НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

# Объектно-ориентированное программирование на языке Java

## I / O в Java.


Yevhen Berkunskyi, NUoS
eugeny.berkunsky@gmail.com
http://www.berkut.mk.ua
Tatyana Smykodub, NUoS
tgsmyk@gmail.com

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

# Input & Output

- Создание хорошей системы ввода / вывода (I / O) является одной из наиболее сложных задач для разработчика языка.
- В прикладном программном интерфейсе Java API объект, из которого можно читать последовательность байтов, называется потоком ввода, а объект, в который можно записывать последовательность байтов, — потоком вывода.
- В роли таких источников и приемников последовательностей байтов чаще всего выступают файлы, но могут также служить сетевые соединения и даже блоки памяти. Абстрактные классы InputStream и OutputStream служат основанием для иерархии классов ввода-вывода.
- Но начнем тему с класса File …

# Класс File (java.io)

- Класс File имеет обманчивое имя; вы можете подумать, что это относится к файлу, но это не так.

*Фактически, «FilePath» было бы лучшим именем для класса.*

- Он может представлять либо имя определенного файла, либо имена набора файлов в каталоге. Если это набор файлов, вы можете запросить этот набор, используя метод list (), который возвращает массив String.
- Этот класс не содержит методы для работы с содержимым файла, но позволяет манипулировать такими свойствами файла, как право доступа, дата и время создания, путь в иерархии каталогов, создание, удаление файла, изменение его имени и каталога и т. д.

# Пакет java.nio.file

- Пакет java.nio.file определяет интерфейсы и классы виртуальной машины Java для доступа к файлам, атрибутам файлов и файловым системам.
- Этот API может использоваться для преодоления многих ограничений класса java.io.File. Метод toPath может использоваться для получения Path, который использует абстрактный путь, представленный объектом File, чтобы найти файл.
- Результирующий Path может использоваться с классом Files для обеспечения более эффективного и расширенного доступа к дополнительным файлам, атрибутам файлов и исключениям ввода-вывода для диагностики ошибок при сбое операции в файле.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

# Класс File (java.io)

Объект класса **File** создается одним из нижеприведенных способов:

```
File obFile = new File("\\com\\file.txt");
File obDir = new File("c:/jdk/src/java/io");
File obFile1 = new File(obDir, "File.java");
File obFile2 = new File("c:\\com", "file.txt");
File obFile3 = new File(new URI("Интернет-адрес"));
```

- В первом случае создается объект, соответствующий файлу, во втором — подкаталогу. Третий и четвертый случаи идентичны.
- Для создания объекта  указывается каталог и имя файла. В пятом — создается объект, соответствующий адресу в Интернете.

При создании объекта класса **File** любым из конструкторов компилятор не выполняет проверку на существование физического файла с заданным путем.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

# Пример

```
package by.bsu.io;
import java.io.File;
import java.util.Date;
import java.io.IOException;
public class FileTest {
public static void main(String[] args) {
//с объектом типа File ассоциируется файл на диске FileTest2.java
File fp = new File("FileTest2.java");
if(fp.exists()) {
                System.out.println(fp.getName() + " существует");
if(fp.isFile()) { // если объект – дисковый файл
  System.out.println("Путь к файлу:\t" + fp.getPath());
  System.out.println("Абсолютный путь:\t"+ fp.getAbsolutePath());
  System.out.println("Размер файла:\t" + fp.length());
  System.out.println("Последняя модификация :\t"+ new
                                    Date(fp.lastModified()));
System.out.println("Файл доступен для чтения:\t"+ fp.canRead());
System.out.println("Файл доступен для записи:\t"+ fp.canWrite());
System.out.println("Файл удален:\t" + fp.delete());
}}
```

# Пример

```
else
System.out.println("файл " + fp.getName() + " не
                                        существует");
try {
   if(fp.createNewFile()) {
     System.out.println("Файл " + fp.getName() + " создан");
   }
} catch(IOException e) {
              System.err.println(e);
}
// в объект типа File помещается каталог
// в корне проекта должен быть создан каталог com.learn с
//несколькими файлами
File dir = new File("com" + File.separator + "learn");
  if (dir.exists() && dir.isDirectory()) { /*если объект
       является каталогом и если этот каталог существует */
       System.out.println("каталог " + dir.getName() + "
                                              существует");
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

# Пример

