

# 18. Учебный проект: большие переменные (Часть 1)

Рабочая ветка `module6-task1`

## Задача

В этом задании мы добавим в наши компоненты работу с данными и сделаем так, чтобы при изменении информации внутри компонента, изменялись данные, из которых эти компоненты были созданы.

Сперва разберёмся с кодом, который скопился у нас в `BoardController` и отвечает за смену задачи на форму редактирования задачи. Для этого создадим `TaskController`:

1. В нем нужно описать конструктор и метод `render`
2. Конструктор должен принимать `container` — элемент, в который контроллер будет всё отрисовывать.
3. Метод `render` должен принимать данные одной задачи. Также в него должен переехать код, который отвечает за отрисовку задачи, ее замену на форму редактирования и наоборот, а также установка связанных с этим обработчиков событий.

## Настроим частичный датабиндинг

Реализуем обработку кликов на кнопках «Favorites», «Archive» у карточки задачи. Обработчики должны изменять данные задачи — добавлять или удалять из избранного и архивировать или разархивировать задачу соответственно, и на основе изменённых данных перерисовывать компонент.

Для этого:

1. В компоненте задачи добавьте методы для установки обработчиков клика для каждой кнопки.
2. В конструкторе `TaskController` добавьте новый аргумент — функцию `onDataChange`. Эта функция должна вызываться в обработчике клика и получать на вход старые и новые данные (задачу и измененную задачу).

**Обратите внимание**, пока что мы занимаемся обновлением только признаков задачи, поэтому при сохранении у нас по-прежнему форма просто заменяется на карточку задачи, а удаление не работает вовсе. С этим мы разберёмся в следующих заданиях.

3. В `BoardController` опишите метод `_onDataChange` с точно таким же интерфейсом, как функция `onDataChange`. Задача метода — обновить моки и вызывать метод `render` у конкретного экземпляра `TaskController` с обновлёнными данными.

4. Чтобы всё заработало, передайте метод `_onDataChange` в `TaskController` при создании его экземпляра.

## Добавим интерактивности

Форма редактирования довольно сложный интерактивный компонент. Но это поведение — не часть бизнес-логики приложения. Это бизнес-логика самого компонента. Поэтому для реализации этой логики введем понятие `SmartComponent` — компонент, который может себя перерисовывать.

1. Создайте абстрактный класс `AbstractSmartComponent`, унаследовав его от `AbstractComponent`, с двумя методами:
  - абстрактный метод `recoveryListeners`, его нужно будет реализовать в наследнике. Его задача — восстанавливать обработчики событий после перерисовки;
  - обычный метод `rerender`, его задачи:
    - удалить старый DOM-элемент компонента;
    - создать новый DOM-элемент;

- поместить новый элемент вместо старого;
  - восстановить обработчики событий, вызвав `recoveryListeners`.
2. Унаследуйте компонент формы редактирования от `AbstractSmartComponent` и объявите в компоненте метод `recoveryListeners`, пока пустым.
3. Теперь нужно реализовать перерисовку формы редактирования после взаимодействия с пользователем:
- показ или скрывание поля ввода даты по клику на «Date», а так же смену «Yes» и «No»;
  - показ или скрывание полей для выбора дней повторения по клику на «Repeat», а так же смену «Yes» и «No»;
  - выбор цвета задачи.

**Обратите внимание**, что кнопку «Save» необходимо блокировать, если поля показаны, а дата или дни повторения не выбраны.

4. При перерисовке компонента все обработчики событий будут утеряны, поэтому их нужно навесить заново в методе `recoveryListeners`.

## Отообразим только одну форму

Мы научились обновлять данные и создавать интерактивные компоненты. Осталось запретить открывать несколько форм редактирования одновременно. Мы реализуем это простым способом: в `BoardController` мы «прикажем» всем экземплярам `TaskController` вернуться в исходное состояние, когда пользователь открывает форму редактирования.

1. Добавьте метод `setDefaultView` в `TaskController` для отображения задачи вместо формы редактирования.
2. Добавьте в конструктор `TaskController` добавьте новый аргумент — функцию `onViewChange` и вызывайте её перед тем, как сменить задачу на форму редактирования.
3. В `BoardController` опишите метод `_onViewChange`, где вызовите у всех экземпляров `TaskController` метод `setDefaultView`.
4. В `BoardController` при создании экземпляров `TaskController` передайте и метод `_onViewChange`.

