**Вопросы по теме InputStream:**

1. Расскажите про класс InputStream и его подклассы.
2. Для чего нужен класс BufferedInputStream?
3. Расскажите про класс FileInputStream.
4. Для чего нужен классObjectInputStream?

**InputStream:**

**InputStream** – абстрактный класс, описывающий поток ввода, который работает с байтами.

Основные методы класса (их имеют все классы наследники):

*int available()* – возвращает количество байт, которое можно прочитать из входного потока;

*void* *close()* – закрывает поток и освобождает ресурсы, связанные с ним;

*int read()* – считывает текущий байт из входного потока и возвращает его или *-1* если данных в потоке больше нет;

*int read​(byte[] b)* – считывает в буфер количество байт равное *b.length*, возвращает количество прочитанных байт или *-1* если данных в потоке больше нет;

*int read​(byte[] b, int off, int len)* – считывает в буфер *b* количество байт *len* начиная с позиции *off*, первый прочитанный байт сохранится в *b[off]*, возвращает количество прочитанных байт или *-1* если данных в потоке больше нет;

*long skip​(long n)* – пропускает первые *n* байт из входного потока, возвращает фактическое количество пропущенных байт;

*void mark​(int readlimit)* – отмечает текущую позицию в этом входном потоке, *readlimit* - максимальное количество байтов, которое может быть прочитано до того, как позиция метки станет недействительной;

*void reset()* – перемещает этот поток в положение последнего вызова метода mark();

*boolean markSupported()* – проверяет, поддерживает ли входной поток методы mark и reset.

**Наследники класса InputStream:**

**ByteArrayInputStream** – класс, использующий в качестве источника данных массив байтов. Имеет конструкторы:

*ByteArrayInputStream(byte[] buf)*

*ByteArrayInputStream(byte[] buf, int offset, int length)*

*buf* – массив байтов, *offset* – с какого байта будем считывать, *length* – количество считываемых байт.

Пример использования:

byte[] array = new byte[]{1, 2, 3, 4, 5};  
ByteArrayInputStream bais = new ByteArrayInputStream(array))  
int data;  
while ((data = bais.read()) != -1) {  
 System.*out*.println(data);  
}

Особенности:

Можно не вызывать метод close(), в этом классе он не имеет эффекта.

**FileInputStream** – класс, использующий в качестве источника данных файл.

Имеет конструкторы:

*FileInputStream(File file)*

*FileInputStream(FileDescriptor fdObj)*

*FileInputStream(String name)*

*fdObj* – экземпляр класса FileDescriptor (файловый дескриптор позволяет получить доступ к файлу даже если этот файл был переименован, удален, закрыт к нему доступ).

Пример использования:

try (FileInputStream fis = new FileInputStream("D:\\temp.txt")) {  
 int n;  
 while ((n = fis.read()) != -1) {  
 System.*out*.print((char) n);  
 }  
}

Особенности:

Предназначен для считывания байт, для считывания символов лучше использовать другие классы.

**FilterInputStream** – класс, предназначенный для фильтрации, модификации или предоставления дополнительных функций для входного потока. Работает почти так же, как класс InputStream. Он переопределяет все методы InputStream, а эти переопределенные методы просто передают все запросы вложенному входному потоку.

protected volatile InputStream in;

protected FilterInputStream(InputStream in) {  
 this.in = in;  
}

*in* – входной поток для фильтрации.

public int read() throws IOException {  
 return in.read();  
}

**DataInputStream** – класс, считывающий из входного потока данные примитивных типов.

Имеет конструкторы:

*DataInputStream(InputStream in)*

Пример использования:

byte[] array = new byte[]{1, 2, 3, 4, 5};  
ByteArrayInputStream bais = new ByteArrayInputStream(array);  
try(DataInputStream dis = new DataInputStream(bais)){  
 byte data;  
 while (dis.available() > 0) {  
 data = dis.readByte();  
 System.*out*.println(data);  
 }  
}

Особенности:

Наследуется от FilterInputStream. Для чтения каждого примитивного типа существует свой метод (readInt(), readChar() и т.д.). Метод readLine() является устаревшим и не рекомендуется к использования, т. к. неверно преобразует байты в символы. Для чтения строк рекомендуется использовать BufferedReader.readLine().

**BufferedInputStream** - накапливает вводимые данные в специальном буфере без постоянного обращения к устройству ввода.

Имеет конструкторы:

*BufferedInputStream(InputStream inputStream)*

*BufferedInputStream(InputStream inputStream, int bufSize)*

*bufSize* — размер буфера в байтах.

Пример использования:

try (BufferedInputStream bis = new BufferedInputStream(new FileInputStream("D:\\temp.txt"))) {  
 int n;  
 while ((n = bis.read()) != -1) {  
 System.*out*.print((char) n);  
 }  
}

Особенности:

Наследуется от FilterInputStream. Предназначен для оптимизации и ускорения процесса считывания информации за счет ее передачи порциями, равными размеру буфера. Размер буфера по умолчанию — 8192 байт.

**ObjectInputStream** — класс, предназначенный для чтения сериализованных данных.

Имеет конструкторы:

*ObjectInputStream()* конструктор для классов, переопределяющих ObjectInputStream

*ObjectInputStream(InputStream in)*

Пример использования:

try(ObjectInputStream ois = new ObjectInputStream(new FileInputStream("D:\\temp.txt")))  
{  
 MyObject myObject = (MyObject) ois.readObject();  
 System.*out*.println(myObject.myMethod());  
}

Особенности:

Для считывания примитивных типов данных используются методы readInt(), readChar() и т. д., для чтение объекта используется метод readObject(). Метод readLine() является устаревшим и не рекомендуется к использования, т. к. неверно преобразует байты в символы. Для чтения строк рекомендуется использовать BufferedReader.readLine().

**PipedInputStream** – класс, предназначенный для связи отдельных потоков друг с другом внутри одной JVM. Обычно связывается PipedInputStream и PipedOutputStream и используются при многопоточном программировании. Каждый раз, когда данные записываются в PipedOutputStream, они автоматически появляются в PipedInputStream.

**PushbackInputStream** – класс, который дает возможность «отодвинуть» или «непрочитать» байты, сохраняя вытесненные байты во внутреннем буфере с помощью метода unread(). Наследуется от FilterInputStream.

**SequenceInputStream** – класс, позволяющий объединить несколько потоков ввода. Данные считываются сначала полностью с первого потока, потом со второго и т.д., пока не будет считана информация со всех объединенных потоков.

**CheckedInputStream** – класс, позволяющий использовать контрольную сумму для проверки целостности входных данных.

**StringBufferInputStream** – устарел, неправильно преобразует символы в байты. Рекомендованная замена – StringReader.