Введение в программирование на Java

Лекция 17. Работа с файлами.

Виталий Олегович Афанасьев

02 июня 2025

Файловая система и файлы (1)

Файловая система — порядок хранения данных на носителях информации. Файловая система работает с **файлами** и **директориями** — единицами информации, имеющими имя.

Файловая система позволяет связать оборудование (жёсткие диски, USB-флешки) с прикладными программами.

Популярные файловые системы:

- NTFS
- FAT32
- ext2/3/4

Файловая система и файлы (2)

Файловая система может быть представлена в виде древовидной структуры:

- HSE
  - Алгосы
    - task1.py
    - task2.py
    - ...
  - Java
    - ДЗ-1
    - ДЗ-2
- Program Files
  - ...
- Игры
  - Дока2.exe
  - ...

Кодировки

По сути, каждый файл — это просто набор байт.

Но набор байт можно интерпретировать по-разному — всё зависит от **кодировки**.

Примеры:

- ASCII
- UTF-8, UTF-16 (рассматривали на лекции про строки)
- cp1251
- КОИ-8

Пути

В файловых системах существует понятие пути до файла.

Различают **относительные пути** и **абсолютные пути**.

Абсолютные пути:

- Windows: `C:\Users\Вася\Проекты\`
- Linux/MacOS: `/home/vasya/pet-projects`

Относительные пути (задаются относительно текущей **рабочей директории**):

- `some-file.pdf`
- `../../petya/secret-data.txt`

Работа с файловой системой в Java

JVM — это абстракция над современными платформами. Код для неё должен быть платформо-независимым.

Средства стандартной библиотеки (в основном пакеты `java.io` и `java.nio`) позволяют работать с файловой системой не думая о конкретике.

В основном, эти средства вдохновлены Unix-based системами.

Класс File (1)

Класс `java.io.File` представляет абстракцию над файлами в ФС:

```
1 File homework = new File("/home/ivan/HSE/homework.pdf");
2
3
4
5
6
7
8
```

Класс File (1)

Класс `java.io.File` представляет абстракцию над файлами в ФС:

```
1 File homework = new File("/home/ivan/HSE/homework.pdf");
2
3 File relative = new File("HSE/homework.pdf");
4
5
6
7
8
```

Класс File (1)

Класс `java.io.File` представляет абстракцию над файлами в ФС:

```
1 File homework = new File("/home/ivan/HSE/homework.pdf");
2
3 File relative = new File("HSE/homework.pdf");
4
5 File relative = new File("HSE" + File.separator + "homework.pdf");
6
7
8
```

Класс File (1)

Класс `java.io.File` представляет абстракцию над файлами в ФС:

```
1 File homework = new File("/home/ivan/HSE/homework.pdf");
2
3 File relative = new File("HSE/homework.pdf");
4
5 File relative = new File("HSE" + File.separator + "homework.pdf");
6
7 File hseDir = new File("HSE");
8
```

Класс File (1)

Класс `java.io.File` представляет абстракцию над файлами в ФС:

```
1 File homework = new File("/home/ivan/HSE/homework.pdf");
2
3 File relative = new File("HSE/homework.pdf");
4
5 File relative = new File("HSE" + File.separator + "homework.pdf");
6
7 File hseDir = new File("HSE");
8 File homework = new File(hseDir, "homework.pdf");
```

Класс File (2)

```
File file = new File("HSE/sample.txt");
```

Класс File (2)

```
File file = new File("HSE/sample.txt");
String path = file.getPath(); // HSE/sample.txt
```

Класс File (2)

```
File file = new File("HSE/sample.txt");
String path = file.getPath(); // HSE/sample.txt
String name = file.getName(); // sample.txt
```

Класс File (2)

```
File file = new File("HSE/sample.txt");
String path = file.getPath(); // HSE/sample.txt
String name = file.getName(); // sample.txt
String absPath = file.getAbsolutePath(); // /home/vasya/HSE/sample.txt
```

Класс File (2)

```
File file = new File("HSE/sample.txt");
String path = file.getPath(); // HSE/sample.txt
String name = file.getName(); // sample.txt
String absPath = file.getAbsolutePath(); // /home/vasya/HSE/sample.txt
File parent = file.getParentFile(); // File(HSE)
```

