



# Lesson 17

24.02.2022

```
public class Ex1 {  
    public static void main(String[] args) {  
        List<Character> list = new ArrayList<>();  
        list.add( 0 , 'E' );  
        list.add( 'X' );  
        list.add( 1 , 'P' );  
        list.add( 3 , 'O' );  
        if (list.contains( 'O' )) {  
            list.remove( 'O' );  
        }  
        for ( char ch : list) {  
            System.out.print(ch);  
        }  
    }  
}
```

```
public class Ex2 {  
    private static void m(int x) {  
        System.out.println("INT VERSION");  
    }  
  
    private static void m(char x) {  
        System.out.println("CHAR VERSION");  
    }  
  
    public static void main(String[] args) {  
        int i = '5';  
        m(i);  
        m('5');  
    }  
}
```

```
public class Ex3 {  
    public static void main(String[] args) {  
        int x = 7;  
        boolean res = x++ == 7 && ++x == 9 || x++ == 9;  
        System.out.println("x = " + x);  
        System.out.println("res = " + res);  
    }  
}
```



```
public class Ex4 {  
    public static void main(String[] args) {  
        Demo demo = new Demo();  
        demo.show();  
    }  
}  
class Demo {  
    public void show() {  
        LinkedList<String> list = new LinkedList<>();  
        System.out.println(list.getFirst());  
    }  
}
```



## Из чего же состоит объект?

Прежде чем определять объем потребляемой памяти, следует разобраться, что же JVM хранит для каждого объекта:

1. Заголовок объекта;
2. Память для примитивных типов;
3. Память для ссылочных типов;
4. Смещение/выравнивание — по сути, это несколько неиспользуемых байт, что размещаются после данных самого объекта. Это сделано для того, чтобы адрес в памяти всегда был кратным машинному слову, для ускорения чтения из памяти + уменьшения количества бит для указателя на объект + предположительно для уменьшения фрагментации памяти. Стоит также отметить, что в java размер любого объекта кратен 8 байтам!



## Структура заголовка объекта

Каждый экземпляр класса содержит заголовок. Каждый заголовок для большинства JVM(Hotspot, openJVM) состоит из двух машинных слов. Если речь идет о 32-х разрядной системе, то размер заголовка — 8 байт, если речь о 64-х разрядной системе, то соответственно — 16 байт. Каждый заголовок может содержать следующую информацию:

1. Маркировочное слово (mark word)

2. Hash Code — каждый объект имеет хеш код. По умолчанию результат вызова метода `Object.hashCode()` вернет адрес объекта в памяти, тем не менее некоторые сборщики мусора могут перемещать объекты в памяти, но хеш код всегда остается одним и тем же, так как место в заголовке объекта как раз может быть использовано для хранения оригинального значения хеш кода.

3. Garbage Collection Information — каждый java объект содержит информацию нужную для системы управления памятью. Зачастую это один или два бита-флага, но также это может быть, например, некая комбинация битов для хранения количества ссылок на объект.

4. Type Information Block Pointer — содержит информацию о типе объекта. Этот блок включает информацию о таблице виртуальных методов, указатель на объект, который представляет тип и указатели на некоторые дополнительные структуры, для более эффективных вызовов интерфейсов и динамической проверки типов.

5. Lock — каждый объект содержит информацию о состоянии блокировки. Это может быть указатель на объект блокировки или прямое представление блокировки.



Для ввода данных используется класс ***Scanner*** из библиотеки пакетов Java.

В классе есть методы для чтения очередного символа заданного типа со стандартного потока ввода, а также для проверки существования такого символа.