## 20. Личный проект: большие переменные (Часть 1)

### Задача

В этом задании мы добавим в наши компоненты работу с данными и сделаем так, чтобы при изменении информации внутри компонента, изменялись данные, из которых эти компоненты были созданы.

## Киноман

Сперва разберёмся с кодом, который скопился у нас в `PageController` и отвечает за показ попапа с подробной информацией о фильме и его закрытие. Для этого создадим `MovieController`:

1. В нем нужно описать конструктор и метод `render`
2. Конструктор должен принимать `container` — элемент, в который контроллер будет всё отрисовывать.
3. Метод `render` должен принимать данные одного фильма. Также в него должен переехать код, который отвечает за отрисовку карточки фильма, показ попапа с подробной информацией о фильме и его закрытие, а также установка связанных с этим обработчиков событий.

## Настроим частичный датабиндинг

Реализуем обработку кликов на кнопках «Add to watchlist», «Already watched», «Add to favorites» у карточки фильма и у попапа подробной информации о фильме. Обработчики должны изменять данные фильма — добавлять или удалять из избранного и списка к просмотру, а также помечать его просмотренным или наоборот, и на основе изменённых данных перерисовывать компоненты.

Для этого:

1. В компоненте фильма и попапа добавьте методы для установки обработчиков клика для каждой кнопки.
2. В конструкторе `MovieController` добавьте новый аргумент — функцию `onDataChange`. Эта функция должна вызываться в обработчике клика и получать на вход старые и новые данные (фильм и измененный фильм).

**Обратите внимание**, что комментарии — это отдельная структура, и функция `onDataChange` занимается обновлением только информации о фильме. С комментариями мы разберёмся в следующих заданиях.

3. В `PageController` опишите метод `_onDataChange` с точно таким же интерфейсом, как функция `onDataChange`. Задача метода — обновить моки и вызывать метод `render` у конкретного экземпляра `MovieController` с обновлёнными данными.
4. Чтобы всё заработало, передайте метод `_onDataChange` в `MovieController` при создании его экземпляра.

## Добавим интерактивности

Попап с подробной информацией о фильме довольно сложный интерактивный компонент. Но это поведение — не часть бизнес-логики приложения. Это бизнес-логика самого



компонента. Поэтому для реализации этой логики введем понятие `SmartComponent` — компонент, который может себя перерисовывать.

1. Создайте абстрактный класс `AbstractSmartComponent`, унаследовав его от `AbstractComponent`, с двумя методами:

- абстрактный метод `recoveryListeners`, его нужно будет реализовать в наследнике. Его задача — восстанавливать обработчики событий после перерисовки;
- обычный метод `render`, его задачи:
  - удалить старый DOM-элемент компонента;
  - создать новый DOM-элемент;
  - поместить новый элемент вместо старого;
  - восстановить обработчики событий, вызвав `recoveryListeners`.

2. Унаследуйте компонент `Movie` с подробной информацией о фильме от `AbstractSmartComponent` и объявите в компоненте метод `recoveryListeners`, пока

пустым.

3. Теперь нужно реализовать перерисовку попапа после взаимодействия с пользователем:

- при клике на эмоцию подставлять её в соответствующий блок.

**Обратите внимание**, что, кроме визуальной подстановки эмоции, нужно также заполнить значение скрытого поля ввода.

4. При перерисовке компонента все обработчики событий будут утеряны, поэтому их нужно навесить заново в методе `recoveryListeners`.

## Отообразим только одну форму

Мы научились обновлять данные и создавать интерактивные компоненты. Осталось запретить открывать несколько попапов одновременно. Мы реализуем это простым способом:

в `PageController` мы «прикажем» всем экземплярам `MovieController` вернуться в исходное состояние, когда пользователь открывает попап.

1. Добавьте метод `setDefaultView` в `MovieController` для скрытия попапа с подробной информацией о фильме.
2. Добавьте в конструктор `MovieController` добавьте новый аргумент — функцию `onViewChange` и вызывайте её перед тем, как показать попап с подробной

информацией о фильме.