Класс File (2)

```
File file = new File("HSE/sample.txt");
String path = file.getPath(); // HSE/sample.txt
String name = file.getName(); // sample.txt
String absPath = file.getAbsolutePath(); // /home/vasya/HSE/sample.txt
File parent = file.getParentFile(); // File(HSE)

boolean exists = file.exists();
```

Класс File (2)

```
File file = new File("HSE/sample.txt");
String path = file.getPath(); // HSE/sample.txt
String name = file.getName(); // sample.txt
String absPath = file.getAbsolutePath(); // /home/vasya/HSE/sample.txt
File parent = file.getParentFile(); // File(HSE)

boolean exists = file.exists();
boolean isDir = file.isDirectory();
```

Класс File (2)

```
File file = new File("HSE/sample.txt");
String path = file.getPath(); // HSE/sample.txt
String name = file.getName(); // sample.txt
String absPath = file.getAbsolutePath(); // /home/vasya/HSE/sample.txt
File parent = file.getParentFile(); // File(HSE)

boolean exists = file.exists();
boolean isDir = file.isDirectory();
boolean isFile = file.isFile();
```

Класс File (2)

```
File file = new File("HSE/sample.txt");
String path = file.getPath(); // HSE/sample.txt
String name = file.getName(); // sample.txt
String absPath = file.getAbsolutePath(); // /home/vasya/HSE/sample.txt
File parent = file.getParentFile(); // File(HSE)

boolean exists = file.exists();
boolean isDir = file.isDirectory();
boolean isFile = file.isFile();
long length = file.length();
```

Класс File (2)

```
File file = new File("HSE/sample.txt");
String path = file.getPath(); // HSE/sample.txt
String name = file.getName(); // sample.txt
String absPath = file.getAbsolutePath(); // /home/vasya/HSE/sample.txt
File parent = file.getParentFile(); // File(HSE)

boolean exists = file.exists();
boolean isDir = file.isDirectory();
boolean isFile = file.isFile();
long length = file.length();

File[] subfiles = parent.listFiles();
```

Класс File (2)

```
File file = new File("HSE/sample.txt");
String path = file.getPath(); // HSE/sample.txt
String name = file.getName(); // sample.txt
String absPath = file.getAbsolutePath(); // /home/vasya/HSE/sample.txt
File parent = file.getParentFile(); // File(HSE)

boolean exists = file.exists();
boolean isDir = file.isDirectory();
boolean isFile = file.isFile();
long length = file.length();

File[] subfiles = parent.listFiles();

boolean success = file.createNewFile();
```

Класс File (2)

```
1  File file = new File("HSE/sample.txt");
2  String path = file.getPath(); // HSE/sample.txt
3  String name = file.getName(); // sample.txt
4  String absPath = file.getAbsolutePath(); // /home/vasya/HSE/sample.txt
5  File parent = file.getParentFile(); // File(HSE)
6
7  boolean exists = file.exists();
8  boolean isDir = file.isDirectory();
9  boolean isFile = file.isFile();
10 long length = file.length();
11
12 File[] subfiles = parent.listFiles();
13
14 boolean success = file.createNewFile();
15 boolean success = parent.mkdir();
16
17
18
```

Класс File (2)

```
1  File file = new File("HSE/sample.txt");
2  String path = file.getPath(); // HSE/sample.txt
3  String name = file.getName(); // sample.txt
4  String absPath = file.getAbsolutePath(); // /home/vasya/HSE/sample.txt
5  File parent = file.getParentFile(); // File(HSE)
6
7  boolean exists = file.exists();
8  boolean isDir = file.isDirectory();
9  boolean isFile = file.isFile();
10 long length = file.length();
11
12 File[] subfiles = parent.listFiles();
13
14 boolean success = file.createNewFile();
15 boolean success = parent.mkdir();
16 boolean success = parent.mkdirs();
17
18
```