- `next ();`
- `nextLine ();`
- `nextInt ();`
- `nextDouble (); ....`





**Потоки в Java** определяются в качестве последовательности данных. Существует два типа потоков:

**InPutStream** – поток ввода используется для считывания данных с источника.

**OutPutStream** – поток вывода используется для записи данных по месту назначения.



## Байтовый поток

Потоки байтов в Java используются для осуществления ввода и вывода 8-битных байтов. Не смотря на множество классов, связанных с потоками байтов, наиболее распространено использование следующих классов: `FileInputStream` и `FileOutputStream`. Ниже рассмотрен пример, иллюстрирующий использование данных двух классов для копирования из одного файла в другой.



## Символьные потоки

Потоки байтов в Java позволяют произвести ввод и вывод 8-битных байтов, в то время как потоки символов используются для ввода и вывода 16-битного юникода. Не смотря на множество классов, связанных с потоками символов, наиболее распространено использование следующих классов: `FileReader` и `FileWriter`. Не смотря на тот факт, что внутренний `FileReader` использует `FileInputStream`, и `FileWriter` использует `FileOutputStream`, основное различие состоит в том, что `FileReader` производит считывание двух байтов в конкретный момент времени, в то время как `FileWriter` производит запись двух байтов за то же время.

Мы можем переформулировать представленный выше пример, в котором два данных класса используются для копирования файла ввода (с символами юникода) в файл вывода.