```
File[] files = dir.listFiles();
for(int i = 0; i < files.length; i++) {
Date date = new Date(files[i].lastModified());
System.out.print("\n"+files[i].getPath()+" \t|
"+files[i].length()+"\t|"+date);
// использовать toLocaleString() или toGMTString()
}
// метод listRoots() возвращает доступные корневые каталоги
File root = File.listRoots()[1];
System.out.printf("\n%s %,d из %,d свободно.",
root.getPath(), root.getUsableSpace(),
root.getTotalSpace());
}
}
```

# Input и output

- Классы библиотеки Java ввода-вывода делятся на ввод и вывод, согласно иерархии классов в документации JDK.
- Через наследование все производные от классов **InputStream** или **Reader** имеют базовые методы, называемые **read** () для чтения одного байта или массива байтов.
- Аналогично, все производные от классов **OutputStream** или **Writer** имеют базовые методы, называемые **write** () для записи одного байта или массива байтов.

# InputStream & OutputStream

- В java.io разработчики библиотеки начали с решения, что все классы, имеющие какое-либо отношение к вводу, будут унаследованы от **InputStream**, а все классы, связанные с выводом, будут унаследованы от **OutputStream**.

# Типы InputStream

В классе **InputStream** определены свойства, общие для байтовых потоков ввода.

Базовый класс **InputStream** представляет классы, которые получают данные из различных источников:

- массив байтов
- строка (String)
- файл
- канал (pipe): данные помещаются с одного конца и извлекаются с другого
- последовательность различных потоков, которые можно объединить в одном потоке
- другие источники (например, подключение к интернету)

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

# Обзорная таблица классов Java IO

| | С битами | | С символами | |
|---|---|---|---|---|
| | Ввод | Вывод | Ввод | Вывод |
| Базовые | InputStream | OutputStream | Reader<br>InputStreamReader | Writer<br>OutputStreamWriter |
| Массивы | ByteArrayInputStream | ByteArrayOutputStream | CharArrayReader | CharArrayWriter |
| Файлы | FileInputStream<br>RandomAccessFile | FileOutputStream<br>RandomAccessFile | FileReader | FileWriter |
| Каналы | PipedInputStream | PipedOutputStream | PipedReader | PipedWriter |
| Буфер | BufferedInputStream | BufferedOutputStream | BufferedReader | BufferedWriter |
| Фильтрация | FilterInputStream | FilterOutputStream | FilterReader | FilterWriter |
| Синтакс. анализ Parsing | PushbackInputStream<br>StreamTokenizer | | PushbackReader<br>LineNumberReader | |
| Строки | | | StringReader | StringWriter |
| Работ а с примитивами | DataInputStream | DataOutputStream | | |
| | | PrintStream | | PrintWriter |
| Работа с сериал. объектами | ObjectInputStream | ObjectOutputStream | | |
| Объединение InputStream | SequenceInputStream | | | |

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

# InputStream

Для работы с указанными источниками используются подклассы базового класса **InputStream**:

