Java

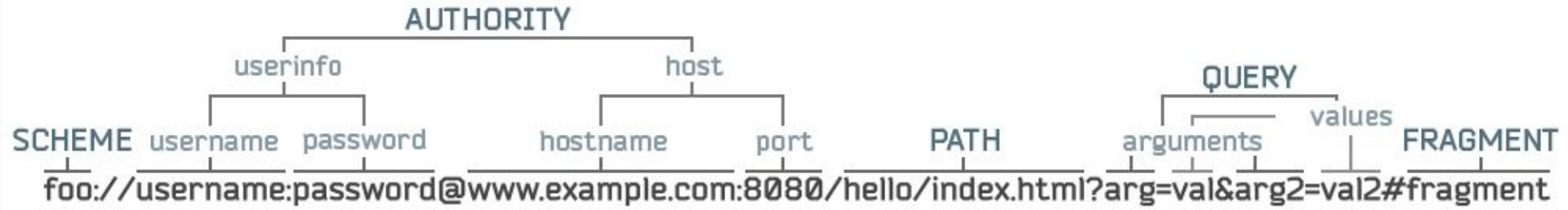
Работа с файловой системой

File

- Находится в пакете java.io (новые объекты в java.nio)
- File file = new File("filePath");
  - new File(parent, child);
- Не открывает файл, только создаёт объект File
- Также может принимать URI

Uniform Resource Identifier (URI)

- Унифицированный идентификатор ресурса
- URI это либо URL либо URN. В контексте файлов это всегда URL



File

- getName() - имя файла (последний сегмент пути)
- isFile() - проверяет что по указанному пути не директория и он существует
- isDirectory() - проверяет что по указанному пути сущесвтующая директория
- exists() - проверяет наличие файла или директории
- getParent() / getParentFile() - получение родительской директории
- isAbsolute() - проверка что путь абсолютный

Как получить список файлов в папке

```
if (file.isDirectory()) {
    File[] files = file.listFiles()
}
```

Работа с папками

- mkdir() - создаёт новую пустую папку
- mkdirs() - создаёт новую пустую папку и все родительские
- delete() - удаляет файл или пустую папку

Специальные обозначения

- . - текущая директория
- .. - родительская директория

Примеры путей

- /home/user/git/../file.txt
- user/../foo/bar/file.txt
- ./relative/path/file.txt
- /home/./user/./file.txt

Сравнение файлов

- Пути
  - /home/./user/./file.txt
  - /home/user/file.txt
- getCanonicalPath()

Java IO

- IO был представлен в Java 1.0
- Работает с InputSream/OutputSream, которые на низком уровне оперируют байтами, по байту за раз
- *Reader и *Writer - абстракция, которая повышает удобство
- Блокируют выполнение потока (Thread) до завершения операции

Java NIO

- NIO был добавлен в 1.4 и доработан в 1.7
- Non-blocking Input/Output или New IO
- NIO работает с буфером, который читает и пишет частями
- Операции не блокируют поток
- Упрощает работу с кодировками

Java NIO

- Path
- Paths
- Files

Path

- Упрощает работу с путями
- resolve(..)
- parent()
- relativize(..)
- normalize()

Paths

- get(pathOrURI)
- Путь должен соответствовать URI

Files

- createDirectories(..)
- createTempFile()
- createTempDirectory()
- delete() / deleteIfExists()
- copy()
- isDirectory()
- isRegularFile()
- exists() / notExists()
- walkFileTree()

Design Patterns (Шаблоны проектирования)

- Помогают общаться на одном языке
- Помогают быстрее находить подходящее стандартное решение
- Помогают другим разработчикам читать чужой код