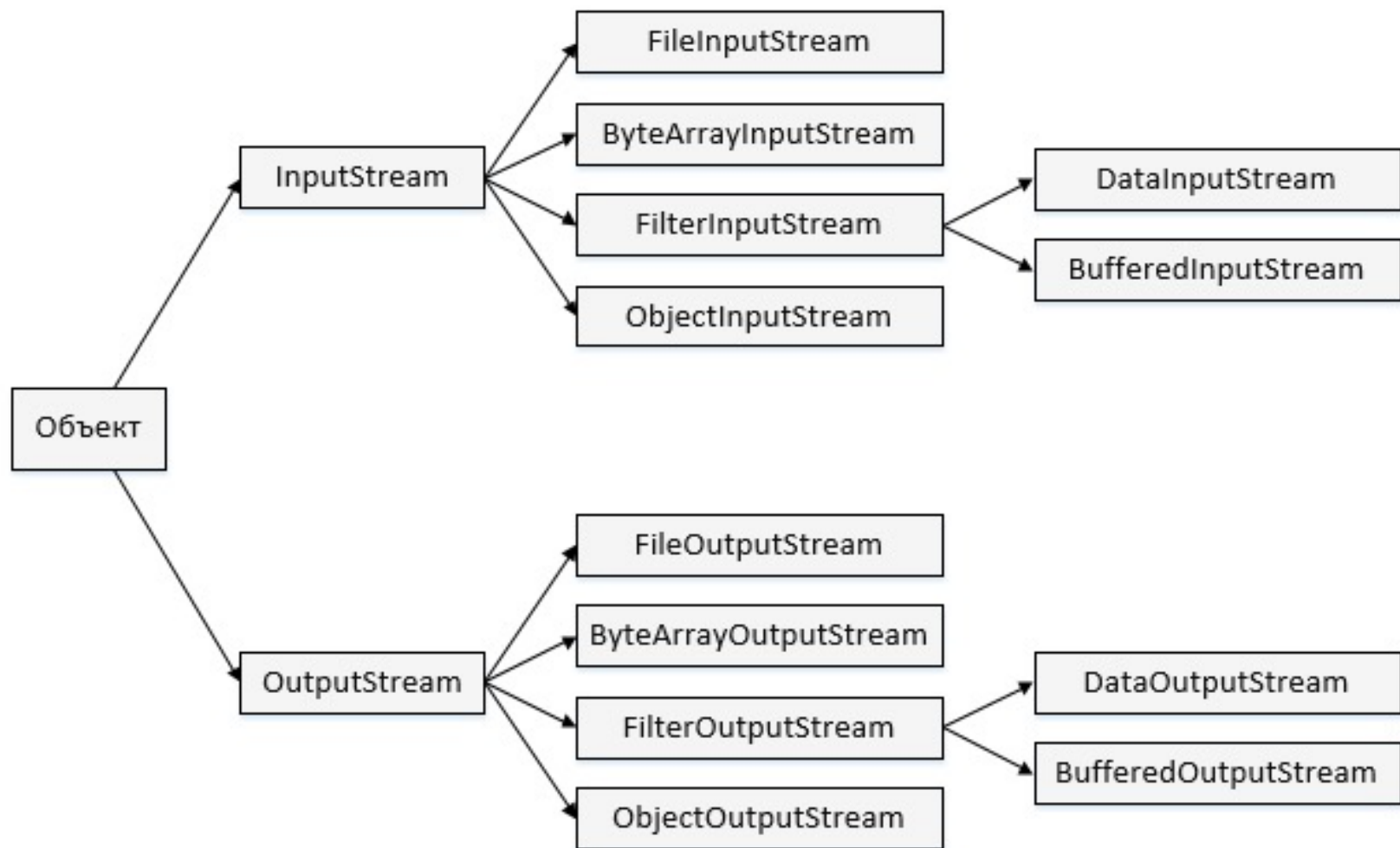
## Стандартные потоки

Все языки программирования обеспечивают поддержку стандартного ввода/вывода, где программа пользователя может произвести ввод посредством клавиатуры и осуществить вывод на экран компьютера. Если вы знакомы с языками программирования C либо C++, вам должны быть известны три стандартных устройства STDIN, STDOUT и STDERR. Аналогичным образом, Java предоставляет следующие три стандартных потока:

Стандартный ввод – используется для перевода данных в программу пользователя, клавиатура обычно используется в качестве стандартного потока ввода, представленного в виде [System.in](#).

Стандартный вывод – производится для вывода данных, полученных в программе пользователя, и обычно экран компьютера используется в качестве стандартного потока вывода, представленного в виде [System.out](#).

Стандартная ошибка – используется для вывода данных об ошибке, полученной в программе пользователя, чаще всего экран компьютера служит в качестве стандартного потока сообщений об ошибках, представленного в виде [System.err](#).



## Поток FileInputStream – чтение из файла

```
InputStream a = new FileInputStream("D:/myprogramm/java/test");
```

```
File a = new File("D:/myprogramm/java/test");  
InputStream a = new FileInputStream(a);
```

№	Метод и описание
1	<b>public void close() throws IOException{}</b> Данный метод в Java закрывает выходной файловый поток. Освобождает какие-либо системные ресурсы, связанные с файлом. Выдает IOException.
2	<b>protected void finalize()throws IOException {}</b> Данный метод выполняет очистку соединения с файлом. Позволяет удостовериться в вызове закрытого метода данного выходного файлового потока при отсутствии каких-либо ссылок на данный поток. Выдает IOException.
3	<b>public int read(int r)throws IOException{}</b> Данный метод осуществляет в Java считывание заданных байтов данных из InputStream. Возврат данных типа int. Возврат следующего байта данных, в конце файла будет произведен возврат к -1.
4	<b>public int read(byte[] r) throws IOException{}</b> Данный метод производит считывание байтов r.length из входного потока в массив. Возврат общего числа считанных байтов. В конце файла будет произведен возврат к -1.
5	<b>public int available() throws IOException{}</b> Выдает число байтов, которые могут быть считаны из входного файлового потока. Возврат данных типа int.