- `BufferedInputStream` - Буферизированный входной поток
- `ByteArrayInputStream` – Позволяет использовать буфер в памяти (массив байтов) в качестве источника данных для входного потока.
- `DataInputStream` - Входной поток, включающий методы для чтения стандартных типов данных
- `JavaFileInputStream` - Для чтения информации из файла
- `FilterInputStream` - Абстрактный класс, предоставляющий интерфейс для классов-надстроек, которые добавляют к существующим потокам полезные свойства.
- `InputStream` - Абстрактный класс, описывающий поток ввода
- `ObjectInputStream` - Входной поток для объектов
- `StringBufferInputStream` - Превращает строку (String) во входной поток данных InputStream

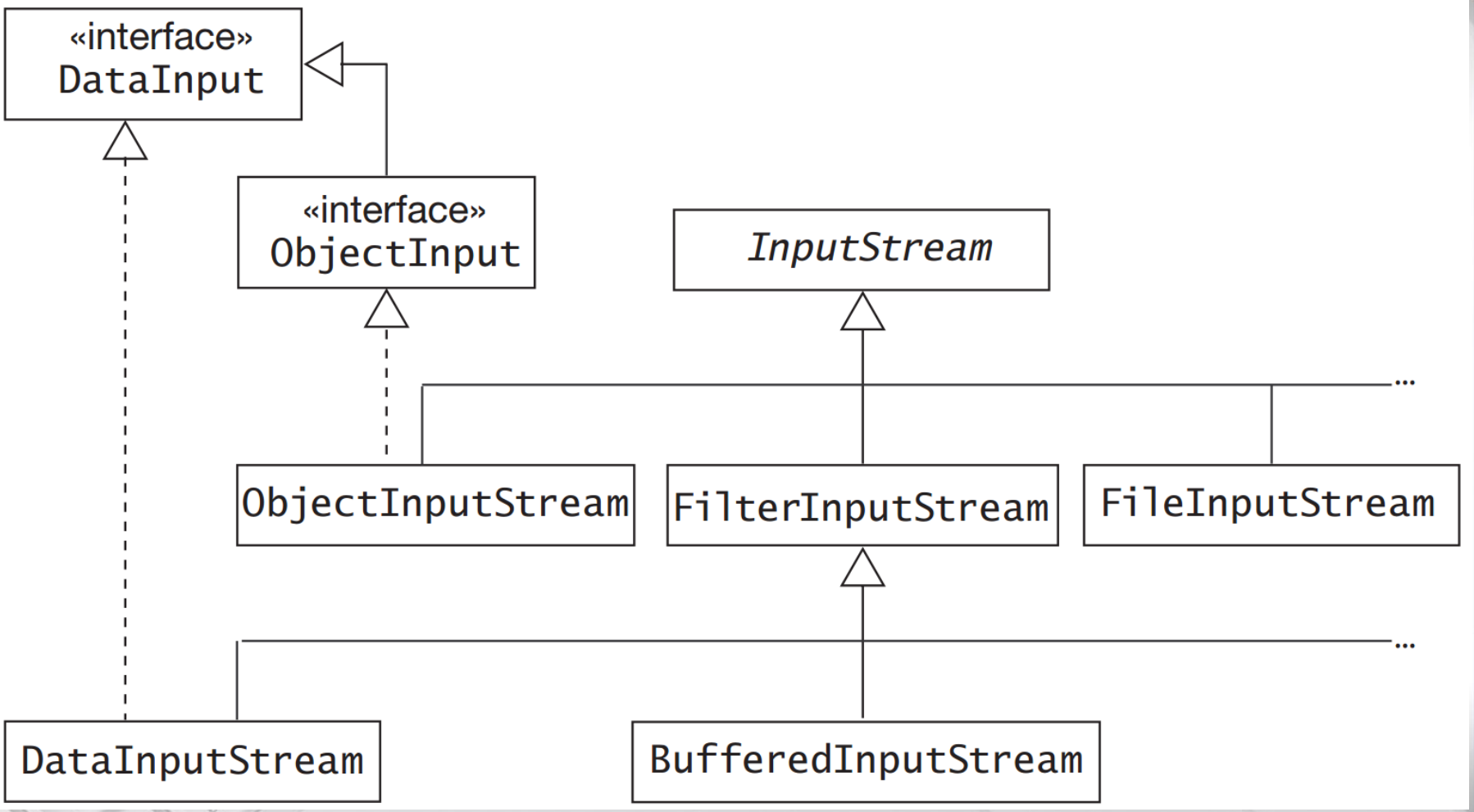
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

# OutputStream

Класс **OutputStream** - это абстрактный класс, определяющий потоковый байтовый вывод. В этой категории находятся классы, определяющие, куда направляются ваши данные: в массив байтов (но не напрямую в String; предполагается что вы сможете создать их из массива байтов), в файл или канал.

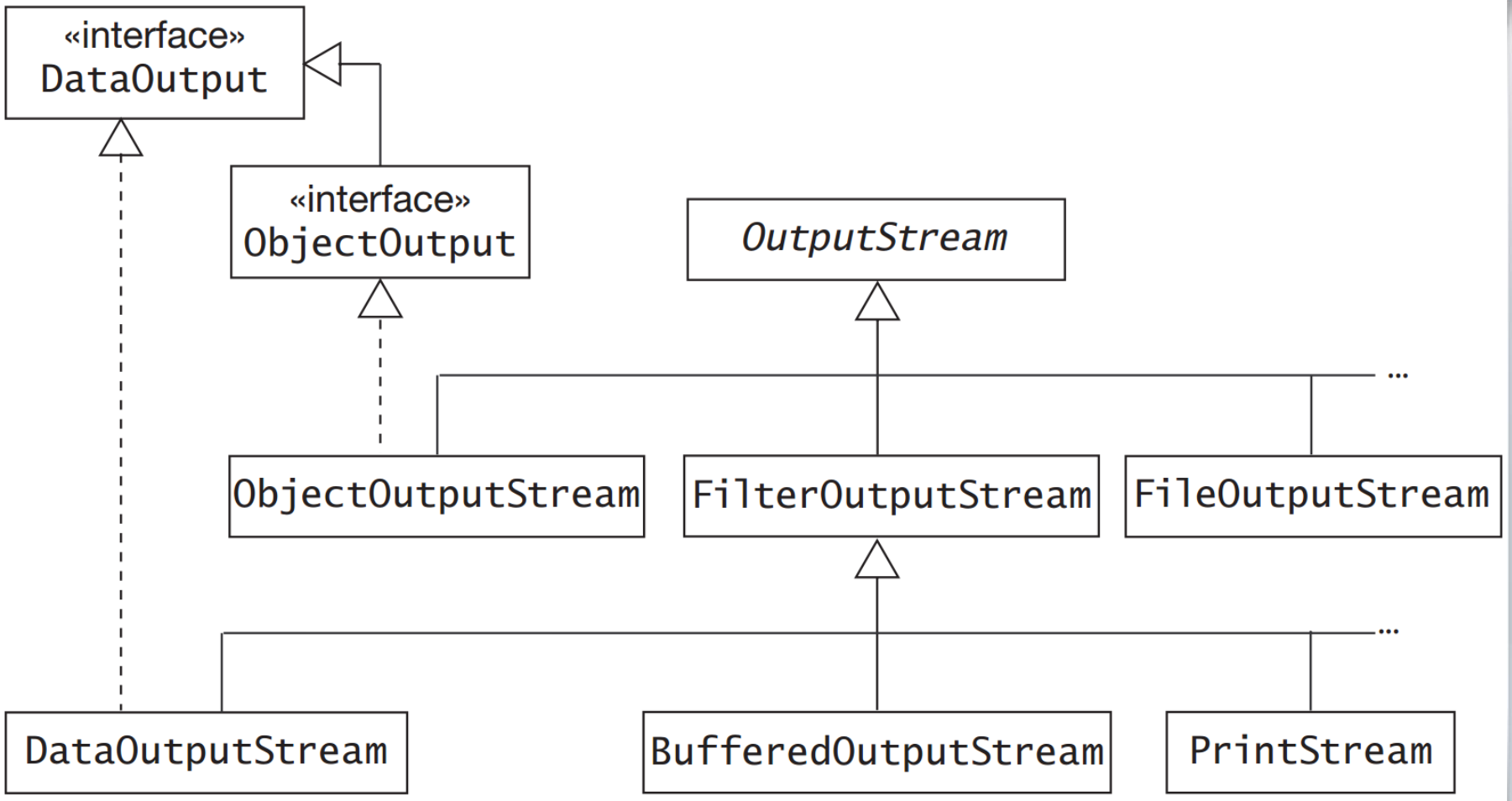
- `BufferedOutputStream` - Буферизированный выходной поток
- `ByteArrayOutputStream` - Создает буфер в памяти. Все данные, посылаемые в этот поток, размещаются в созданном буфере
- `DataOutputStream` - Выходной поток, включающий методы для записи стандартных типов данных
- `JavaFileOutputStream` - Отправка данных в файл на диске. Реализация класса OutputStream
- `ObjectOutputStream` - Выходной поток для объектов
- `PipedOutputStream` - Реализует понятие выходного канала.
- `FilterOutputStream` - Абстрактный класс, предоставляющий интерфейс для классов-надстроек, которые добавляют к существующим потокам полезные свойства.

# Иерархия InputStreams



НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

# Иерархия OutputStreams



НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

# Символьные потоки: чтение и запись

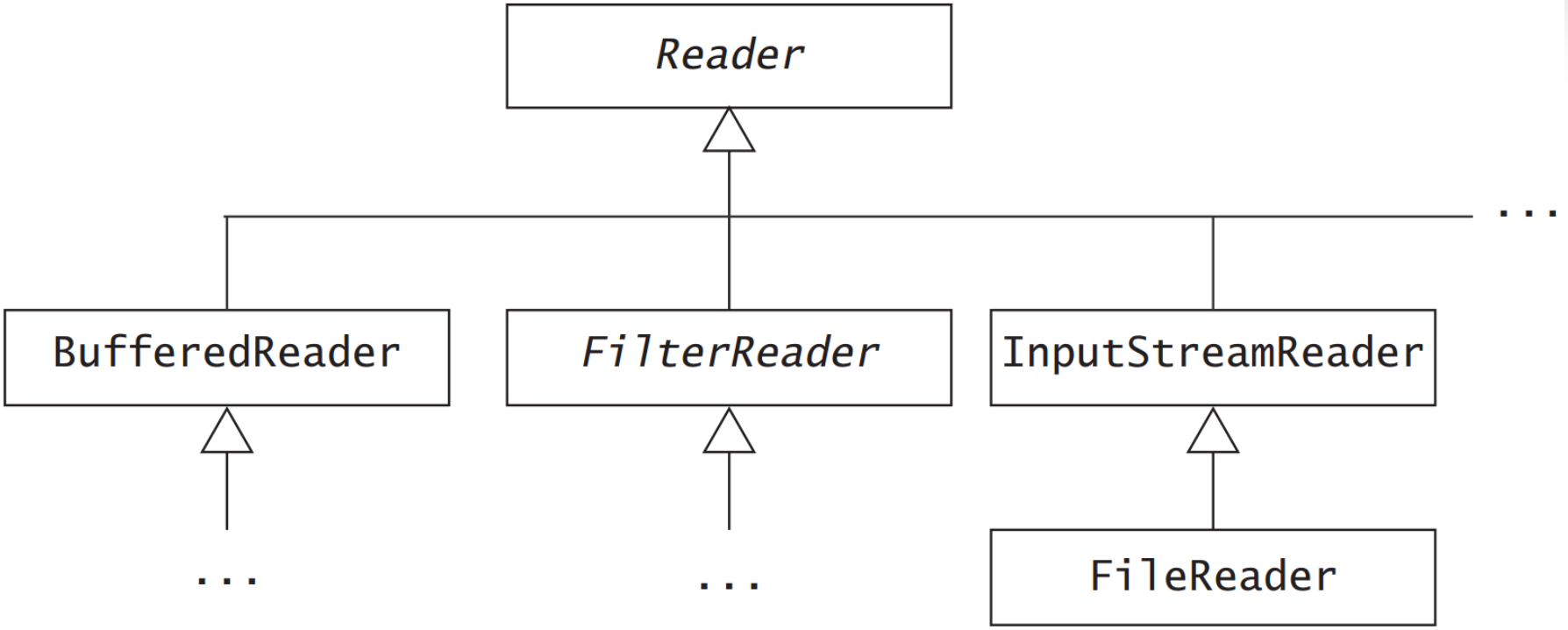
- Причиной появления классов **Reader** и **Writer** стала интернационализация, старая библиотека поддерживала 8 битное представление символов и неправильно иногда работала с 16-ти. Для правильной обработки символьных потоков в формате Unicode применяется отдельная иерархия подклассов абстрактных классов **Reader** и **Writer**, которые почти полностью повторяют функциональность байтовых потоков, но являются более актуальными при передаче текстовой информации.
- **Reader** представляет собой поток входных символов, который считывает последовательность символов Unicode, а **Writer** - это выходной поток символов, который записывает последовательность символов Unicode .

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

# Чтение

| | |
|---|---|