3. В `PageController` опишите метод `_onViewChange`, где вызовите у всех экземпляров `MovieController` метод `setDefaultView`.
4. В `PageController` при создании экземпляров `MovieController` передайте и метод `_onViewChange`.

## Bigtrip

Сперва разберёмся с кодом, который скопился у нас в `TripController` и отвечает за смену точки маршрута на форму редактирования. Для этого создадим `PointController`:

1. В нем нужно описать конструктор и метод `render`
2. Конструктор должен принимать `container` — элемент, в который контроллер будет всё отрисовывать.
3. Метод `render` должен принимать данные одной точки маршрута. Также в него должен переехать код, который отвечает за отрисовку точки маршрута, ее замену на форму редактирования и наоборот, а также установка связанных с этим обработчиков событий.

## Настроим частичный датабиндинг

Реализуем обработку кликов на кнопке «Favorite» (звёздочка) в форме редактирования. Обработчики должны изменять данные точки маршрута — добавлять или удалять из избранного, и на основе изменённых данных перерисовывать компонент.

Для этого:

1. В компоненте формы редактирования добавьте метод для установки обработчика клика для звёздочки.
2. В конструкторе `PointController` добавьте новый аргумент — функцию `onDataChange`. Эта функция должна вызываться в обработчике клика и получать на вход старые и новые данные (точку маршрута и измененную точку маршрута).

**Обратите внимание**, пока что мы занимаемся обновлением только признаков точки маршрута, поэтому при сохранении у нас по-прежнему форма просто заменяется на точку маршрута, а удаление не работает вовсе. С этим мы разберёмся в следующих заданиях.

3. В `TripController` опишите метод `_onDataChange` с точно таким же интерфейсом, как функция `onDataChange`. Задача метода — обновить моки и вызывать метод `render` у конкретного экземпляра `PointController` с обновлёнными данными.
4. Чтобы всё заработало, передайте метод `_onDataChange` в `PointController` при создании его экземпляра.

**Добавим интерактивности**

Форма редактирования довольно сложный интерактивный компонент. Но это поведение — не часть бизнес-логики приложения. Это бизнес-логика самого компонента. Поэтому для реализации этой логики введем понятие `SmartComponent` — компонент, который может себя перерисовывать.

1. Создайте абстрактный класс `AbstractSmartComponent`, унаследовав его от `AbstractComponent`, с двумя методами:

- абстрактный метод `recoveryListeners`, его нужно будет реализовать в наследнике. Его задача — восстанавливать обработчики событий после перерисовки;
- обычный метод `render`, его задачи:
  - удалить старый DOM-элемент компонента;
  - создать новый DOM-элемент;
  - поместить новый элемент вместо старого;
  - восстановить обработчики событий, вызвав `recoveryListeners`.

2. Унаследуйте компонент формы редактирования от `AbstractSmartComponent` и объявите в компоненте метод `recoveryListeners`, пока пустым.
3. Теперь нужно реализовать перерисовку формы редактирования после взаимодействия с пользователем:
  - при смене типа точки маршрута нужно менять плейсхолдер у пункта назначения и показывать соответствующий типу набор дополнительных опций;
  - при выборе пункта назначения нужно менять описание.

**Обратите внимание**, что при смене дополнительных опций выбранные значения не сохраняются, а ещё выбор дополнительных опций не влияет на стоимость, указанную в соответствующем поле ввода.

4. При перерисовке компонента все обработчики событий будут утеряны, поэтому их нужно навесить заново в методе `recoveryListeners`.

## Отообразим только одну форму

Мы научились обновлять данные и создавать интерактивные компоненты. Осталось запретить открывать несколько форм редактирования одновременно. Мы реализуем это простым способом: в `TripController` мы «прикажем» всем экземплярам `PointController` вернуться в исходное

состояние, когда пользователь открывает форму редактирования.

1. Добавьте метод `setDefaultView` в `PointController` для отображения точки маршрута вместо формы редактирования.
2. Добавьте в конструктор `PointController` новый аргумент — функцию `onViewChange` и вызывайте её перед тем, как сменить точку маршрута на форму редактирования.
3. В `TripController` опишите метод `_onViewChange`, где вызовете у всех экземпляров `PointController` метод `setDefaultView`.
4. В `TripController` при создании экземпляров `PointController` передайте и метод `_onViewChange`.