Задачи

1. Целью данной работы было проверить применимость уже имеющихся навыков в новой области программирования.
2. Получить опыт использования конструкции enum в контексте Java
3. Более подробно разобраться в концепции интерфейсов, лямбда-функций и ссылок на функции.

Концепция

В практике встроенной разработки на языке С мне встречались задачи типа:

1. Генерация ответа на определенный ID запроса.
2. Создание конечных автоматов и их обработчиков.

Для решения такого рода задач хорошо подходит связка указателей на функции + перечисления. Например, создается перечисление, содержащее id поддерживаемых обработчиков:

```
typedef enum {
    ACTION0 = 0,
    ACTION1 = 1,
    ACTION2 = 2,
    ACTION_LIMIT = 3
} actions_t;
```

Затем создается тип данных – указатель на функцию-обработчик с его сигнатурой:

```
typedef void (*fPtr)(uint32_t arg1, uint32_t arg2);
```

В отдельных модулях прописываются реализации обработчиков:

mod0.c

```
void hndlr_action0(uint32_t arg1, uint32_t arg2) {
    // blah-blah
    return;
}
```

mod1.c

```
void hndlr_action1(uint32_t arg1, uint32_t arg2) {
    // blah-blah
    return;
}
```

mod2.c

```
void hndlr_action2(uint32_t arg1, uint32_t arg2) {
    // blah-blah
    return;
}
```

И, наконец, создается таблица указателей на функции, с помощью которой достаточно лаконично решаются задачи 1 и 2.

solution.c

```
fPtr handlersTable[ACTION_LIMIT] = {
    [ACTION0] = hndlr_action0,
    [ACTION1] = hndlr_action1,
    [ACTION2] = hndlr_action2,
};

void handlerWrapper(action_t id, uint32_t data1, uint32_t data2) {
    handlersTable[id]( data1, data2);
}
```

Соответственно, мне было интересно проверить, насколько эта концепция применима в языке Java.

Проблема

В финальной версии проекта(ветка main) можно было заметить, что в классах Stylish и Plain используется примерно один и тот же код в методе генератора отчета, в котором отличается лишь формат строк для разных статусов записи. Я поставил себе цель разделить этот метод на две части:

- скелет (пакет builder), который использует в качестве входных данных как сам список с диффами, так и соответствующий данному стилю обработчики с помощью описанной выше концепции;
- обработчики(пакет formatters) – методы, результатом работы которых является строка для отдельной записи из списка диффов, представленная в требуемом формате (Stylish/Plain/Json).

Решение

Для реализации было сделано следующее:

1. Создан функциональный интерфейс RecordMaker.java(пакет builder).
2. Создано перечисление RecordStatus.java, соответствующее статусу записи: UNCHANGED, CHANGED, ADDED, DELETED.
3. Внутри классов, соответствующих определенным форматам вывода (Stylish, Plain, пакет formatters) были созданы обработчики статусов записей: buildUnchanged, buildChanged, buildAdded, buildDeleted. Там же был создан массив типа RecordMaker размером RecordStatus.LIMIT, и каждый элемент массива с индексом, обозначающим статус, был проинициализирован соответствующим обработчиком (например: builders[RecordStatus.UNCHANGED.ordinal()] = this::buildUnchanged).
4. Далее, в метод build класса CommonBuilder (пакет builders), наряду со списком диффов передается массив обработчиков builders типа RecordMaker для соответствующего стиля. Внутри этого метода перебирается весь список диффов. Для каждого его элемента проверяется статус (UNCHANGED, CHANGED, ADDED, DELETED). Далее, на его основе из массива обработчиков builders выбирается нужный. И уже он генерирует отчет в виде строки для очередного элемента списка.

Таким образом, проблема использования практически одинакового кода в форматерах для Stylish и Plain была решена. Но это же породило следующую проблему: для форматера Json используется не рукописный код, а отдельная библиотека. Для ее решения интерфейс Style был также объявлен функциональным, а в классы форматеров Stylish, Plain и Json(папка formatters), реализующих этот интерфейс, был добавлен метод apply, вызывающий тот или

иной билдер. И для наглядности добавлено перечисление RecordStyle.java, в котором соответствующим сущностям STYLISH, PLAIN и JSON был присвоен соответствующий метод форматирования.

В идеале можно было бы прийти к двумерному массиву такого рода обработчиков, но это заняло бы еще какое-то время. А я, кажется, слишком увлекся этой задачей и изрядно отстал от графика.)

И в качестве иллюстрации ограничения диапазона в класс Parser вместо строковых констант с именами поддерживаемых расширений было добавлено перечисление Filetypes. Каждый элемент этого класса генерирует парсер определенного типа, что вроде бы соответствует концепции «фабрики», указанной в описании к одному из шагов проекта. Но тут я могу ошибаться.

Выводы

«+»:
- за счет использования enum вместо строковых констант исключается возможность использования каких-то недопустимых/посторонних значений;
- также мне понравилось использование метода valueOf для проверки вхождения значения в enum, это делает код лаконичнее.
- был увеличен уровень атомарности кода за сет более тщательного разбиения на модули.

«-»:
- атомарность стала и минусом, так как увеличение модульности кода приводит к усложнению понимания его структуры и, следовательно, дальнейшей поддержки и сопровождения;
- в модуле Plain в предыдущей версии статус UNCHANGED никак не обрабатывался. В новой версии из-за обязательной реализации метода buildUnchanged пришлось возвращать в ней пустую строку и, соответственно, использовать «костыль» для проверки наличия пустых строк в отчете (CommonBuilder.java, 18).

То есть с моей точки зрения минусы все же перевешивают плюсы, поэтому предложенный мной вариант реализации не является оптимальным.

Вопросы

1 Возможно, в силу малого опыта я упустил некоторые моменты, которые сделали бы код более понятным?
2. Еще хотел бы спросить про такой момент: enum представляет собой, фактически, полноценный класс со встроенными ограничениями. Уместно ли его использование только лишь как список констант, то есть без всей полноты его функционала?
3. Имеет ли смысл использование связки enum+(создание массива ссылок на функции) в приложениях, написанных на Java и какие у такого подхода есть преимущества и недостатки?
4. В целом, есть ли какие-то советы по выбору и планированию архитектуры приложения в Java? Как показал мне самому мой небольшой эксперимент, этот момент играют большую роль в этом ЯП.