| BufferedReader | Читатель, который буферизует символы, читаемые от основного читателя. Необходимо указать базовый читатель и указать необязательный размер буфера. |
| InputStreamReader | Символы считываются из байтового входного потока, который должен быть указан. Кодировка символов по умолчанию используется, если явно не указано кодирование символов. |
| FileReader | Читает символы из файла, используя кодировку по умолчанию.<br>Файл может быть задан объектом File, FileDescriptor или именем файла String. Он автоматически создает FileInputStream, связанный с файлом. |

# Character Streams: Readers and Writers

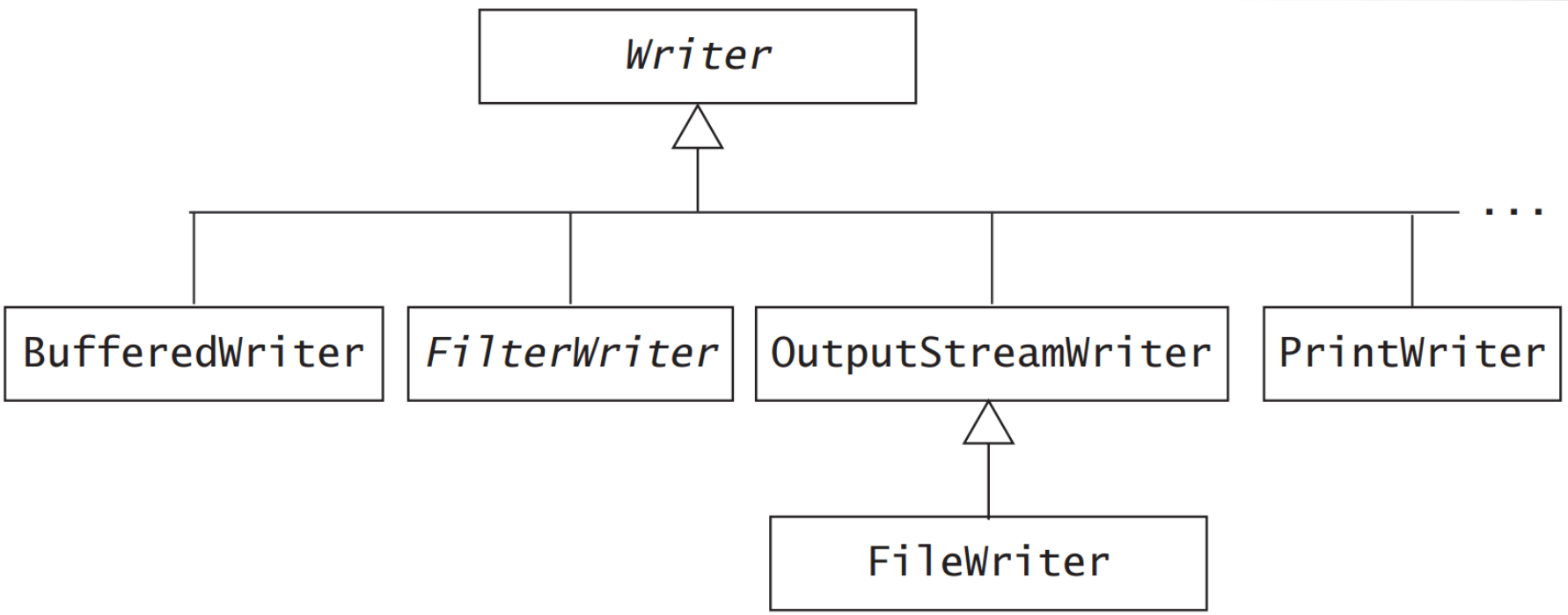


НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

# Запись

| | |
|---|---|
| BufferedWriter | Писатель, который буферизирует символы, прежде чем записывать их в основной писатель. Основной писатель должен быть указан, и может быть указан необязательный размер буфера. |
| OutputStreamWriter | Символы записываются в поток байтов, который должен быть указан. Используется кодировка символов по умолчанию, если не указано явное кодирование символов. |
| FileWriter | Записывает символы в файл, используя кодировку по умолчанию. Файл может быть задан объектом File, FileDescriptor или именем файла String. Он автоматически создает FileOutputStream, связанный с файлом |
| PrintWriter | Фильтр, позволяющий записывать текстовое представление объектов Java и примитивных типов Java в основной поток. Необходимо указать основной выходной поток или писатель. |

# Character Streams: Readers and Writers



# Примеры

Java поддерживает несколько типов потоков, но независимо от того, какой тип вы собираетесь использовать, в вашей программе должны быть выполнены следующие три шага:

- ОТКРОЙТЕ поток, который указывает на некоторое хранилище данных.
- Выполните ЧТЕНИЕ или ЗАПИСЬ данных из / в этот поток.
- ЗАКРОЙТЕ поток (Java может сделать это автоматически).

В реальном мире люди используют разные типы труб для переноса различного содержимого (например, газа, масла, молочных коктейлей). Аналогичным образом, Java-программисты используют разные классы в зависимости от типа данных, которые им необходимы для переноса потока. Некоторые потоки используются для переноса текста, некоторые для байтов и т. д.

# Примеры Потоки байтов

При написании программы, которая считывает данные из файла, а затем отображает их на экране, необходимо знать, данные какого типа в этом файле содержатся.
С другой стороны, программа, которая просто копирует файлы из одного места в другое, может даже не знать, что в них - изображения, текст или же музыка. Такая программа считывает первоначальный файл в оперативную память, а затем записывает его в папку назначения байт за байтом, используя классы Java FileInputStream или FileOutputStream.

Следующий пример показывает, как использовать класс FileInputStream для чтения файла

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

# Примеры. Чтение байтов.

