Сравнение ModelMapper и MapStruct и Orika на примерах

1. Мэппинг нескольких полей на единственное

- Domain :

```
class Note {
  private String title;
  private String brief;
  private String text;
  private String type;
}
```

- ViewObject

```
class SimpleNoteView {
    private String description;
  private String type;
}
```

@Test twoInOne()

Задача: Мы хотим маппить из Note поля title и brief на description

MapStruct: Мэппинг удалось решить с помощью expression а.

```
@Mapper(componentModel = "spring")
public interface NoteMapper {
   @Mapping(target = "description",
         expression = "java( String.valueOf(note.getTitle() + ' ' + note.getBrief()) )")
   NoteView noteToNoteView(Note note);
```

//Фрагмент сгенерированного кода:

```
   NoteView noteView = new NoteView();
   noteView.setType( note.getType() );
   noteView.setDescription( String.valueOf(note.getTitle() + ' ' + note.getBrief()) );
```

Замечание: В перспективе возможно для этого можно попробовать использовать decorator. К сожалению в данный момент он не поддерживается для Spring конфигурации

ModelMapper: В Model Mapper есть возможность делать мэппинг для полей с помощью PropertyMap. Но возникла проблема из за proxies types

```
   public class NoteMap extends PropertyMap<Note, NoteView> {
         protected void configure() {
            String description = source.getTitle() + ' ' + source.getBrief();
            map().setDescription(description);
```

```
        }
    }
```

```
//Будет брошено исключение "Cannot map final type java.lang.StringBuilder."
Удалось решить только конвертером.
```

```
       private Converter<Note, NoteView> noteConverter = new AbstractConverter<Note, NoteVie
             protected NoteView convert(Note note) {
                NoteView noteView = new NoteView();
                noteView.setDescription(note.getTitle() + ' ' + note.getBrief());
                return noteView;
             }
          };
      modelMapper.addConverter(noteConverter);
```

2. Мэппинг с условием

Задача: Допустим необходимо смаппить на commentary title если type == common Добавим в NoteView поле commentary

ModelMapper: В ModelMapper есть Condition. Таким образом будет выглядеть условие

```
    Condition<Note, NoteView> typeIsCommon = new Condition<Note, NoteView>() {
         @Override
         public boolean applies(MappingContext<Note, NoteView> context) {
            System.out.println("test");
            Note note = (Note)context.getParent().getSource();
            boolean flag = note.getType().equals("common");
            return flag;
         }
      };
```

Добавим в PropertyMap

```
        public class NoteMap extends PropertyMap<Note, NoteView> {
             protected void configure() {
                when(typeIsCommon).map().setCommentary(source.getTitle());
             }
          }
```

и в ModelMapper

```
        modelMapper.addMappings(new NoteMap());
```

MapStruct: Здесь приведу как вариант Статический метод + expression. Дальше будет другой пример по аналогии с которым можно вместо статического метода использовать custom mapper.
  статический метод:

```
         public static String getCommentaryByType(Note note){
                if(note.getType().equals("common")){
```

```
                return note.getTitle();
            }
            return "";
        };
```

И Expression :

```
            @Mapping(target = "commentary",
                    expression = "java( NoteConditions.getCommentaryByType(note) )")
              NoteView noteToNoteVieww(Note note);
```

3. Дженерики

Задача: Допустим есть AbstractNote с методом возвращающим абстрактный объект AbstractComment А нам нужно сделать маппинг его наследника с переопределённым методом.

```
 @Mappings({
         @Mapping(target = "commentary",
               expression = "java( ((VelvetNote)note).getCommentary().toString()
)"),
         @Mapping(target = "advanceTitle",
               expression = "java( ((VelvetNote)note).getAdvanceTitle() )")
    })
```

ModelMapper: Подхватил свойства наследника без дополнительных настроек

4. Мэппинг сущностей тип полей которых может меняться от внешнего условия

- Domain :

```
public class Control {
   private Long id;
   private String name;
   private Details details;

public class Details {
   private String name;
   private String description;
```

- ViewObject

```
public class ControlView {
   private String name;
   private String status;
   private DetailsView detailsView;
```

А DetailsView типом в зависимости от внешнего условия

```
public class AdminDetailsView extends DetailsView {
   private String description;или


public class DetailsView {
   private String name;
```

Удалось сделать решение без использования дополнительного конвертера только на Map Struct.

Основной Mapper с методом:

```
@Mapping(target = "detailsView", expression = "java(abstractDetailsMapper.fromDetailsView
   ControlView controlToControlView(Control control, Boolean isEmployee);
```

Дополнительный :

```
@Mapper(componentModel = "spring")
public interface DetailsMapper {
   DetailsView detailsToDetailsView(Details details);
   AdminDetailsView adminDetailsToDetailsView(Details details, Boolean isEmployee);
}
```

isEmployee и есть параметр внешнее условие и custom mapper который делает выбор по условию

```
@Component
public class AbstractDetailsMapper {
@Autowired
private DetailsMapper detailsMapper;
public DetailsView fromDetailsView(Details details) {
   return detailsMapper.detailsToDetailsView(details);
}
public DetailsView fromDetailsView(Details details, Boolean isEmployee) {
   if (isEmployee) {
      return detailsMapper.adminDetailsToDetailsView(details, isEmployee);
   } else {
       return detailsMapper.detailsToDetailsView(details);
   }
 }
```

5. О проверка маппинга

ModelMapper: В ModelMapper можно проверить что все поля были смаппле-ны с помощью вызова метода validate у mapper а. Если мы в PropertyMap закомментарим

```
map().setDescription(source.getText());
```

То на проверки будет брошено исключение

```
1) Unmapped destination properties found in TypeMap[Note –> NoteView]:
```

MapStruct: я не нашёл способов чтобы он бросал исключение при возникновении ошибок тем не менее, но он может выводить информацию в лог о полях который не были замапплены.