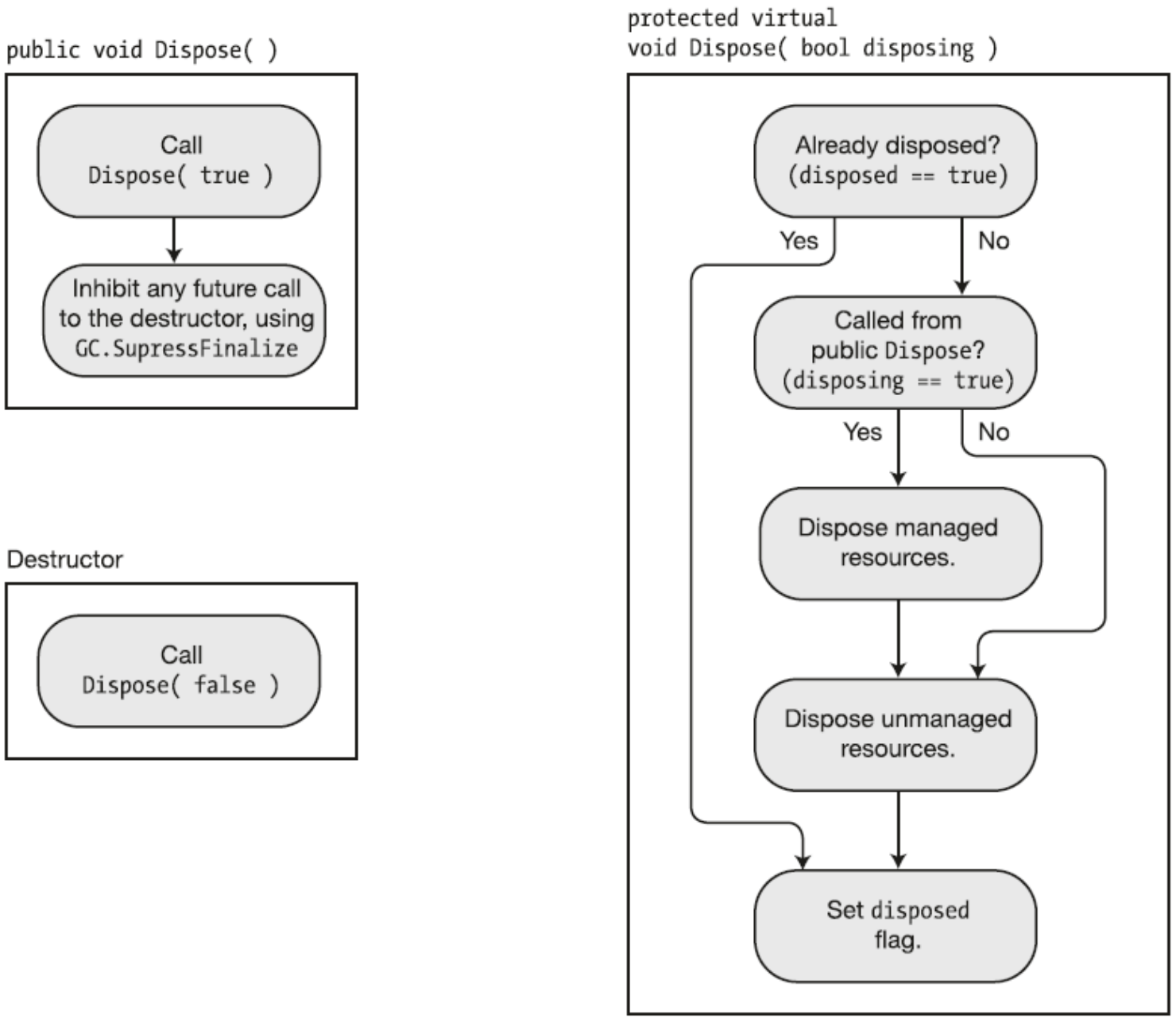
Класс, реализующий интерфейс IEnumerator<T>

```
using System.Collections;
using System.Collections.Generic;

class MyGenEnumerator: IEnumerator< T >    {
  public T Current { get; }              // IEnumerator<T>--Current
          явная реализация
                  ↓
  object IEnumerator.Current { get { ... } } // IEnumerator--Current
  public bool MoveNext() { ... }      // IEnumerator--MoveNext
  public void Reset() { ... }         // IEnumerator—Reset

  public void Dispose() { ... }                  // IDisposable--Dispose
...
}
```