```
import java.io.FileInputStream;
import java.io.IOException;
 public class MyByteReader {

  public static void main(String[] args) {

  try (FileInputStream myFile =                    //(1)
         new FileInputStream("abc.dat")){
     int byteValue;
     while ((byteValue = myFile.read()) != -1) {// (2)
           System.out.print(byteValue + " ");  // (3)
           }
  } catch (IOException ioe) {                      // (4)
         System.out.println("Could not read file: " +
                            ioe.getMessage());
         }
   }
}
```

# Примеры

1. Сначала мы открываем файл для чтения, создавая экземпляр класса FileInputStream, передавая имя файла abc.dat в качестве параметра. Когда есть некоторые программные ресурсы, которые необходимо открыть и закрыть (например, FileInputStream), просто создайте их в круглых скобках сразу после ключевого слова try, и Java закроет их для вас автоматически. Это избавляет вас от написания блока finally, содержащего код для закрытия ресурсов.
2. Эта строка, вызывает метод read (), который считывает один байт, присваивает его значение переменной byteCode и сравнивает его с -1 (конец индикатора файла).
3. Если значение в текущем байте не равно -1, мы печатаем его с символом пробела.
4. Если возникает IOException, поймайте его и напечатайте причину этой ошибки.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

# Примеры. Запись байтов.

