

Лекция 8: Поведенческие шаблоны

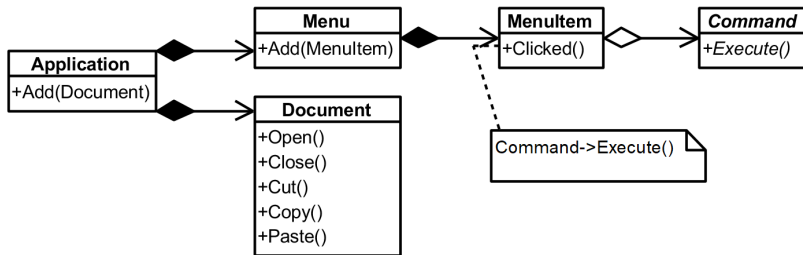
Юрий Литвинов
yurii.litvinov@gmail.com

28.04.2020г

Паттерн “Команда”, мотивация

- ▶ Хотим отделить инициацию запроса от его исполнения
- ▶ Хотим, чтобы тот, кто “активирует” запрос, не знал, как он исполняется
- ▶ При этом хотим, чтобы тот, кто знает, когда исполнится запрос, не знал, когда он будет активирован
- ▶ Но зачем?
 - ▶ Команды меню приложения
 - ▶ Палитры инструментов
 - ▶ ...
- ▶ “Просто вызвать действие” не получится, вызов функции жёстко свяжет инициатора и исполнителя

Решение: обернём действие в объект



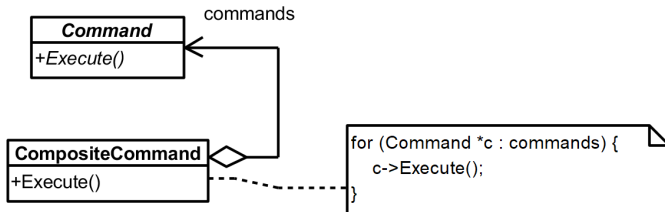
Команда вставки



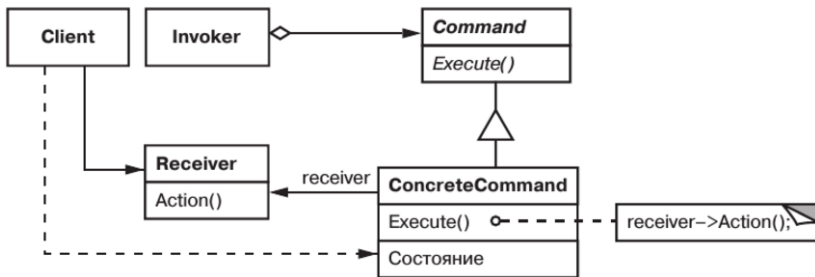
Команда открытия документа



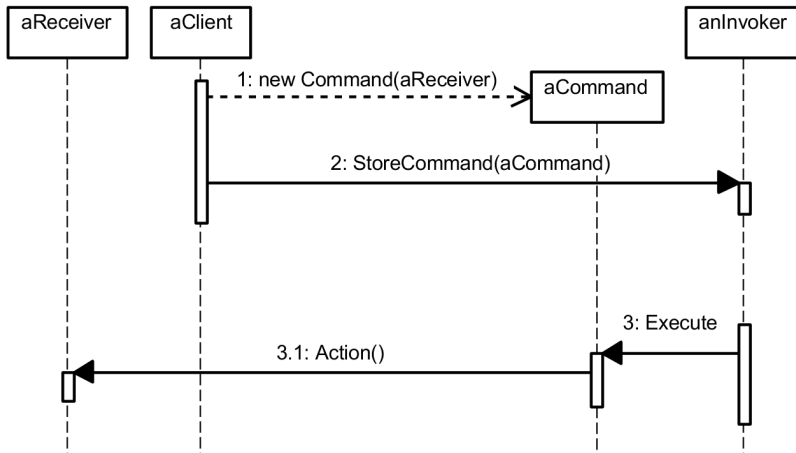
Составная команда



Паттерн "Команда"



Взаимодействие объектов



Команда, применимость

- ▶ Параметризовать объекты выполняемым действием
- ▶ Определять, ставить в очередь и выполнять запросы в разное время
- ▶ Поддерживать отмену операций
- ▶ Структурировать систему на основе высокоуровневых операций, построенных из примитивных
- ▶ Поддерживать протоколирование изменений

