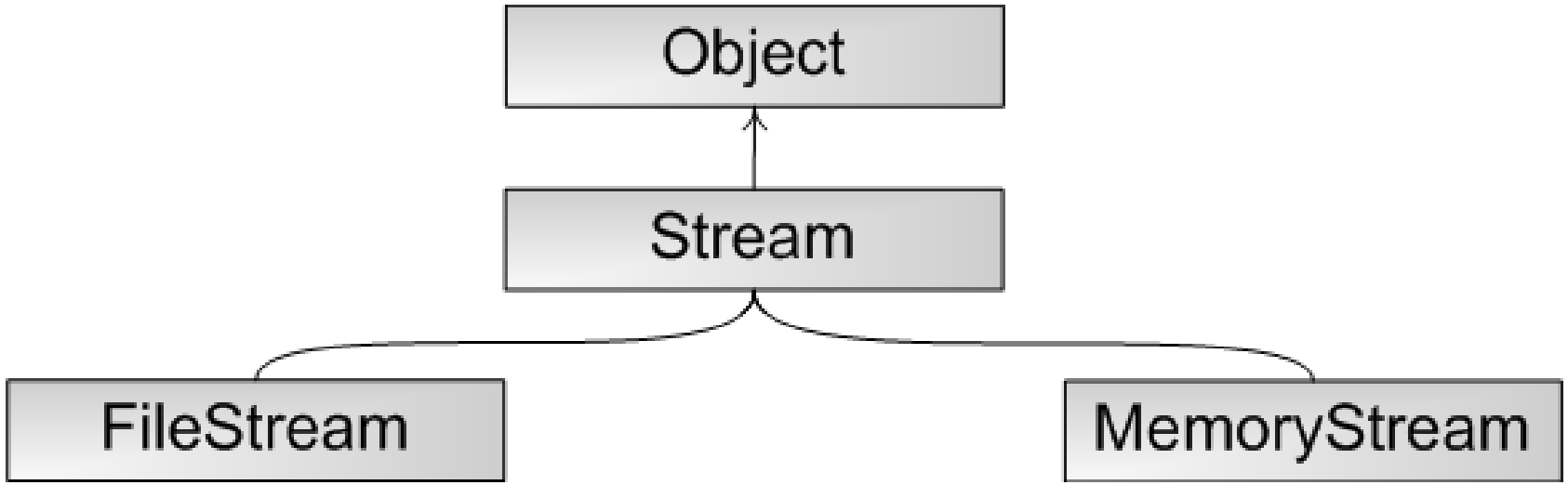
Файлы и потоки

Потоки данных

- ***Поток данных – абстракция, обозначающая источник получения или средство приема данных, позволяющая унифицировать процессы обмена данными между программой и внешним окружением.***



Методы абстрактного класса Stream

Имя	Назначение
Close()	Закрывает поток и освобождает ассоциированные с ним ресурсы.
Flush()	Освобождает буфер обмена, предварительно передав из него данные в информационный источник.
Read() **ReadByte()**	Читает байт или последовательность байтов из потока и перемещает текущую позицию на прочитанное число байтов
Seek()	Устанавливает текущую позицию в потоке
SetLength()	Устанавливает длину потока
Write() **WriteByte()**	Записывает в поток байт или последовательность байтов, перемещает указатель на количество записанных байтов