Метод Dispose( )

```
class MyClass                {
bool disposed = false;               // флаг статуса освобождения
//////////////////////////////////////////////////////////
public void Dispose()    {
   if (!disposed)            {          // проверка флага
      // вызвать Dispose для управляемых ресурсов….
      // освободить неуправляемые ресурсы….
   }
   disposed = true;                    // установка флага статуса
   GC.SuppressFinalize(this);      // чтобы GC не вызывал Finalize
}
//////////////////////////////////////////////////////////
~MyClass() // деструктор
{
  if (!disposed)   {            // проверка флага
   // освободить неуправляемые ресурсы
...
}
}
```

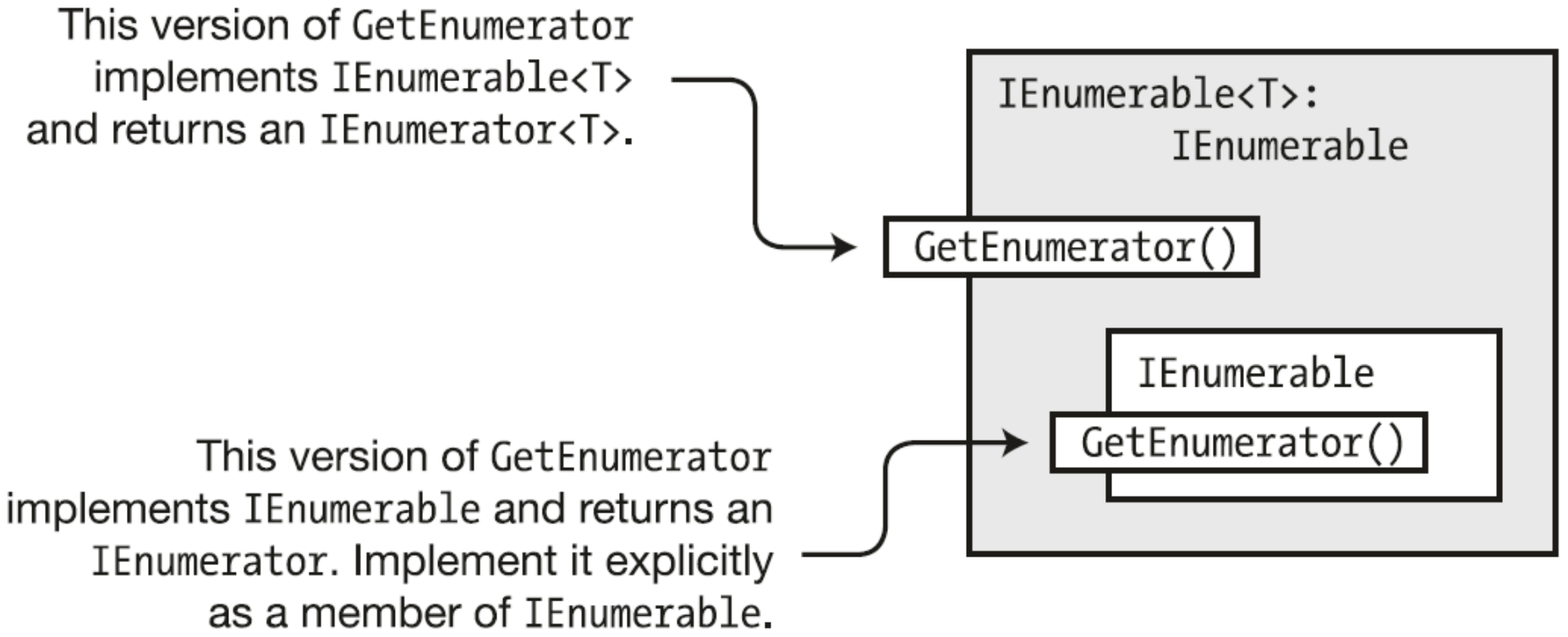
Стандартный паттерн Dispose



Стандартный паттерн Dispose

```
class MyClass: IDisposable  {
        bool disposed = false; // флаг статуса освобождения
        public void Dispose()  { // открытый Dispose
                Dispose( true );
                GC.SuppressFinalize(this);
        }
        ~MyClass()  {    // деструктор
                Dispose(false);
        }

        protected virtual void Dispose(bool disposing)  { // защищенный!
        if (!disposed)  {
                if (disposing)  {
                  // Освободить управляемые ресурсы
                }
          // Освободить неуправляемые ресурсы
        }
        disposed = true;
}
}
```