## Поток `FileOutputStream` – создание и запись файла

```
OutputStream a = new FileOutputStream("D:/myprogramm/java/test")
```

```
File a = new File("D:/myprogramm/java/test");  
OutputStream a = new FileOutputStream(a);
```



№	Метод и описание
1	<p><b>public void close() throws IOException{</b></p> <p>Данный метод в Java закрывает выходной файловый поток. Освобождает какие-либо системные ресурсы, связанные с файлом. Выдает IOException.</p>
2	<p><b>protected void finalize()throws IOException {</b></p> <p>Данный метод выполняет очистку соединения с файлом. Позволяет удостовериться в вызове закрытого метода данного выходного файлового потока при отсутствии каких-либо ссылок на данный поток. Выдает IOException.</p>
3	<p><b>public void write(int w)throws IOException{</b></p> <p>Данный метод осуществляет запись заданного байта в выходной поток.</p>
4	<p><b>public void write(byte[] w)</b></p> <p>Запись байтов w.length из указанного массива байтов в OutputStream.</p>

## Каталоги в Java

В Java каталог представлен Файлом, который может содержать список других файлов и каталогов. Используя объект `File`, вы можете создать каталог, прокрутить список файлов, представленных в каталоге. Для получения более детальных сведений, ознакомьтесь с перечнем всех методов, которые могут быть вызваны из объекта `File`, будучи связанными с каталогами.


### Создание каталогов

Существуют два служебных метода `File`, которые могут быть использованы для создания каталогов:

Метод `mkdir()` позволяет создать папку в Java, возвращая значение `true` при успехе операции, и `false` в случае сбоя. Сбой свидетельствует о том, что путь указанный в объекте `File` уже существует, либо что каталог не может быть создан в связи с тем, что полный путь еще не существует.

Метод `mkdirs()` создает каталог и все вышестоящие каталоги.

В следующем примере представлено создание папки `"/java/proglang/newdir"`:



```
public String getName()
```

Возвращает имя файла или каталога, по указанному абстрактному имени пути.

```
public String getParent()
```

Возвращает строковый путь родителя абстрактного пути, или null, если путь не указывает родительский каталог.

```
public String getParent()
```

Возвращает строковый путь родителя абстрактного пути, или null, если путь не указывает родительский каталог.

```
public boolean canRead()
```

Проверяет, может ли приложение прочитать файл, по указанному абстрактному имени пути. Возвращает true тогда и только тогда, когда файл, указанный в абстрактном пути, существует и может быть прочитан приложением; в противном случае false.

```
public boolean canWrite()
```

Проверяет, может ли приложение изменять файл, по указанному абстрактному имени пути. Возвращает true тогда и только тогда, когда файловая система фактически содержит файл, по указанному абстрактному имени пути, и приложению разрешено записывать в файл; в противном случае false.

### **public boolean exists()**

Проверяет, существует ли файл или каталог, по указанному абстрактному имени пути. Возвращает true тогда и только тогда, когда существует файл или каталог, по указанному абстрактному имени пути; в противном случае false.

### **public boolean isDirectory()**

Проверяет, является ли файл, по указанному абстрактному имени пути, каталогом. Возвращает true тогда и только тогда, когда файл, обозначенный этим абстрактным именем, существует и является каталогом; в противном случае false.

### **public boolean isFile()**

Проверяет, является ли файл, по указанному абстрактному имени пути, нормальным файлом. Файл является нормальным, если он не является каталогом и, кроме того, удовлетворяет другим системным критериям. Любой файл без каталога, созданный приложением Java, гарантированно является нормальным файлом. Возвращает true тогда и только тогда, когда файл, обозначенный этим абстрактным пустым именем, существует и является нормальным файлом; в противном случае false.

### **public long lastModified()**

Возвращает время последнего изменения файла, по указанному абстрактному имени пути. Возвращает длинное значение, представляющее время последнего изменения файла, измеренное в миллисекундах с эпохи (00:00:00 GMT, 1 января 1970 г.) или 0L, если файл не существует или возникает ошибка ввода-вывода.



### **public boolean createNewFile() throws IOException**

Атомарно создает новый пустой файл, названный этим абстрактным именем пути, тогда и только тогда, когда файл с этим именем еще не существует. Возвращает true, если названный файл не существует и был успешно создан; false, если именованный файл уже существует.