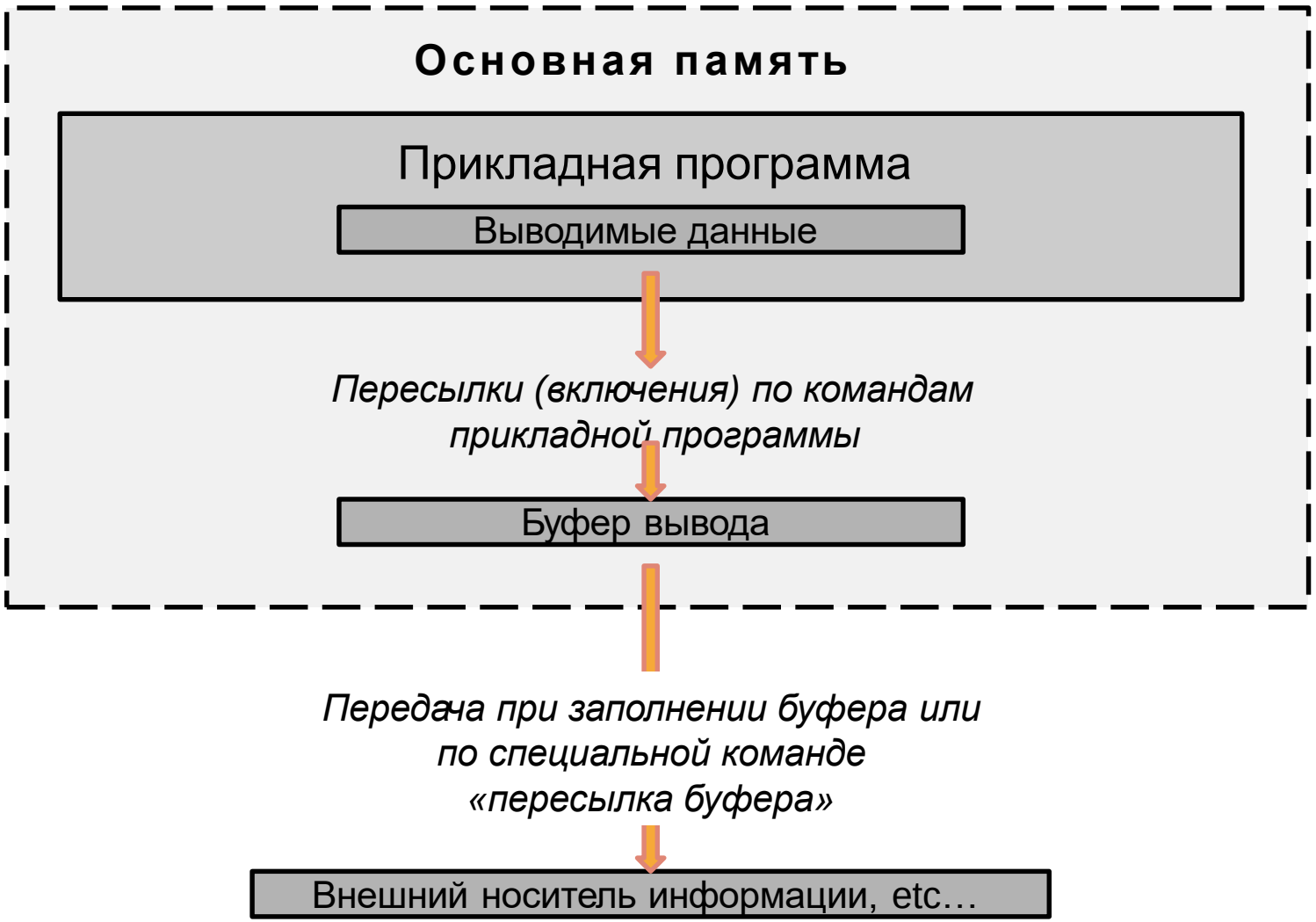
Поддерживается IDisposable.