Класс File (2)

```
1  File file = new File("HSE/sample.txt");
2  String path = file.getPath(); // HSE/sample.txt
3  String name = file.getName(); // sample.txt
4  String absPath = file.getAbsolutePath(); // /home/vasya/HSE/sample.txt
5  File parent = file.getParentFile(); // File(HSE)
6
7  boolean exists = file.exists();
8  boolean isDir = file.isDirectory();
9  boolean isFile = file.isFile();
10 long length = file.length();
11
12 File[] subfiles = parent.listFiles();
13
14 boolean success = file.createNewFile();
15 boolean success = parent.mkdir();
16 boolean success = parent.mkdirs();
17 boolean success = file.delete();
18
```

Класс File (2)

```
1  File file = new File("HSE/sample.txt");
2  String path = file.getPath(); // HSE/sample.txt
3  String name = file.getName(); // sample.txt
4  String absPath = file.getAbsolutePath(); // /home/vasya/HSE/sample.txt
5  File parent = file.getParentFile(); // File(HSE)
6
7  boolean exists = file.exists();
8  boolean isDir = file.isDirectory();
9  boolean isFile = file.isFile();
10 long length = file.length();
11
12 File[] subfiles = parent.listFiles();
13
14 boolean success = file.createNewFile();
15 boolean success = parent.mkdir();
16 boolean success = parent.mkdirs();
17 boolean success = file.delete();
18 boolean success = file.renameTo(new File("HSE/newname.txt"));
```

Java NIO (1)

`File` — класс довольно устаревший и плохо спроектированный.

Для большего удобства рекомендуется использовать сущности из пакета `java.nio` (NIO — New IO).

- `java.nio.file.Path`
- `java.nio.file.Paths`
- `java.nio.file.Files`

Java NIO (2)

```
1  Path path = Paths.get("HSE", "Algo", "task1.py");
2
3
4
5
6
7
8
9
10
11
12
13
14
```

Операций ОГРОМНОЕ количество: документация Path

Java NIO (2)

```
1  Path path = Paths.get("HSE", "Algo", "task1.py");
2
3  File fromPath = path.toFile();
4
5
6
7
8
9
10
11
12
13
14
```

Операций ОГРОМНОЕ количество: документация Path

Java NIO (2)

```
1  Path path = Paths.get("HSE", "Algo", "task1.py");
2
3  File fromPath = path.toFile();
4  Path fromFile = fromPath.toPath();
5
6
7
8
9
10
11
12
13
14
```

Операций ОГРОМНОЕ количество: документация Path

Java NIO (2)

```
1  Path path = Paths.get("HSE", "Algo", "task1.py");
2
3  File fromPath = path.toFile();
4  Path fromFile = fromPath.toPath();
5
6  Path parent = path.getParent(); // HSE/Algo
7
8
9
10
11
12
13
14
```

Операций ОГРОМНОЕ количество: документация Path

Java NIO (2)

```
1  Path path = Paths.get("HSE", "Algo", "task1.py");
2
3  File fromPath = path.toFile();
4  Path fromFile = fromPath.toPath();
5
6  Path parent = path.getParent(); // HSE/Algo
7  String name = path.getFileName(); // task1.py
8
9
10
11
12
13
14
```

Операций ОГРОМНОЕ количество: документация Path

Java NIO (2)

```
1  Path path = Paths.get("HSE", "Algo", "task1.py");
2
3  File fromPath = path.toFile();
4  Path fromFile = fromPath.toPath();
5
6  Path parent = path.getParent(); // HSE/Algo
7  String name = path.getFileName(); // task1.py
8  Path absolute = path.toAbsolutePath(); // /home/petya/HSE/algo/task1.py
9
10
11
12
13
14
```

Операций ОГРОМНОЕ количество: документация Path

Java NIO (2)

```java
Path path = Paths.get("HSE", "Algo", "task1.py");

File fromPath = path.toFile();
Path fromFile = fromPath.toPath();

Path parent = path.getParent(); // HSE/Algo
String name = path.getFileName(); // task1.py
Path absolute = path.toAbsolutePath(); // /home/petya/HSE/algo/task1.py

boolean startsWith = absolute.startsWith("/home");
```