### **public boolean delete()**

Удаляет файл или каталог, по указанному абстрактному имени пути. Если это имя пути обозначает каталог, каталог должен быть пустым, чтобы его можно было удалить. Возвращает true тогда и только тогда, когда файл или каталог успешно удалены; в противном случае false.

### **public void deleteOnExit()**

Просит, чтобы файл или каталог, обозначенные данным абстрактным пустым именем, были удалены при завершении работы виртуальной машины.

### **public String[] list()**

Возвращает массив строк, называющий файлы и каталоги в каталоге, обозначаемом этим абстрактным именем пути.

### **public File[] listFiles(FileFilter filter)**

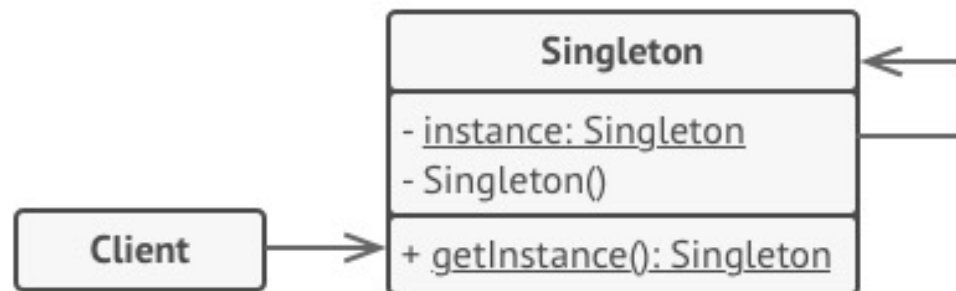
Возвращает массив абстрактных путей, обозначающих файлы и каталоги в каталоге, обозначаемом этим абстрактным пустым именем пути, которое удовлетворяет указанному фильтру.



## *Singleton*

**Гарантирует наличие единственного экземпляра класса.** Чаще всего это полезно для доступа к какому-то общему ресурсу, например, базе данных.

**Предоставляет глобальную точку доступа.** Это не просто глобальная переменная, через которую можно достигаться к определённому объекту. Глобальные переменные не защищены от записи, поэтому любой код может подменять их значения без вашего ведома.



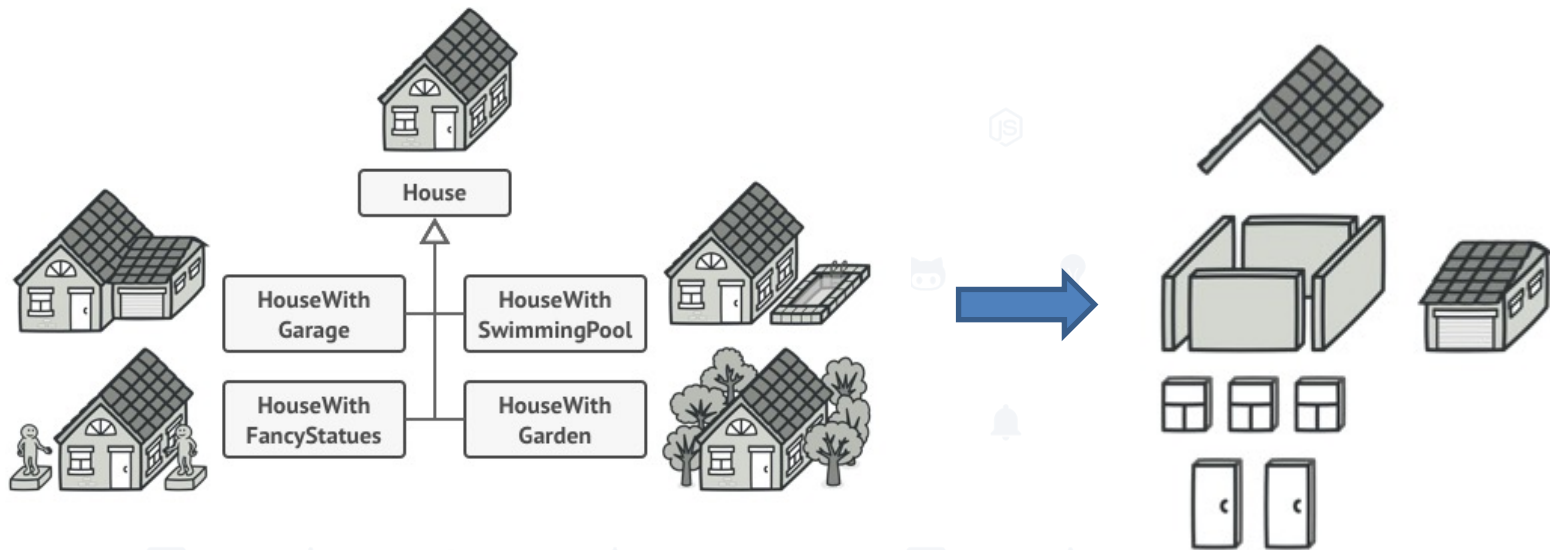
**1** **Одиночка** определяет статический метод `getInstance`, который возвращает единственный экземпляр своего класса.