Назначение метода **Flush( )**

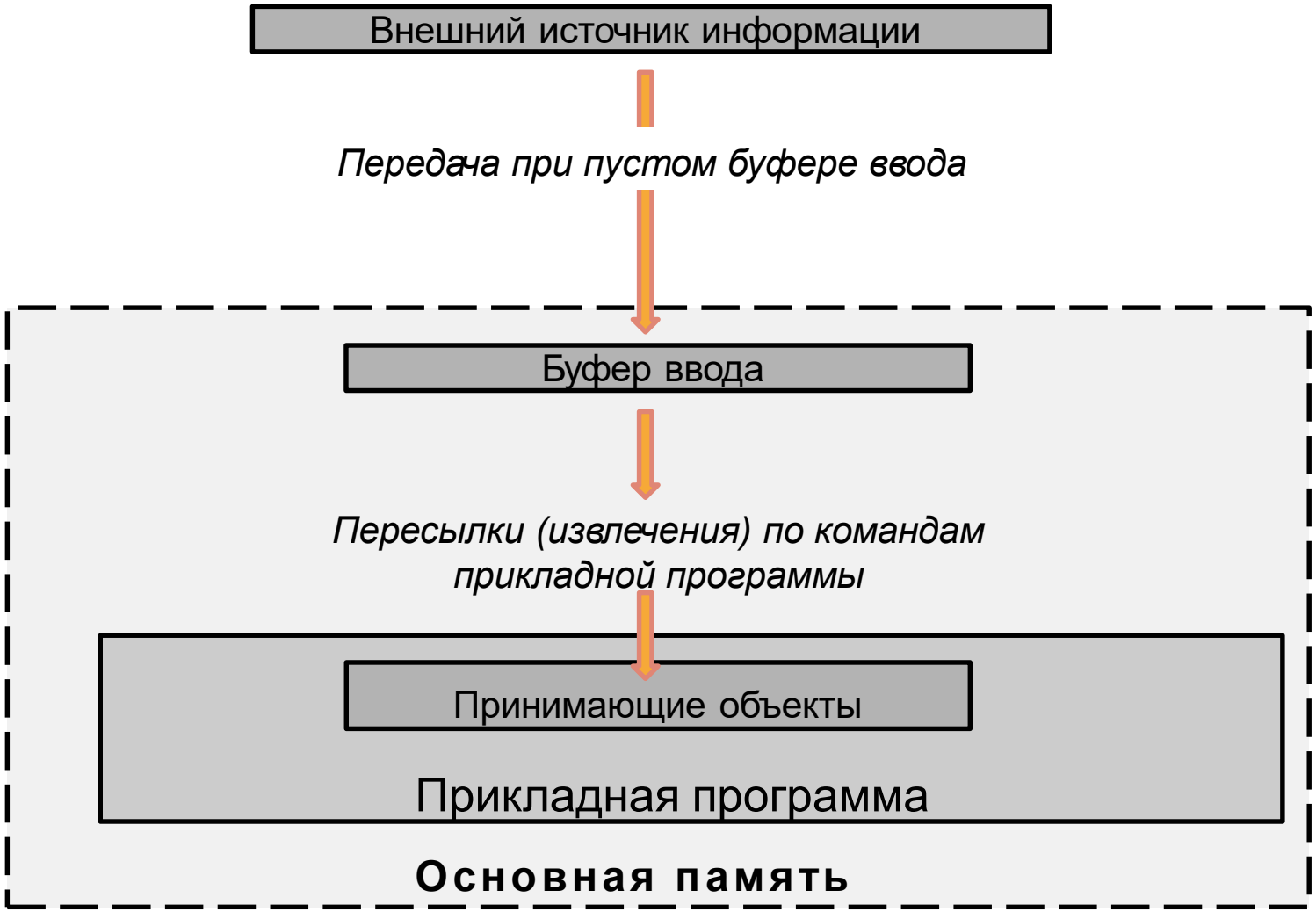
Свойства абстрактного класса Stream

Имя	Назначение
CanRead **CanSeek** **CanWrite**	Проверяют для потока возможность чтения, записи и возможность изменения текущей позиции
Length	Длина потока в байтах
Position	Текущая позиция в байтах

Выходной поток данных



Входной поток данных



Возможности класса FileStream

Для создания экземпляра класса FileStream необходимо явно или неявно указать:

1. Файл, с которым будет ассоциирован поток;
2. Способ доступа к файлу (**FileAccess**);
3. Режим открытия файла (**FileMode**);
4. Допуск к совместному использованию (**FileShare**).