Реализация интерфейса IEnumerable<T>



Класс, реализующий интерфейс IEnumerable<T>

```
using System.Collections;
using System.Collections.Generic;

class MyGenEnumerable: IEnumerable<T>
{
    public IEnumerator<T> GetEnumerator() { ... } // IEnumerable<T>
                явная реализация
                          ↓
    IEnumerator IEnumerable.GetEnumerator() { ... } // IEnumerable
...
}
```

public interface IEnumerable<out T> : System.Collections.IEnumerable

21

Итераторы и yield return

возврат обобщенного перечислителя

↓

```
public IEnumerator<string> BlackAndWhite() // версия 1
{
        yield return "black";          // yield return
        yield return "gray";           // yield return
        yield return "white";          // yield return
}
```

Итераторы (2)

возврат обобщенного перечислителя

↓

```
public IEnumerator<string> BlackAndWhite() // версия 2
{
    string[] TheColors = { "black", "gray", "white" };
    for (int i = 0; i < TheColors.Length; i++)
        yield return TheColors[i];  // yield return
}
```

Блок с итератором возвращает IEnumerator<T> или IEnumerable<T>

```
// перечислитель (enumerator):
public IEnumerator<string> IteratorMethod()
{       …..
        yield return … ;
}
```

```
// перечисляемое (enumerable):
public IEnumerable<string> IteratorMethod()
{       …..
        yield return … ;
}
```

Использование итератора для создания перечислителя

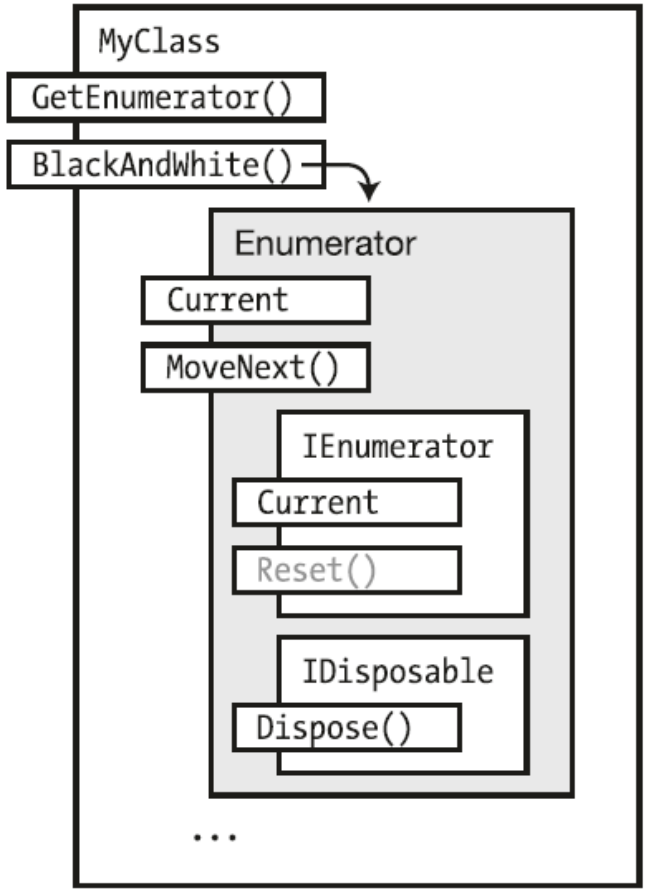
```
class MyClass        {
   public IEnumerator<string> GetEnumerator()     {
      return BlackAndWhite(); // возвращает перечислитель
   }
   public IEnumerator<string> BlackAndWhite()     {  // итератор
      yield return "black";
      yield return "gray";
      yield return "white";
   }
}
class Program        {
   static void Main()  {
      MyClass mc = new MyClass();
      foreach (string shade in mc)
                  Console.WriteLine(shade);
   }
}
```

Блок с итератором для создания перечислителя

```
class Myclass
{
   public IEnumerator<string> GetEnumerator()
   {
      return BlackAndWhite();
   }

   public IEnumerator<string> BlackAndWhite()
   {
      yield return "black";
      yield return "gray";
      yield return "white";
   }
}
```

The iterator construct

produces a method that returns an enumerator.