Конструктор одиночки должен быть скрыт от клиентов. Вызов метода `getInstance` должен стать единственным способом получить объект этого класса.

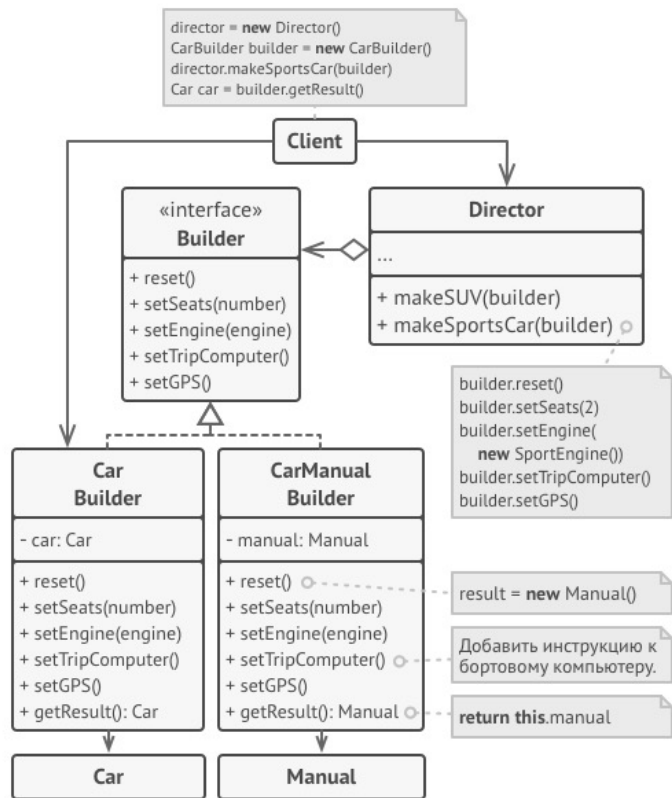
```
if (instance == null) {  
    // Внимание, если вы пишете  
    // многопоточный код, то здесь  
    // нужно синхронизировать потоки.  
    instance = new Singleton()  
}  
return instance
```

## Строитель

**Строитель** — это порождающий паттерн проектирования, который позволяет создавать сложные объекты пошагово. Строитель даёт возможность использовать один и тот же код строительства для получения разных представлений объектов.







**Интерфейс строителя** объявляет шаги конструирования продуктов, общие для всех видов строителей.

**Конкретные строители** реализуют строительные шаги, каждый по-своему. Конкретные строители могут производить разнородные объекты, не имеющие общего интерфейса.

**Продукт** — создаваемый объект. Продукты, сделанные разными строителями, не обязаны иметь общий интерфейс.

**Директор** определяет порядок вызова строительных шагов для производства той или иной конфигурации продуктов.