Паттерн Visitor

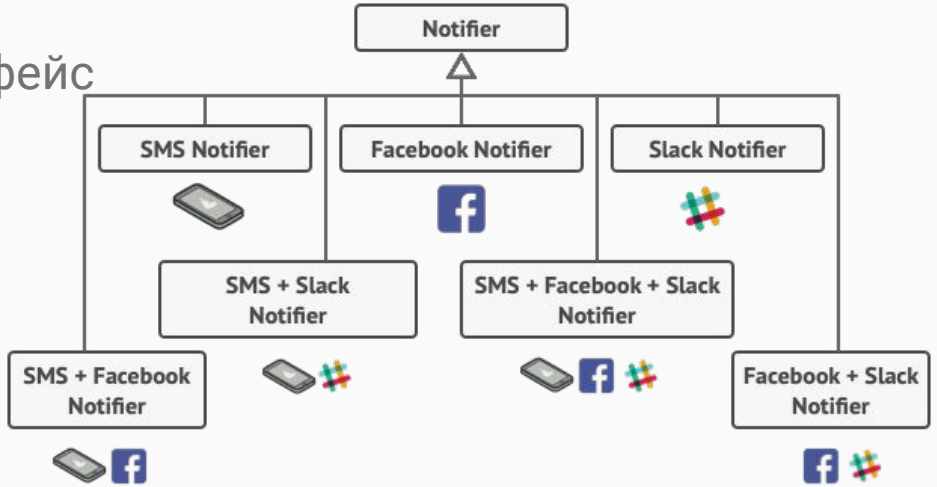
- Поведенческий паттерн (шаблон)
- Позволяет отделить алгоритм от объектов, над которыми он работает
- Предназначен для структурированных данных

Паттерн Visitor

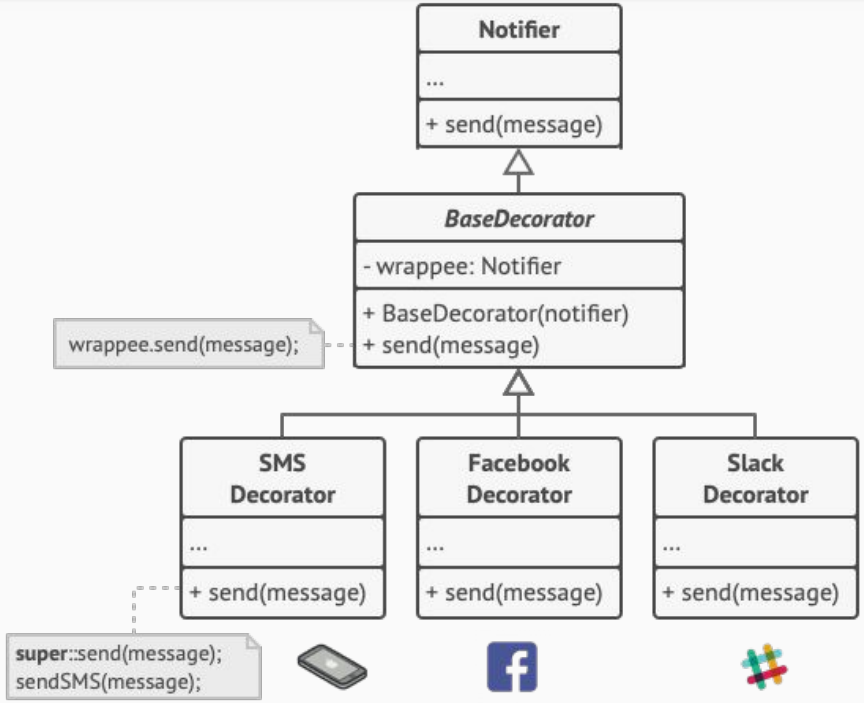
1. Объявить интерфейс с набором методов-”визитёров”
2. Определить интерфейс элемента
3. Реализовать методы-”визитёры” для каждого типа
4. Для каждой операции создаётся своя реализация

Декоратор (Decorator/Wrapper)

- Структурный паттерн
- Позволяет добавлять новые поведения методом создания объекта-обёртки
- Обёртка принимает общий интерфейс



Декоратор


Notifier
...
+ send(message)
BaseDecorator
- wrappee: Notifier
+ BaseDecorator(notifier)
+ send(message)
wrappee.send(message);
SMS Decorator
...
+ send(message)
Facebook Decorator
...
+ send(message)
Slack Decorator
...
+ send(message)
super::send(message);
sendSMS(message);

Git

- В гит как правило попадают только нужные изменения
- ДЗ можно оформлять как в виде репозитория, так и в виде ветки
- Нужно убедиться что в Pull Request добавлены все нужные файлы разом
- Если ДЗ реализуется в отдельном репозитории то можно создавать PR на отдельные пункты