Использование итератора для создания перечисляемого

```
class MyClass    {
    public IEnumerator<string> GetEnumerator( ){
        IEnumerable<string> myEnumerable =
            BlackAndWhite();
        return myEnumerable.GetEnumerator();
}
    public IEnumerable<string> BlackAndWhite()   {
        yield return "black";
        yield return "gray";
        yield return "white";
    }
}
```

Использование итератора для создания перечисляемого (2)

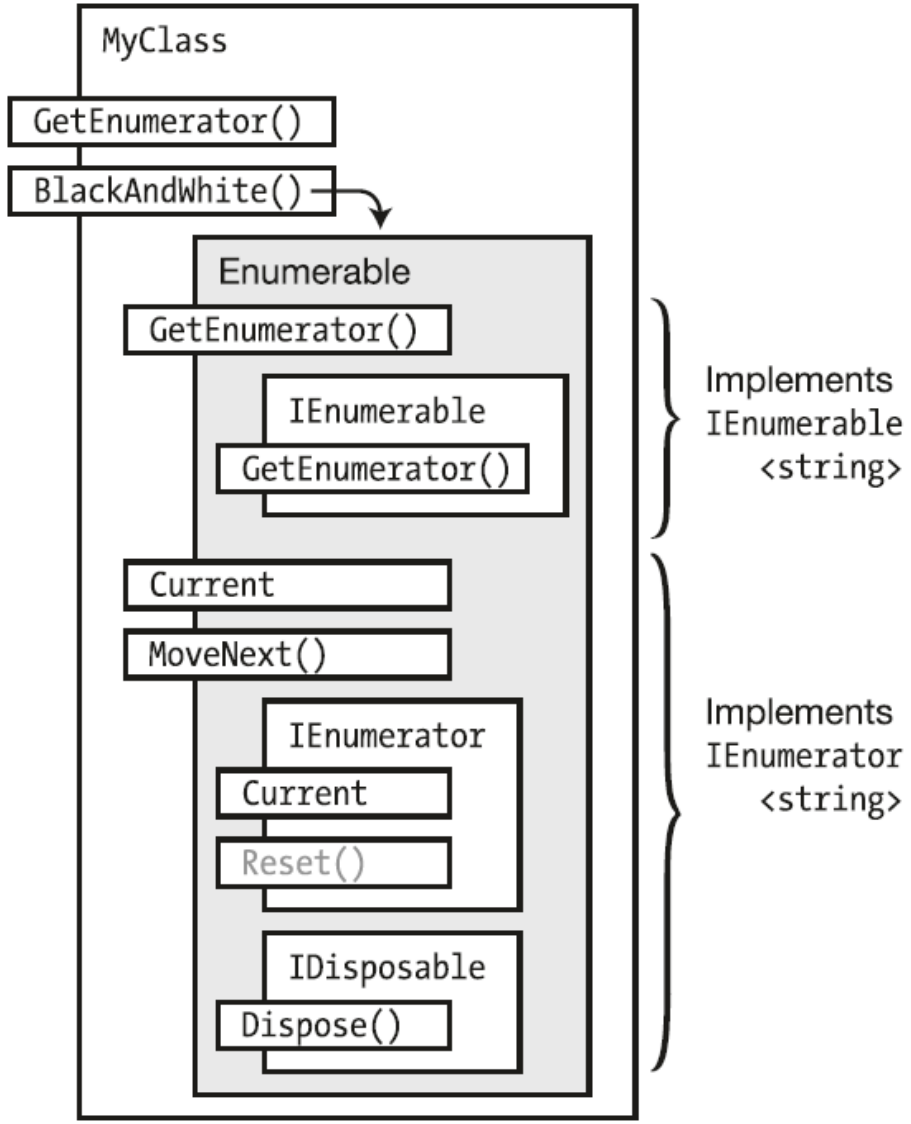
```
class Program    {
static void Main()          {
    MyClass mc = new MyClass();
    foreach (string shade in mc) // используем объект класса
            Console.Write("{0} ", shade);
    // используем метод-итератор класса:
    foreach (string shade in mc.BlackAndWhite())
            Console.Write("{0} ", shade);
    }
}
```

Результат работы:
black gray white black gray white

An iterator block that produces Enumerable

```
class Myclass
{
   public IEnumerator<string> GetEnumerator()
   {
      IEnumerable<string> myEnumerable =
                              BlackAndWhite();
      return myEnumerable.GetEnumerator();
   }

   public IEnumerable<string> BlackAndWhite()
   {
      yield return "black";
      yield return "gray";
      yield return "white";
   }
}
```