Операций ОГРОМНОЕ количество: документация Path

Java NIO (2)

```java
Path path = Paths.get("HSE", "Algo", "task1.py");

File fromPath = path.toFile();
Path fromFile = fromPath.toPath();

Path parent = path.getParent(); // HSE/Algo
String name = path.getFileName(); // task1.py
Path absolute = path.toAbsolutePath(); // /home/petya/HSE/algo/task1.py

boolean startsWith = absolute.startsWith("/home");
Path task2 = path.resolveSibling("task2.py"); // HSE/Algo/task2.py
```

Операций ОГРОМНОЕ количество: документация Path

Java NIO (2)

```java
Path path = Paths.get("HSE", "Algo", "task1.py");

File fromPath = path.toFile();
Path fromFile = fromPath.toPath();

Path parent = path.getParent(); // HSE/Algo
String name = path.getFileName(); // task1.py
Path absolute = path.toAbsolutePath(); // /home/petya/HSE/algo/task1.py

boolean startsWith = absolute.startsWith("/home");
Path task2 = path.resolveSibling("task2.py"); // HSE/Algo/task2.py

long size = Files.size(path);

```

Операций ОГРОМНОЕ количество: документация Path

Java NIO (2)

```java
Path path = Paths.get("HSE", "Algo", "task1.py");

File fromPath = path.toFile();
Path fromFile = fromPath.toPath();

Path parent = path.getParent(); // HSE/Algo
String name = path.getFileName(); // task1.py
Path absolute = path.toAbsolutePath(); // /home/petya/HSE/algo/task1.py

boolean startsWith = absolute.startsWith("/home");
Path task2 = path.resolveSibling("task2.py"); // HSE/Algo/task2.py

long size = Files.size(path);
Files.copy(path, Paths.get("HSE/Algo/task1-copy.py"));
```

Операций ОГРОМНОЕ количество: документация Path

Паттерн "Декоратор" (1)

Рассмотрим интерфейс с простой реализацией:

```
1  public interface Printer {
2      void println(String s);
3  }
4
5  public class ConsolePrinter implements Printer {
6      @Override
7      public void println(String s) {
8          System.out.println(s);
9      }
10 }
```

Что, если мы хотим:

- Шифровать выводимую строку?
- Переводить строку в нижний регистр перед выводом?
- Сжимать выводимую строку?
- Использовать всё перечисленное в любых комбинациях?

Паттерн "Декоратор" (2)

Паттерн проектирования "Декоратор" позволяет **динамически** расширять функционал у экземпляров классов.

Этап 1: при помощи агрегации реализуем нужные расширения:

```
1  public class EncryptingPrinter implements Printer {
2      private final Printer nested;
3      public EncryptingPrinter(Printer nested) {
4          this.nested = nested;
5      }
6      @Override
7      public void println(String s) {
8          nested.println(encrypt(s));
9      }
10     private void encrypt(String s) { ... }
11 }
```

Паттерн "Декоратор" (2)

Паттерн проектирования "Декоратор" позволяет **динамически** расширять функционал у экземпляров классов.

Этап 1: при помощи агрегации реализуем нужные расширения:

```
1  public class LowercasePrinter implements Printer {
2      private final Printer nested;
3      public LowercasePrinter(Printer nested) {
4          this.nested = nested;
5      }
6      @Override
7      public void println(String s) {
8          nested.println(s.toLowerCase());
9      }
10 }
```

Паттерн "Декоратор" (2)

Паттерн проектирования "Декоратор" позволяет **динамически** расширять функционал у экземпляров классов.