“Команда” (Command), детали реализации

- ▶ Насколько “умной” должна быть команда
- ▶ Отмена и повторение операций — тоже от хранения всего состояния в команде до “вычислимого” отката
 - ▶ Undo-стек и Redo-стек
 - ▶ Может потребоваться копировать команды
 - ▶ “Искусственные” команды
 - ▶ Композитные команды
- ▶ Паттерн “Хранитель” для избежания ошибок восстановления

"Команда", пример

- ▶ Qt, класс QAction:

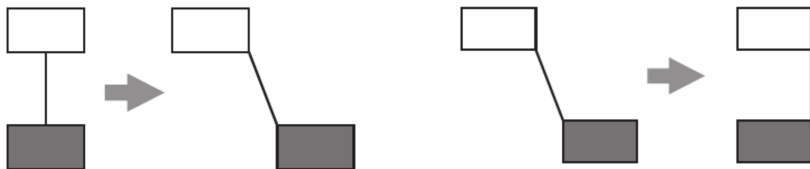
```
const QIcon openIcon = QIcon(":/images/open.png");  
QAction *openAct = new QAction(openIcon, tr("&Open..."), this);
```

```
openAct->setShortcuts(QKeySequence::Open);  
openAct->setStatusTip(tr("Open an existing file"));
```

```
connect(openAct, &QAction::triggered, this, &MainWindow::open);
```

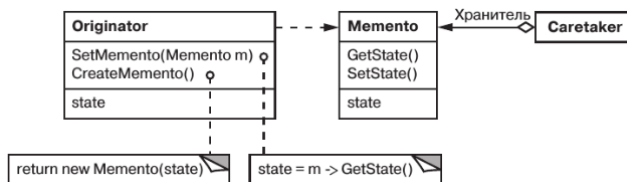
```
fileMenu->addAction(openAct);  
fileToolBar->addAction(openAct);
```

Паттерн “Хранитель”, мотивация



- ▶ Хотим уметь фиксировать внутреннее состояние объектов
- ▶ И восстанавливать его при необходимости
- ▶ Не раскрывая внутреннего устройства объектов кому не надо

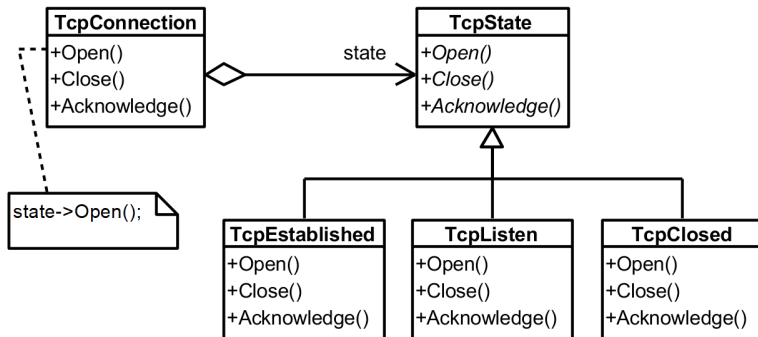
Паттерн “Хранитель”



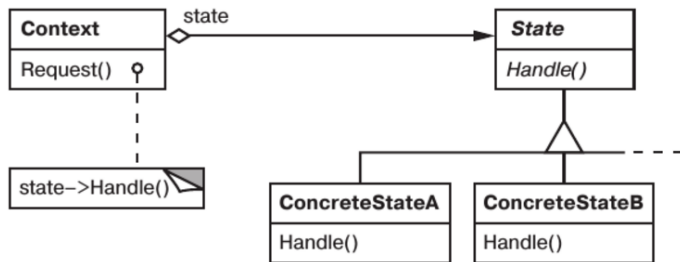
“Хранитель” (Memento), детали реализации

- ▶ Два интерфейса: “широкий” для хозяев и “узкий” для остальных объектов
 - ▶ Требуется языковая поддержка
- ▶ Можно хранить только дельты состояний

Паттерн "Состояние", мотивация



Паттерн "Состояние"



“Состояние” (State), детали реализации

- ▶ Переходы между состояниями — в Context или в State?
- ▶ Таблица переходов
 - ▶ Трудно добавить действия по переходу
- ▶ Создание и уничтожение состояний
 - ▶ Создать раз и навсегда
 - ▶ Создавать и удалять при переходах

“Состояние” результаты

- ▶ Локализует зависящее от состояния поведение
- ▶ Делает явными переходы между состояниями
- ▶ Объекты состояния можно разделять

Когда применять:

- ▶ Поведение объекта зависит от его состояния и должно изменяться во время выполнения
- ▶ Обилие условных операторов, в которых выбор ветви зависит от состояния