Iterator



Общий паттерн итератора с перечислителем (1)

```
// шаблон проектирования итератора
class MyClass   { // утиная типизация
        public IEnumerator<string>  GetEnumerator()
        {
                return IteratorMethod();
        }
        public IEnumerator<string> IteratorMethod()
        {
                yield return … ;
        }
}
void Main()       {
        MyClass mc = new MyClass();
        foreach (string s in mc)
        …..
        foreach (string s in mc.IteratorMethod())
        …..

}
```

Общий паттерн итератора с перечисляемым (2)

```
// шаблон проектирования итератора
class MyClass   { // утиная типизация
        public IEnumerator<string>  GetEnumerator()
        {
                return IteratorMethod().GetEnumerator();
        }
        public IEnumerable<string> IteratorMethod()
        {
                yield return … ;
        }
}
void Main()       {
        MyClass mc = new MyClass();
        foreach (string s in mc)
        …..
        foreach (string s in mc.IteratorMethod())
        …..
}
```

Множественные перечисляемые (1)

```
using System;
using System.Collections.Generic;

class ColorCollection  {
    string[ ] Colors={"Red", "Orange", "Yellow",
                      "Green", "Blue", "Purple"};
    public IEnumerable<string> Forward() { // перечислимое
        for (int i = 0; i < Colors.Length; i++)
            yield return Colors[i];
    }
    public IEnumerable<string> Reverse() { // перечислимое
        for (int i = Colors.Length - 1; i >= 0; i--)
        yield return Colors[i];
    }
}
```

Множественные перечисляемые (2)

```
static void Main( )   {
      ColorCollection cc = new ColorCollection();
      foreach (string color in cc.Forward())
           Console.Write("{0} ", color);
      Console.WriteLine( );
      foreach (string color in cc.Reverse())
          Console.Write("{0} ", color);

      Console.WriteLine( );

      IEnumerable<string> ieable = cc.Reverse();
      IEnumerator<string> ieator = ieable.GetEnumerator();
      while (ieator.MoveNext( ))
          Console.Write("{0} ", ieator.Current);
      Console.WriteLine( );
}
```

Множественные перечислители (1)

```
class MyClass: IEnumerable<string>  {
        bool ColorFlag = true;
        public MyClass(bool flag)                    // конструктор
        {  ColorFlag = flag;  }
        IEnumerator<string> BlackAndWhite  // свойство
        { get  {   yield return "black";
                   yield return "gray";
                   yield return "white";     }
        }
        IEnumerator<string> Colors                   // свойство
        {  get  {    string[] theColors = { "blue", "red", "yellow" };
                     for (int i = 0; i < theColors.Length; i++)
                         yield return theColors[i];
                }
        }
```

Множественные перечислители (2)

```
    public IEnumerator<string> GetEnumerator( )
    {  return ColorFlag  ?
                Colors                      // возврат перечислителя Colors
                : BlackAndWhite;            // возврат перечислителя BlackAndWhite
    }

    System.Collections.IEnumerator
                System.Collections.IEnumerable.GetEnumerator( )
    {  return ColorFlag  ?
                Colors              // возврат перечислителя Colors
                : BlackAndWhite; // возврат перечислителя BlackAndWhite
    }
}
```

