Reader/Writer

Java Core
9 уровень, 2 лекция

ОТКРЫТА

— Привет, Амиго! Сегодня Элли тебе рассказала про паттерн адаптер.

Большинство классов, относящихся к потокам ввода-вывода, реализовано в виде адаптера. Они преобразуют либо равнозначные интерфейсы, либо соединяют их по принципу от простого к сложному.

— А **InputStreamReader** и **BufferedReader** – тоже адаптеры? По крайней мере, они очень похожи на них по стилю использования: объект после создания передается в конструктор другого объекта.

— Да, **InputStreamReader** преобразует интерфейс **InputStream** к интерфейсу **Reader**. **BufferedReader** не адаптер в чистом виде, т.к. разработчики Java решили не выделять его методы в отдельный интерфейс. Но по духу, он стоит очень близко к ним.

Вместо того, чтобы писать 100500 различных классов, разработчики Java написали два десятка адаптеров и разрешили их соединять друг с другом, как программисту захочется.

Такой подход очень удобен. Программист всегда может написать свой класс и/или адаптер, реализовать в нем стандартный интерфейс и включить его в собранную им цепочку объектов-адаптеров.

— Так вот как оно, оказывается, все устроено. Вместо больших сложных классов – цепочки простых объектов и адаптеры. А ты просто создаешь их и соединяешь в правильном порядке!

— И реализовываешь то, чего не хватает.

— Да, я понимаю.

— Но вообще-то я хотел сегодня рассказать тебе про Reader и Writer. Это два абстрактных класса, которые очень похожи на классы InputStream и OutputStream. Но в отличие от них, эти два класса работают с символами. Они читают символы и записывают символы. Они очень удобны при работе с текстовой информацией. Давай посмотрим, какие методы у них есть:

Методы класса Reader	Что метод делает
`int read(char[] cbuf);`	— метод сразу читает много символов в буфер (массив символов), пока буфер не заполнится или не закончатся символы там, откуда он их читает. Метод возвращает количество реально прочитанных символов (оно может быть меньше длины массива)
`int read();`	— метод читает один символ и возвращает его как результат. Результат расширяется до int, для красоты. Если доступных символов нет, метод вернет «-1».
`boolean ready();`	— метод возвращает true если есть еще непрочитанные символы для методов read
`void close();`	— метод «закрывает» поток, вызывается после окончания работы с потоком. Объект выполняет служебные операции, связанные с закрытием файла на диске и т.д. Из потока больше нельзя читать данные.

— Оказывается, благодаря методу read(char[] cbuf) из Reader'а можно читать символы целыми блоками, а не по одному символу. Так и быстрее и удобнее.

— Да. А теперь посмотрим, какие методы есть у Writer:

Метод	Что метод делает
`void write(int c);`	— метод записывает один символ. Тип int сужается до char, лишняя часть просто отбрасывается.
`void write(char[] cbuff);`	— метод записывает массив символов.

```
1  void write(String s);
```

— метод записывает строку. Она просто преобразовывается в массив символов и вызывается второй метод.

```
1  void flush();
```

— если есть данные, которые хранятся где-то внутри и еще не записаны, то они записываются.

```
1  void close();
```

— метод «закрывает» поток – вызывается после окончания работы с потоком.
Объект выполняет служебные операции, связанные с закрытием файла на диске и т.д.В поток больше нельзя писать данные, flush при этом вызывается автоматически.

Важно понять, что **Reader** и **Writer** – это абстрактные классы. Они ничего не делают и практически не содержат кода. Все их методы должны будут реализовываться в классах, которые будут унаследованы от них. Их же задача – стандартизировать механизм взаимодействия между классами. Разработчикам не нужно изобретать свои стандарты для взаимодействия друг с другом. Гораздо удобнее всем поддерживать несколько базовых стандартов. Тогда классы, написанные разными программистами, смогут легко взаимодействовать не только с классами, написанными разработчиками Java, но и с классами других программистов.

Стандарты – великая сила.

— Согласен. Поддержка общих стандартов – благо для всех.

‹ Предыдущая

×19 ›

Программистами не рождаются