```
import java.io.FileOutputStream;
import java.io.IOException;
public class MyByteWriter {
    public static void main(String[] args) {
      int someData[] = {56, 230, 123, 43, 11, 37};   //(1)

       try (FileOutputStream myFile = new
              FileOutputStream("qwerty.dat")) {   //(2)
         int arrayLength = someData.length;
         for (int i = 0; i < arrayLength; i++) {
              myFile.write(someData[i]);        //(3)
           }
        } catch (IOException ioe) {
           System.out.println("Could not write into the
                  file: " + ioe.getMessage());    // (4)
       }
    }
}
```

# Примеры. Запись байтов.

1. В классе MyByteWriter у нас есть целочисленный массив someData, заполненный целыми числами
2. С помощью ***оператора try с ресурсами*** работаем с файлом **`qwerty.dat`**. Главное преимущество оператора try с ресурсами заключается в том, что он предотвращает ситуации, в которых файл (или другой ресурс) непреднамеренно остается неосвобожденным и после того, как необходимость в его использовании отпала.
3. Затем переходим к записи элементов массива в файл.
4. Если ошибка возникает, мы ее поймаем и сообщим причину.

Примеры вышеописанного кода читают или записывают в файл по одному байту за раз. Один вызов чтения читает один байт, и один вызов записи записывает один байт. В общем, доступ к диску намного медленнее, чем обработка, выполняемая в памяти; поэтому не рекомендуется обращаться к диску тысячу раз, чтобы прочитать файл размером 1000 байт. Чтобы минимизировать количество обращений к диску, Java предоставляет буферы, которые служат хранилищами данных.

# Примеры. Буфер. Чтение байтов.

Класс BufferedInputStream работает как посредник между FileInputStream и самим файлом. Он считывает большой кусок байтов из файла в память (буфер) за один раз, а затем объект FileInputStream считывает из него отдельные байты.

```
import java.io.BufferedInputStream;
import java.io.FileInputStream;
import java.io.IOException;
public class MyBufferedByteReader {
   public static void main(String[] args) {
      try (FileInputStream myFile = new    FileInputStream("abc.dat"); // (1)
         BufferedInputStream buff = new   BufferedInputStream(myFile);){
         int byteValue;
         while ((byteValue = buff.read()) != -1) { //(2)
            System.out.print(byteValue + " ");
         }
      } catch (IOException e) {     e.printStackTrace();
      }  }    }
```

# Примеры. Буфер. Чтение байтов.

Здесь мы снова используем синтаксис ***try с ресурсами***. На этот раз мы создаем экземпляр FileInputStream, а затем и экземпляр BufferedInputStream , получающий FileInputStream в качестве аргумента.

1. Так мы соединяемся с фрагментами «трубы», которая использует файл abc.dat в качестве источника данных.

2. BufferedInputStream считывает с диска  большой фрагмент данных в буфер памяти, а затем  метод buff.read() читает по одному байту за раз из буфера.

Программа MyBufferedByteReader выполняет тот же вывод, что и MyByteReader, но будет работать немного быстрее.

# Примеры.

Класс ***Files*** упрощает и ускоряет выполнение типичных операций над файлами. Например, содержимое всего файла нетрудно прочитать следующим образом:

```
byte [] myFileBytes =
  Files.readAllBytes(Paths.get("abc.dat));
```

Если файл содержит текст, вы можете прочитать файл в переменную String следующим образом:

```
String myFileText =
          new String (Files.readAllBytes
               (Paths.get ("abc.dat")));
```

# Примеры. Символьные потоки.

Приведенные выше простые методы предназначены для обращения с текстовыми файлами умеренной длины. Если же файл крупный или двоичный, то для обращения с ним можно воспользоваться потоками ввода-вывода или чтения и записи данных, как показано ниже. Их методы избавляют от необходимости обращаться непосредственно к классам FileInputStream, FileOutputStream, BufferedReader или BufferedWriter.