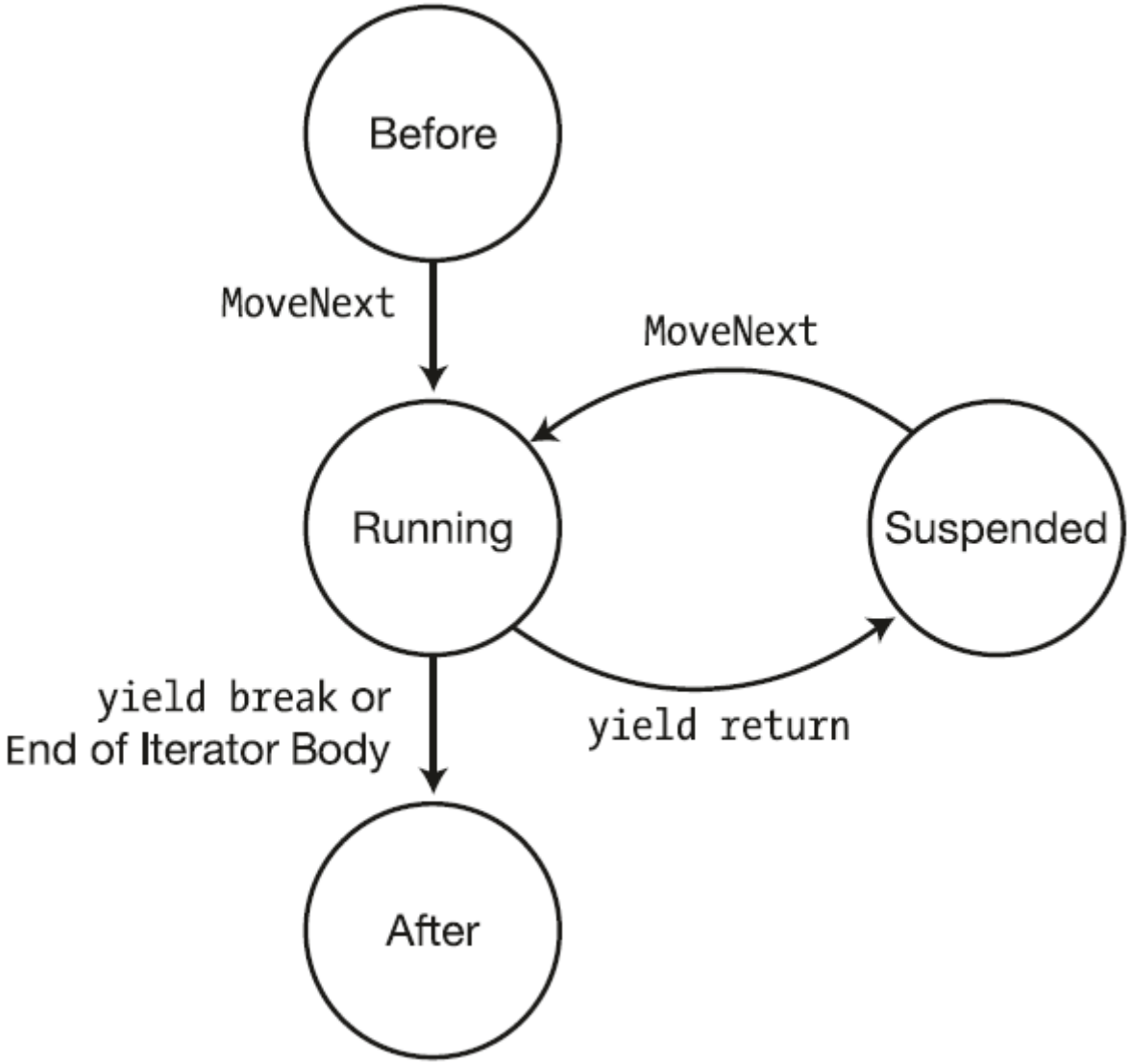
Множественные перечислители (3)

```
static void Main()  {
   MyClass mc1 = new MyClass( true ); // вызов конструктора
   foreach (string s in mc1)
            Console.Write("{0} ", s);
   Console.WriteLine();
   MyClass mc2 = new MyClass( false ); // вызов конструктора
   foreach (string s in mc2)
           Console.Write("{0} ", s);
   Console.WriteLine();
}
```

Результат работы программы:

```
blue red yellow
black gray white
```

Итератор, как конечный автомат



Класс с итератором 1

```
class Numbers {
        public uint Numb { get; set; }
        public Numbers(uint n) { Numb = n; }
        public IEnumerator<uint> GetEnumerator() {
            return IterMethod();   }
        public IEnumerator<uint> IterMethod()  {
            uint newX = Numb ;
            do {    uint d = newX % 10;
                    yield return d;
                    newX = newX / 10;
            }
            while (newX != 0);
        }
}
```

Класс с итератором 2

```
class Digits {
        public uint Numb { get; set; }
        public Digits(uint n) { Numb = n; }
        public IEnumerator<uint> GetEnumerator()  {
            return IterMethod().GetEnumerator();  }
        public IEnumerable<uint> IterMethod()  {
            List<uint> list = new List<uint>();
            uint newX = Numb ;
            do {    uint d = newX % 10;
                    list.Add(d);
                    newX = newX / 10;       }
            while(newX != 0);
            for(int i=list.Count-1; i != -1; i--)
                yield return list[i];
        }
}
```

Вложенные итераторы

```
public IEnumerable<uint> IterMethodEvenIndexOnly
                                        (IEnumerable<uint> seq)
{
        int i = 0;
        try
        {
            foreach (var item in seq)
            {
                if (i++ % 2 == 0)
                    yield return item;
                // yield break; // завершает перечисление
            }
        }
        // catch (Exception ex) { }          // нельзя использовать try-catch!
        finally {                            // использовать try-finally можно!
                // в finally нельзя использовать yield!
        }
}
```