Этап 1: при помощи агрегации реализуем нужные расширения:

```
1  public class CompressingPrinter implements Printer {
2      private final Printer nested;
3      public CompressingPrinter(Printer nested) {
4          this.nested = nested;
5      }
6      @Override
7      public void println(String s) {
8          nested.println(compress(s));
9      }
10     private void compress(String s) { ... }
11 }
```

Паттерн "Декоратор" (3)

Этап 2: создаём экземпляр базового класса и "декорируем" его нужным функционалом:

```
1  Printer basePrinter = new ConsolePrinter();
2  basePrinter.println("Sample text"); // Просто выводим
3
4  Printer example1 = new CompressingPrinter(basePrinter);
5  example1.println("Sample text"); // сжимаем -> выводим
6
7  Printer example2 = new EncryptingPrinter(
8      new CompressingPrinter(basePrinter)
9  );
10 example2.println("Sample text"); // шифруем -> сжимаем -> выводим
11
12 Printer example3 = new LowercasePrinter(
13     new CompressingPrinter(basePrinter)
14 );
15 example3.println("Sample text"); // меняем регистр -> сжимаем -> выводим
```

Потоки ввода-вывода (1)

Потоки ввода-вывода (**не путать с потоками данных!**) представляют Java-объекты, позволяющие считывать (input stream) или записывать (output stream) последовательность байт.

Базовые классы (абстрактные):

- `java.io.InputStream`
- `java.io.OutputStream`

Также есть классы (тоже абстрактные) для работы с символами и строками:

- `java.io.Reader`
- `java.io.Writer`

У всех этих классов есть множество декораторов.

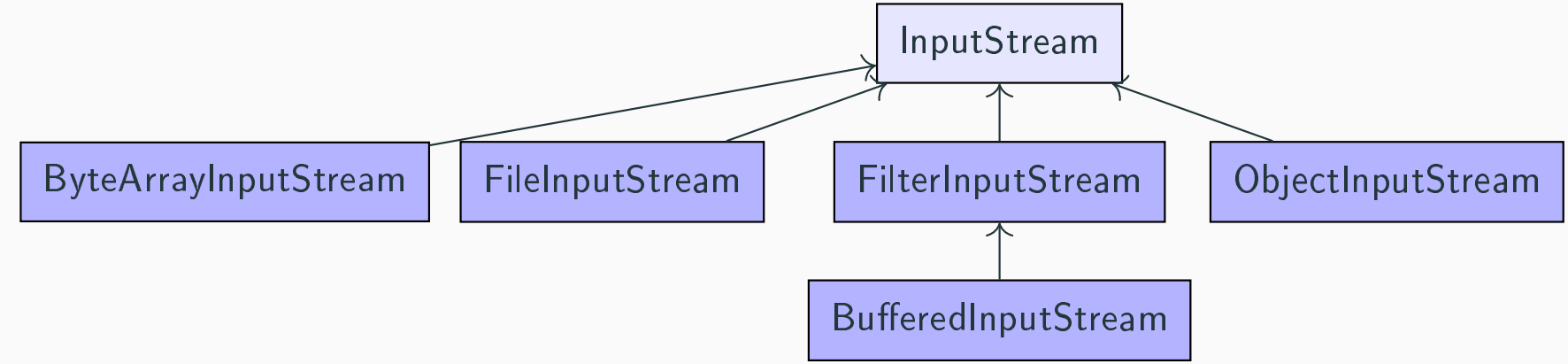
Потоки ввода-вывода (2)

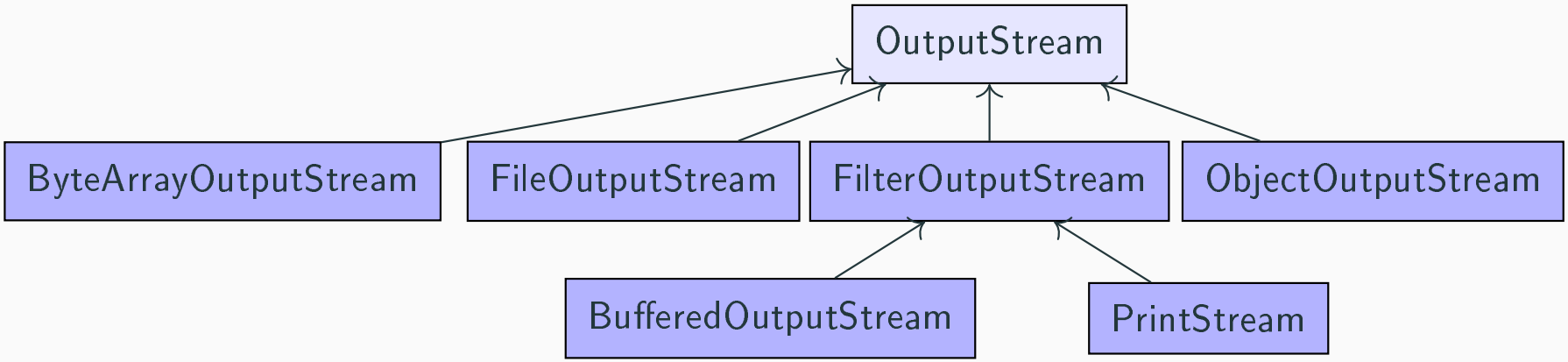
Потоки ввода-вывода — это абстракция над источниками и приёмниками данных.