Конструкторы класса FileStream

FileStream (string имя_файла, FileMode режим)

FileStream (string имя_файла, FileMode режим, FileAccess доступ)

Пример:

```
FileStream f1 = new FileStream("data.dat", FileMode.Create);

FileStream f2 = new FileStream("data.bin", FileMode.Open,  FileAccess.Write);
```

Методы классов *File* и *FileInfo*, возвращающие ссылку на экземпляр *FileStream*

Имя	Назначение
Create()	Создает новый файл и поток, ассоциированный с созданным файлом. Либо создает поток, ассоциированный с существующим файлом.
Open()	Открывает файл и возвращает ссылку на поток, ассоциированный с ним. С помощью параметра может быть указано назначение потока (чтение, запись и.т.д)
OpenRead()	Открывает файл и создает поток, предназначенный только для чтения.
OpenWrite()	Открывает файл и создает поток, предназначенный только для записи.

Применение методов классов *FileInfo*, *FileStream*

```
// создаём объект, но не файл
FileInfo fi1 = new FileInfo("fileTest.txt");

// создаём файл и байтовый поток
FileStream fs1 = fi1.Open(FileMode.OpenOrCreate);

// создаются и потоковый объект и файл
FileStream fs = File.Create(@"C:\!\test.txt");

fi1.Delete(); // удалить файл, представленный объектом fi1

fs1.Close(); //закрыть поток

fs.Dispose();   // закрываем поток
```

Важно помнить: FileStream резализует интерфейс IDisposable.

Пример работы с классом FileStream

```
using System;
using System.IO;
  FileInfo fi = new FileInfo("Alphabet.txt");
  using (FileStream fs = fi.Open(FileMode.OpenOrCreate)) {
      long len = fs.Length;   // Размер файла
      if (len == 26)
          Console.WriteLine("Алфавит собран!");
      else {
          if (len == 0)
              Console.WriteLine("Файл пуст!");
          fs.Seek(len, SeekOrigin.Begin);
          byte bt = (byte)('A' + len);
          fs.WriteByte(bt);
          Console.WriteLine("Добавляем в файл букву " + (char)bt);
      }
      Console.WriteLine("Буквы в файле:");
      fs.Seek(0, SeekOrigin.Begin);
      int u;
      while ((u = fs.ReadByte()) != -1)
          Console.Write((char)u + " ");
      Console.WriteLine();
  }
```

Нет необходимости в:
fs.Flush();
fs.Close();
fs = null;

Пример работы с классом FileStream

Результаты первого выполнения программы:
Файл пуст!
Добавляем в файл букву A
Буквы в файле:
A
****
Результаты шестого выполнения программы:
Добавляем в файл букву F
Буквы в файле:
A B C D E F
****
Результаты после двадцать шестого выполнения программы:
Алфавит собран!
Буквы в файле:
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Чтение и запись байтов

void *WriteByte****(byte)***
int *ReadByte* ***()***

void *Write****(byte[ ]*** *массив****,***
int *начало****,*** ***int*** *длина****)***

int *Read* ***(byte[ ] массив,***
int *начало****,*** ***int*** *длина****)***

Если достигнут конец файла или файл пуст, то методы Read() и ReadByte() возвращают -1

Пример работы с классом FileStream

```
// Задача: вывод всех цифр из исходного текста программы
using System;
using System.IO;

static void Main() {
    using (FileStream inFi =
        new FileStream(@"..\..\..\Program.cs", FileMode.Open)) {
         int t;         // числовое значение прочитанного байта
         int k = 0;     // позиция байта в потоке (в файле)
         while ((t = inFi.ReadByte()) != -1) {
             if (t >= '0' && t <= '9')
                 Console.WriteLine(t + " - " + (char)t + " - " + k);
             k++;
         }   // while
    }   // using
}   // Main()
```

Результаты выполнения программы:

```
48 - 0 - 267
49 - 1 - 370
48 - 0 - 406
57 - 9 - 423
```

Оператор using

using (*аргумент*)
{
    операторы
}

«Аргумент» для оператора **using** должен иметь тип, реализовавший интерфейс **System.IDisposable.**

Эквивалент оператора **using**:

try { . . . . }
finally {
    аргумент.Dispose();
}

Эквивалент оператора using для предыдущей программы

```
FileStream inFi = null;
try {
    inFi =
        new FileStream(@"..\..\..\Program.cs", FileMode.Open);

. . .
}
finally
{
     (inFi as IDisposable)?.Dispose();
}
```

Замена вызова Dispose():

```
inFi.Flush();  // Освободить буфер
inFi.Close();  // Закрыть поток и файл
inFi = null;   // Обнулить ссылку на поток
```

Преобразования в массив байтов.

