НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Объектно-ориентированное программирование на языке Java

I / O в Java. Продолжение.

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

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());
         }
   }
}
```

Пояснение

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)
       }
    }
}
```

Пояснение.

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();
      }  }    }
```

Пояснение.

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.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Еще раз о классе Files.

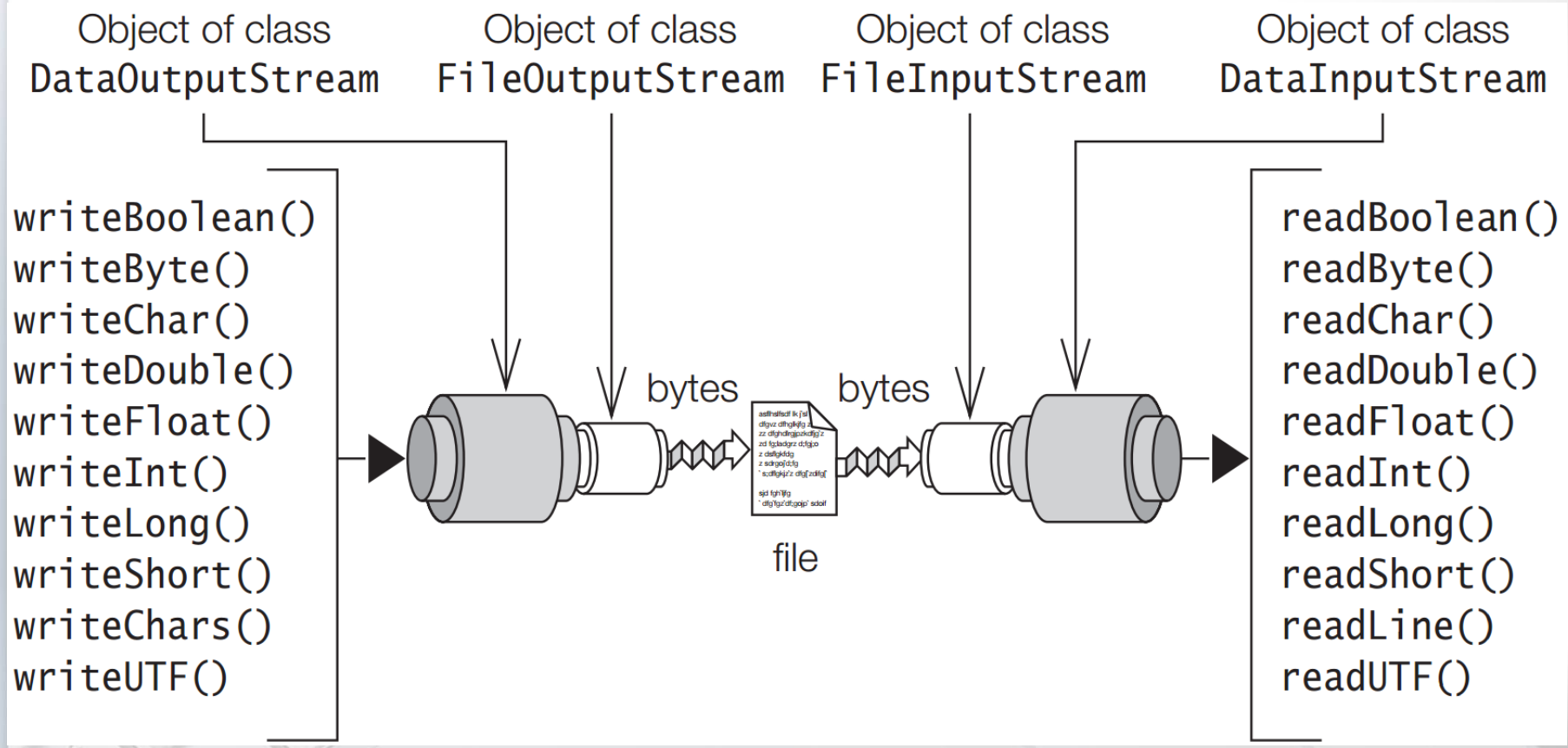
Класс ***Files*** упрощает и ускоряет выполнение типичных операций над файлами. Например, содержимое всего файла нетрудно прочитать следующим образом:

```
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")));
```

А как же примитивные типы?



НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

DataOutputStream & DataInputStream

Класс DataInputStream позволяет считывать примитивы Java (int, float, long и т. д.) Вы передаете InputStream в DataInputStream, а затем уже можете считывать Java-примитивы из DataInputStream. Поэтому класс называется DataInputStream – т.к. считывает данные (числа), а не только байты.

Пример:

```
try(InputStream input = new FileInputStream("data/data.bin");
    DataInputStream dataInputStream =
       new DataInputStream(input)){
int data = dataInputStream.readInt();
int int123 = dataInputStream.readInt();
float float12345 = dataInputStream.readFloat();
long long789 = dataInputStream.readLong();
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

DataOutputStream & DataInputStream

Класс DataOutputStream позволяет записывать примитивы Java в OutputStream вместо байтов. Вы передаете OutputStream в DataOutputStream, а затем пишем примитивы.

Пример:

```
try(OutputStream output = new
      FileOutputStream("data/data.bin");
    DataOutputStream dataOutputStream
         = new DataOutputStream(output)){
dataOutputStream.writeInt(123);
dataOutputStream.writeFloat(123.45F);
dataOutputStream.writeLong(789);
}
```

Примеры. Символьные потоки.

Классы FileReader и FileWriter из пакета java.io были специально созданы для работы с текстовыми файлами, но они работают только с кодировкой символов по умолчанию и не обрабатывают локализацию должным образом. Если вы хотите , использовать другую кодировку символов, тогда эти классы не подойдут.

Для более эффективной работы рекомендуется работать с классами BufferedReader или BufferedWriter, которые считывают или записывают символы из / в буфер потока.

Пример:

```
try(FileReader fileReader = new
       FileReader("c:\\data\\text.txt")){
int data = fileReader.read();
while(data != -1) {
System.out.print((char) data)); data = fileReader.read(); }
}// чтение по 1 символу из файла
```

Примеры. Символьные потоки.

Приведенный выше пример целесообразно применять к 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.

Для двоичных

```
InputStream in = Files.newInputStream(path);
OutputStream out = Files.newOutputStream(path);
```

Для символьных

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

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Символьные потоки. BufferedReader

```
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

Символьные потоки. BufferedReader

```
 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, проверяющий наличие файла, 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.