В качестве них могут выступать:

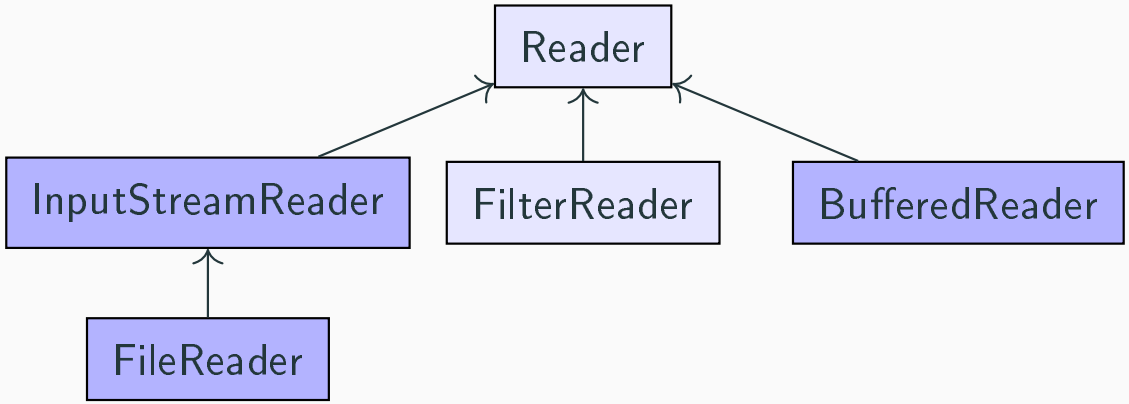
- Файлы.
- Массивы байт или символов.
- Сетевые соединения (для взаимодействия разных программ).
- и т.д.

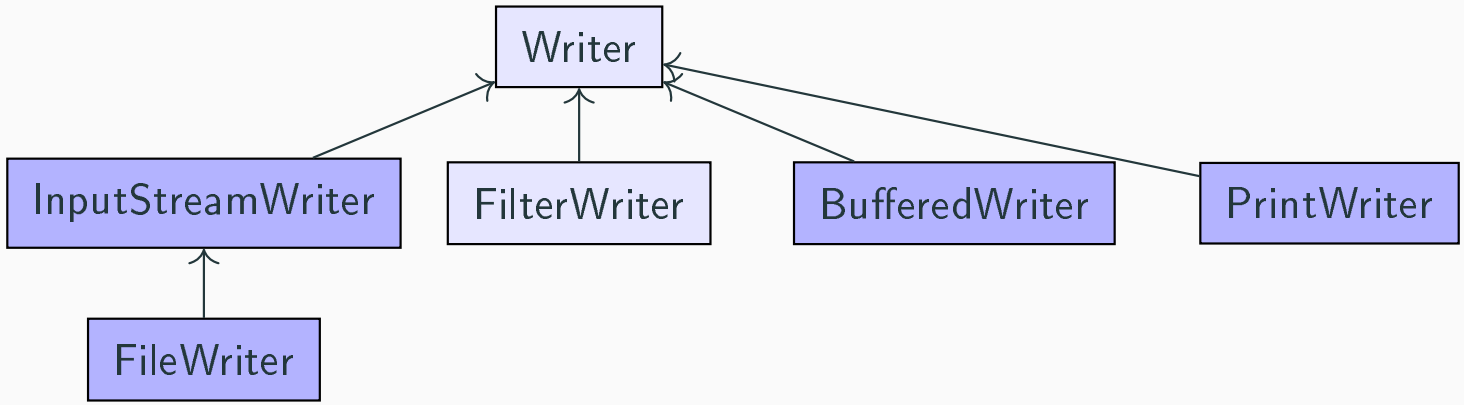
InputStream и OutputStream





Reader и Writer





Стандартные потоки ввода-вывода

Стандартные потоки ввода-вывода тоже используют это API:

- `System.in` — `InputStream`
- `System.out` и `System.err` — `PrintWriter`

Пример записи в поток

Если мы хотим напечатать строку в файл, то можно сделать это так:

```
1 FileOutputStream fos = new FileOutputStream("example.txt");
2 BufferedOutputStream bos = new BufferedOutputStream(fos);
3 PrintStream printStream = new PrintStream(bos);
4 printStream.println("Sample text");
```

У `PrintStream` есть перегрузка конструктора, позволяющая печатать текст в файл в стандартной кодировке:

```
1 PrintStream printStream = new PrintStream("example.txt");
2 printStream.println("Sample text");
```

Замечание: этот код не до конца корректен! Дальше узнаем, почему.

Пример чтения из потока

Аналогично и со чтением:

```
1 FileInputStream fis = new FileInputStream("example.txt");
2 InputStreamReader isr = new InputStreamReader(fis);
3 BufferedReader reader = new BufferedReader(isr);
4 String line = reader.readLine();
```

Можно и покороче:

```
1 FileReader fileReader = new FileReader("example.txt");
2 BufferedReader reader = new BufferedReader(fileReader);
3 String line = reader.readLine();
```

Замечание: этот код не до конца корректен! Дальше узнаем, почему.

Интерфейс Closeable

Все операции чтения/записи в файл подразумевают открытие файла.

Но если что-то было открыто — надо это обратно закрыть.

Все рассмотренные потоки ввода-вывода реализуют интерфейс `Closeable`. Этот интерфейс имеет метод `close`, закрывающий поток.

Закрытие потоков ввода-вывода (1)

Закроем поток после использования:

```
1 PrintStream printStream = new PrintStream("example.txt");
2 printStream.println("Sample text");
3 printStream.close();
```

Проблема: промежуточные операции (`println`) могут бросить исключение, и поток останется открытым.

Закрытие потоков ввода-вывода (2)

Обернём это в `try-finally`:

```
1 PrintStream printStream = null;
2 try {
3     printStream = new PrintStream("example.txt");
4     printStream.println("Sample text");
5 } finally {
6     if (printStream != null) {
7         printStream.close();
8     }
9 }
```

Проблема: а если у потока несколько декораторов?

Закрытие потоков ввода-вывода (3)

В Java есть конструкция try-with-resources (TWR), автоматически закрывающая указанные потоки:

```
1 try (PrintStream printStream = new PrintStream("example.txt")) {
2     printStream.println("Sample text");
3 } // Поток автоматически закроется после выхода из блока
```

Side note: указанная конструкция работает для всех типов, реализующих интерфейс `AutoCloseable`.

Закрытие потоков ввода-вывода (4)

Если потоков несколько — они указываются через точку с запятой:

```
try (
        FileReader fileReader = new FileReader("example.txt");
        BufferedReader reader = new BufferedReader(fileReader)
) {
    String line = reader.readLine();
} // Поток автоматически закроется после выхода из блока
```

Исключение IOException

Почти все операции с файловой системой могут завершиться ошибкой.

В Java такие ошибки описываются исключениями типа `IOException` (и наследниками).

Данное исключение — checked, поэтому его необходимо обрабатывать или указывать через `throws`:

```
1 try (PrintStream printStream = new PrintStream("example.txt")) {
2     printStream.println("Sample text");
3 } catch (IOException ioException) {
4     // Просто сообщаем об ошибке в консоль (в реальности - хитрее)
5     System.out.println("Error writing to file 'example.txt': " +
6         ioException.getMessage()
7     );
8     ioException.printStackTrace(System.out);
9 }
```