```
InputStream in = Files.newInputStream(path);
OutputStream out = Files.newOutputStream(path);
Reader in = Files.newBufferedReader(path, charset);
Writer out = Files.newBufferedWriter(path, charset);
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

# Примеры. Чтение.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public class MyTextFileReader {
    public static void main(String[] args){

        Path path = Paths.get("abc.dat");      //(1)
        System.out.println("The absolute path is " +
                path.toAbsolutePath());
 try {
      if ( Files.exists(path)){          // (2)
      System.out.println("The file size is " +
                                          Files.size(path));
            }
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

# Примеры. Символьные потоки.

```
 BufferedReader bufferedReader=   // (3)
    Files.newBufferedReader(path, StandardCharsets.UTF_8);

        String currentLine;
        while ((currentLine =
              bufferedReader.readLine()) != null){    // (4)
              System.out.println(currentLine);
         }
} catch (IOException ioe) {
           System.out.println("Can't read file: " +
                    ioe.getMessage());
        }

        System.out.println(           //    (5)
                "Your default character encoding is " +
                                  Charset.defaultCharset());
    }
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

# Примеры. Символьные потоки.

1. Программа начинается с создания объекта Path из abc.dat и вывода на экран его абсолютного пути.

2. Затем проверяем, существует ли файл, представленный этим путем, и печатаем его размер в байтах.

3. Для чтения используем  метод *newBufferedReader*, способный читать текст, закодированный с помощью набора символов UTF-8.

4. Чтение текстовых строк из буфера.

5. Печатаем кодировку по умолчанию.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

# Примеры. Запись. Символьные потоки.

Для больших файлов используйте BufferedWriter, например:

```
import java.io.BufferedWriter;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.time.LocalDateTime;

public class MyTextFileBufferedFileWriter {
   public static void main(String[] args) {

        String myScore = "My game score is 28000 " +
LocalDateTime.now() + "\n";    //(1)

        Path path = Paths.get("scores.txt");  //(2)

        try (BufferedWriter writer =              // (3)
                    getBufferedWriter(path)) {
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

# Примеры. Запись. Символьные потоки.

```
        writer.write(myScore);          // (4)
            System.out.println("The game score was saved at
" + path.toAbsolutePath());
        } catch (IOException ioe) {
            System.out.println("Can't write file: " +
                    ioe.getMessage());
        }
    }

    private static BufferedWriter getBufferedWriter(Path
path) throws IOException{

        if (Files.exists(path)) {      //(5)
            return Files.newBufferedWriter(path,
                    StandardOpenOption.APPEND);
        } else {
            return Files.newBufferedWriter(path,
                    StandardOpenOption.CREATE);
        }    }     }
```

# Примеры. Запись. Символьные потоки.

В этом классе код, который проверяет, существует ли файл помещен в отдельный метод getBufferedWriter.

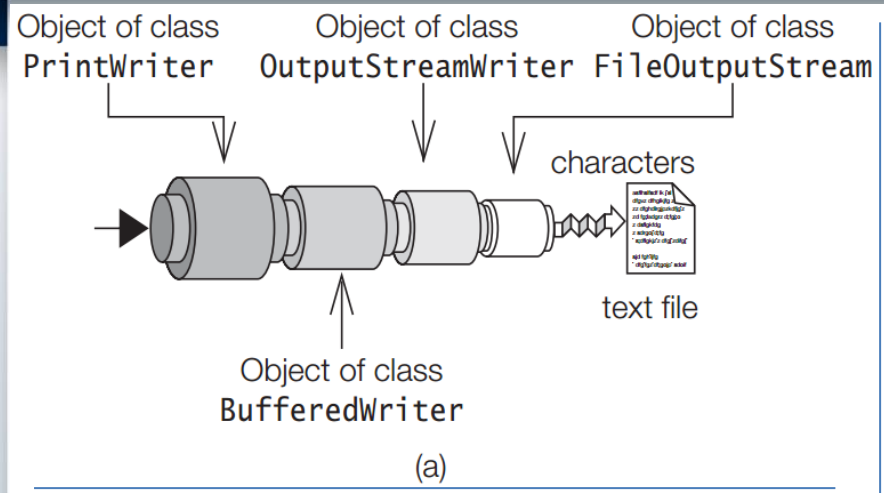
Такой подход называется Паттерн Factory Method (шаблон проектирования фабрик). Фабрика может строить разные объекты, не так ли? Метод getBufferedWriter также создает разные экземпляры BufferedReader в зависимости от наличия файла, указанного в пути. На жаргоне программистов методы, которые создают и возвращают разные экземпляры объектов, основанные на каком-то параметре, называются фабриками.

# Примеры. Запись. Символьные потоки.

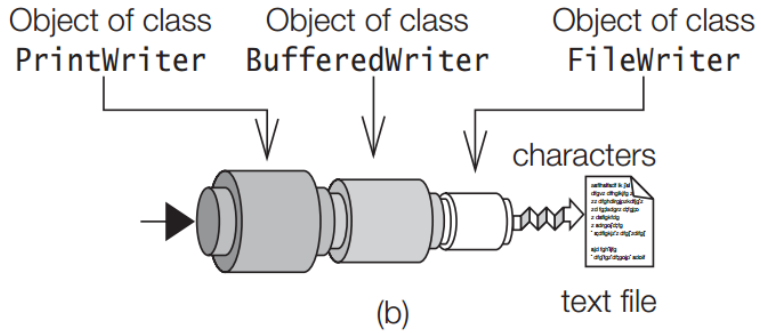
1. Присваиваем строке значение **`"My game score is 28000 "`** и объединяет с текущей датой, временем, и символом \ n.
2. Созданием объект Path, указывающий на файл scores.txt.
3. Для записи больших объёмов, используйте BufferedWriter
4. Записываем строку в буфер.
5. Проверяем, существует ли файл score.txt, добавляем к нему новый контент. Если файл не существует - создаем его и записываем в него контент.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

