**Стандартные интерфейсы: InputStream, OutputStream**

[Java Core](https://javarush.ru/quests/QUEST_JAVA_CORE)

[Уровень 3](https://javarush.ru/quests/lectures?quest=QUEST_JAVA_CORE&level=3), Лекция 8

— Привет, Амиго! Сейчас я тебе расскажу про два интерфейса – **InputStream** и **OutputStream**. Объявлены они как абстрактные классы, но если начать разбираться, то можно увидеть, что**по своей сути – это интерфейсы.** Почти все их методы абстрактные, кроме нескольких незначительных методов. Очень похожи на нашего «телохранителя», которого мы рассматривали.

Это очень интересные интерфейсы. Пока что я специально буду называть их интерфейсы, чтобы ты понял, зачем они нужны. А потом мы поговорим, почему же их все-таки сделали абстрактными классами.

— Хорошо. Так что это за интерфейсы?

— Сейчас расскажу.

Есть такая интересная вещь в Java как «**поток**». **Поток** – это очень простая сущность. И его простота есть залог очень мощного механизма обмена данными. Потоки бывают двух видов: **поток для чтения и поток для записи.**

В поток для записи, как ты уже, наверное, догадался, **можно записывать данные**. Для этого у него есть метод **write**(). Из потока для чтения **можно данные читать**. Для этого у него есть метод **read**().

**InputStream** – это интерфейс потока чтения, описывающий такую способность: «из меня можно читать байты».

А **OutputStream**– это, соответственно, интерфейс потока записи, описывающий способность: «в меня можно записывать байты».

— И это все?

— Фактически да. Но все дело в том, что в Java есть очень много классов, которые умеют работать с интерфейсами **InputStream** и **OutputStream**. Например, ты хочешь прочитать файл с диска и вывести его содержимое на экран. Нет ничего проще.

Для того, чтобы прочитать данные из файла на диске, есть специальный класс **FileInputStream**, который реализует интерфейс **InputStream**. Хочешь записать прочитанные данные в другой файл? Для этого есть класс **FileOutputStream**, который реализует интерфейс **OutputStream**. Вот как выглядит код копирования [данных одного] файла в другой.

Код

public static void main(String[] args) throws IOException

{

InputStream inStream = new FileInputStream("c:/source.txt");

OutputStream outStream = new FileOutputStream("c:/result.txt");

while (inStream.available() > 0)

{

int data = inStream.read(); //читаем один байт из потока для чтения

outStream.write(data); //записываем прочитанный байт в другой поток.

}

inStream.close(); //закрываем потоки

outStream.close();

}

Представь, что мы написали класс, и добавили ему способности **InputStream** и **OutputStream**.

Если мы корректно реализовали поддержку этих интерфейсов, то объекты нашего класса теперь можно сохранить в файл на диске. Просто вычитав их содержимое через метод **read**. Или загрузить из файла, создав объект и записав в него содержимое файла через метод **write**.

— А можно пример?

— Можно.

|  |  |
| --- | --- |
| **Код** | **Описание** |
| class MyClass  {  private List<Integer> list =  new ArrayList<>(Arrays.asList(111, 222, 333));  } | Для простоты представим, что наш класс содержит в себе один объект – ArrayList типа Integer. Наполним список тестовыми данными. |

Теперь добавим в него методы read и write

|  |  |
| --- | --- |
| **Код** | **Описание** |
| class MyClass  {  private List<Integer> list =  new ArrayList<>(Arrays.asList(111, 222, 333));  public void write(int data)  {  list.add(data);  }  public int read()  {  int first = list.get(0);  list.remove(0);  return first;  }  public int available()  {  return list.size();  }  } | Теперь у нас в классе реализован метод **read**, который позволяет последовательно вычитать все содержимое нашего списка **list**.  И метод **write**, который позволяет записывать в наш list значения. |

Это, конечно, не реализация интерфейсов InputStream и OutputStream, но очень похоже.

— Да, это понятно. А как все-таки сохранить содержимое такого объекта в файл?

— Давай я напишу тебе пример:

Запись объекта MyClass в файл

public static void main(String[] args) throws IOException

{

MyClass myObject = new MyClass();

OutputStream outStream = new FileOutputStream ("c:/my-object-data.txt");

while (myObject.available() > 0)

{

int data = myObject.read(); //читаем один int из потока для чтения

outStream.write(data); //записываем прочитанный int в другой поток.

}

outStream.close();

}

3

Задача

Java Core,  3 уровень,  8 лекция

Набираем код Ӏ Java Core: 3 уровень, 8 лекция

Java Core: 3 уровень, 8 лекция. Иногда думать не надо, строчить надо! Как ни парадоксально звучит, порой пальцы «запоминают» лучше, чем сознание. Вот почему во время обучения в секретном центре JavaRush вы иногда встречаете задания на набор кода. Набирая код, вы привыкаете к синтаксису и зарабатываете немного материи. А ещё — боретесь с ленью.

Чтение объекта MyClass из файла

public static void main(String[] args) throws IOException

{

InputStream inStream = new FileInputStream("c:/my-object-data.txt");

MyClass myObject = new MyClass();

while (inStream.available() > 0)

{

int data = inStream.read(); //читаем один int из потока для чтения

myObject.write(data); //записываем прочитанный int в другой поток.

}

inStream.close(); //закрываем потоки

}

— Ух ты! Действительно, очень похоже на работу с InputStream/OutputStream. Потоки – это крутая вещь!

— А то!