System.BitConverter

***static byte*[ ] *GetBytes*(*тип_значения*)**

Типы параметров:

Boolean, Char, Double, Int16, Int32, Int64, Single, Uint16, Uint32, Uint64.

Байтовое представление целого:

```
byte[ ] bin = BitConverter.GetBytes(4097);
foreach (var b in bin)
    Console.Write(b+"-");
```

Результат на экране: 1-16-0-0-

Преобразования из массива байтов.
System.BitConverter

static char** ToChar**(byte**[ ] *массив*, **int** *начало**)

static double** ToDouble**(byte**[ ] *массив*, **int** *начало**)
ToInt16(), ToInt32(), ToInt64(), ToSingle(), ToUint16(), ToUint32(), ToUint64()

***static string** ToString**(byte**[ ] *массив*[, **int** *начало*
[int *длина*]])***

Пример работы с классом BitConverter

```
static void Main() {
    int numb = 100;
    byte[] binArray = BitConverter.GetBytes(numb);
    string str = BitConverter.ToString(binArray);
    Console.WriteLine("str = " + str);
    int res = BitConverter.ToInt32(binArray, 0);
    Console.WriteLine("res = " + res);
    char ch = BitConverter.ToChar(binArray, 0);
    Console.WriteLine("ch = " + ch);
}
```

Результат выполнения программы:

str = 64-00-00-00

res = 100

ch = d

Запись в файл двоичных данных

```
using System;
using System.IO;

static void Main() {
    double doublePI = Math.PI;
    byte[] biPI = BitConverter.GetBytes(doublePI);
    using (FileStream fileStream = new FileStream(@"..\..\..\PiFile.bin",
        FileMode.OpenOrCreate, FileAccess.ReadWrite)) {
            fileStream.Write(biPI, 0, 8);
    }
    Console.WriteLine("Сохранили код числа PI.");
}
```

Чтение из файла двоичных данных

```
using System;
using System.IO;

// Чтение из файла числа
private static void Main() {
    if (!File.Exists(@"..\..\..\PiFile.bin")) {
        Console.WriteLine("Файл отсутствует!");
        return;
    }
    byte[] arrayPI = new byte[8];
    using (FileStream fileStream = new FileStream(@"..\..\..\PiFile.bin",
       FileMode.Open, FileAccess.ReadWrite)) {
        fileStream.Read(arrayPI, 0, 8);
    }
    double restorePI = BitConverter.ToDouble(arrayPI, 0);
    Console.WriteLine("Из файла восстановлено число PI: " + restorePI);
}
```

Байтовые потоки и BitConverter

```
// Запись и чтение двоичных данных
static void Main() {
    FileInfo file = new FileInfo(@"..\..\..\ints.bin"); // создан объект
    using (FileStream fs = file.Create()) {   // создан файл и поток
        int next,       // Очередное число
        number = 10,    // Общее количество чисел в файле
        pattern = 3;      // Число, задающее кратность
        byte[] bin = null;   // Массив байтов для кода целого
        Random gen = new Random();
        for (int i = 0; i < number; i++) {
            next = gen.Next(1000);
            bin = BitConverter.GetBytes(next);// байтовое представление
            fs.Write(bin, 0, bin.Length);// запись в файл
        }
        fs.Flush();

       // Чтение данных – см. след слайд

    }
}
```

Байтовые потоки и BitConverter

```
fs.Position = 0;     // Вернуться в начало файла (потока)

long lenFs = fs.Length; // Определить размер файла (потока)

for (int k = 0; k < lenFs / 4; k++) {
    fs.Read(bin, 0, bin.Length); // прочитать 4 байта
    int decod = BitConverter.ToInt32(bin, 0); // получить число
    if (decod % pattern == 0)
        Console.WriteLine("decod=" + decod);
}
```

Результат выполнения программы:

decod=171
decod=183
decod=930
decod=117