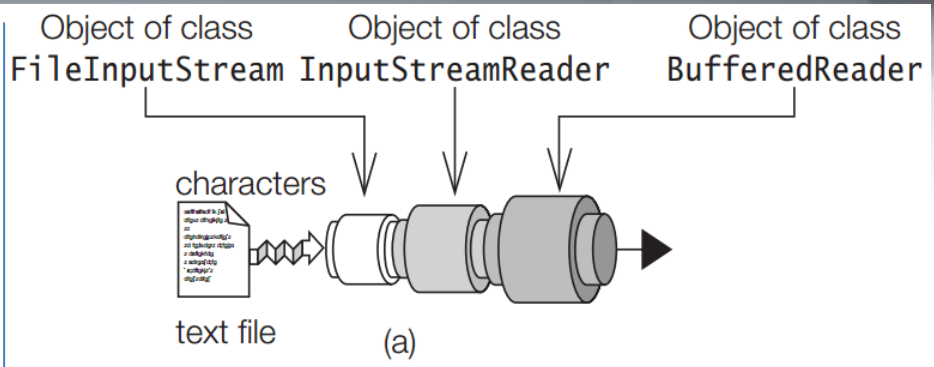
# Using Buffered Writers & Readers



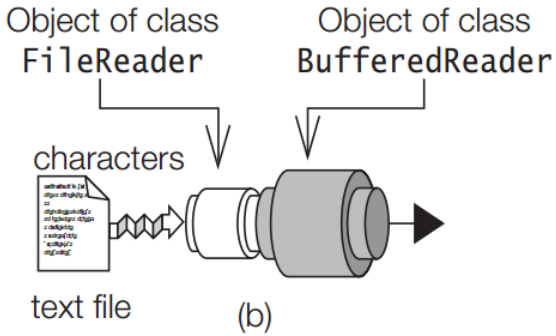

(a)




(b)




(a)




(b)

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

# Итоги

Для примеров понадобится файл с данными:

file.txt

```
1 линия
2 линия2
3 линия3
4 линия4444
```

# Buffered Reader

Одним из старых способов считывания данных построчно есть **BufferedReader**

```
public static void main(String[] args) {
    StringBuilder lines = new StringBuilder();
    Path path = Paths.get("file.txt");
    try (BufferedReader reader = Files.newBufferedReader(path)) {
        String line;
        while ((line = reader.readLine()) != null)
            lines.append(line).append("\n");
    } catch (IOException ignored) {
    }
    System.out.println(lines); // write your code here
}
```

# Scanner

Еще одним вариантом считывания есть **Scanner**

```
public static void main(String[] args) {
    StringBuilder lines = new StringBuilder();
    Path path = Paths.get("file.txt");

    try (Scanner scanner = new Scanner(path)) {
        while (scanner.hasNext())
            lines.append(scanner.nextLine()).append("\n");
    } catch (IOException ignored) {
    }

    System.out.println(lines);
}
```

# readAllBytes

В JDK7 пакет NIO был значительно обновлен. Давайте рассмотрим пример с использованием класса *Files* и метода *readAllBytes* . Метод *readAllBytes* принимает *Path.*

```
public static void main(String[] args) {
    String lines = "";
    Path path = Paths.get("file.txt");

    try {
        lines = new String(Files.readAllBytes(path));
    } catch (IOException ignored) {
    }

    System.out.println(lines);
}
```

# Stream

В **Java 8** в класс **Files** добавили метод **lines()** который возвращает **Stream** из строк

```
public static void main(String[] args) {
        StringBuilder lines = new StringBuilder();
        Path path = Paths.get("file.txt");

        try (Stream<String> lineStream = Files.lines(path)) {

            lineStream.forEach(str -> lines.append(str).append("\n"));

        } catch (IOException ignored) {
        }

System.out.println(lines);}
```

# Stream

Еще вариант:

```
public static void main(String[] args) {
    String lines = new String();
    Path path = Paths.get("file.txt");
    try (Stream<String> lineStream = Files.lines(path)) {
       lines= lineStream.collect(Collectors.joining("\n"));
    } catch (IOException ignored) {
    }
    System.out.println(lines);
}
```

# Stream

В **Java 8** метод **lines()** так же добавили в класс **BufferedReader**

```
public static void main(String[] args) {
   public static void main(String[] args) {
    List<String> lines = new ArrayList<>();
            Path path = Paths.get("file.txt");

    try (Stream<String> lineStream =
            Files.newBufferedReader(path).lines()) {
                lines = lineStream.collect(Collectors.toList());

    } catch (IOException ignored) {
    }

    System.out.println(lines);}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

# А если данные числа?

```
public static void main(String[] args) {
      String lines = new String();
      Path path = Paths.get("file.txt");
      try (Stream<String> lineStream = Files.lines(path)) {
         lines= lineStream.collect(Collectors.joining(" "));
      } catch (IOException ignored) {
      }
      System.out.println(lines);

     String [] temp= lines.strip().split(" |\t");
     int [] mas =new int[temp.length];
     int sum=0;
      for (int i = 0; i < temp.length; i++) {
         mas [i] =Integer.parseInt(temp[i]);
         sum+=mas[i];
      }
      System.out.println(Arrays.toString(mas));
      System.out.println("sum="+sum);
   }}
```

Если в файле хранятся числовые данные. Мы сначала считываем их и компонуем в одну большую строку, а потом парсим и преобразовываем в нужный тип

P.S. Для знака табуляции могут возникнуть проблемы (обращайте внимание на кодировку текста). В примере текст набран в Notepad кодировка UTF-8.