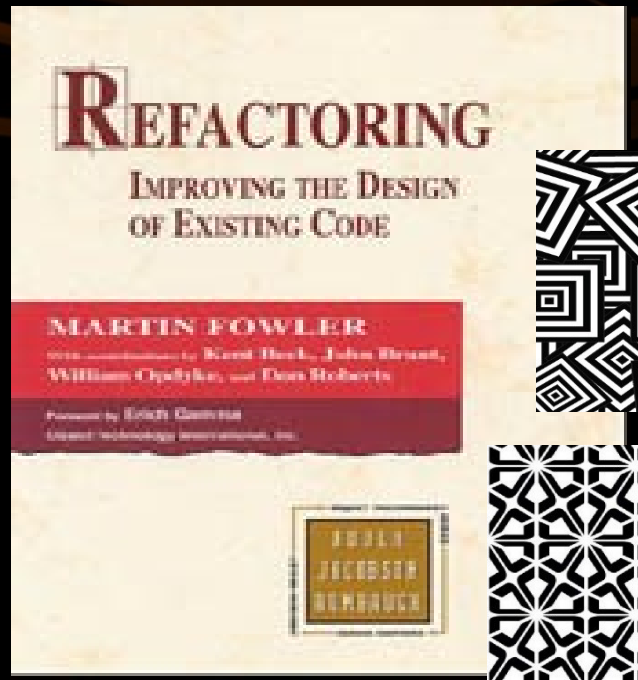
Шаблони за преработка на кода



Учителски екип

Обучение за ИТ кариера

https://it-kariera.mon.bg/e-learning/

Съдържание

- Шаблони за преработка
  - Façade шаблон
- Нива на преработка
  - На ниво данни
  - На ниво изрази
  - На ниво методи
  - На ниво клас
  - На системно ниво

Шаблони за преработка на кода

- Кога се налага преработка на кода?
  - Неприятното усещане в кода подсказва нуждата от преработка
- Компонентните тестове гарантират, че при преработката ще се запази поведението
- Шаблони за преработка
  - Големи повтарящи се фрагменти код → извличане на дублирания код в метод
  - Големи методи → разделяме ги локално на части
  - Голямо тяло на цикъл или дълбоко влагане → отива в метод

Шаблони за преработка (2)

- Клас или метод има слаба свързаност → разделя се на няколко класа / метода
- Една промяна се разпространява в няколко класа → класовете имат силна зависимост → плаче за преработка
- Свързани данни винаги се ползват заедно, но не са част от един клас → да се групират в клас
- Метод има твърде много параметри → създаване на клас, който да обедини параметрите
- Метод вика повече методи от чужд клас, вместо от своя → местим го

Шаблони за преработка (3)

- Два класа са силно зависими → сливат се или се преработват, за да си разделят отговорностите
- Public полета, които не са константни → стават private и се дефинират свойства за достъп
- Мистериозни числа в кода → дали да не станат константи?
- Лошо именуван клас / метод / променлива → преименува се
- Много сложно булево условие → разделя се на няколко израза или извиквания на методи

Шаблони за преработка (4)

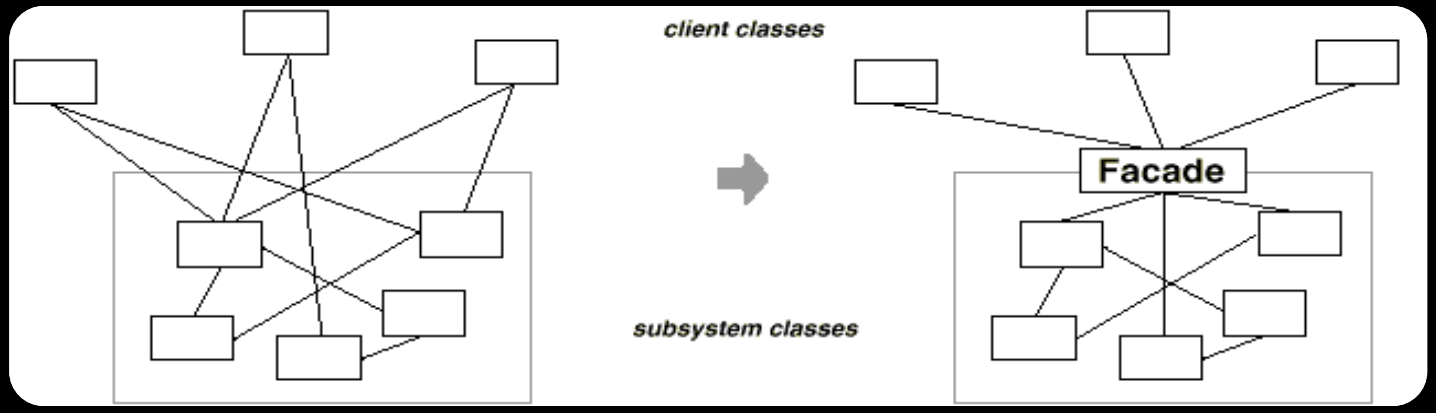
- Сложен израз → разделя се на няколко прости части
- Няколко константи, ползвани като изброим тип → преобразуват се в изброим тип
- Твърде сложна логика в метода → разделя се на няколко по-прости метода или дори се създава нов клас
- Неизползвани класове, методи, параметри, променливи → махат се
- Голямо количество данни, предавани по стойност без особено добра причина → да се предават по адрес

Шаблони за преработка (5)

- Няколко класа, споделящи една и съща функционалност → извлича се общ родителски клас и се слага в него общия код
- Различни класове биват създавани в зависимост от някаква настройка в конфигурацията → използва се factory шаблона
- Кодът не е добре форматиран → преподреждане
- Твърде много класове в едно пространство от имена → разделят се класовете смислово в няколко пространства от имена
- Неизползвани `using` дефиниции → махат се
- Неописателни съобщения за грешки → оправят се
- Липса на защитно програмиране → добавя се

Шаблон Façade

- За предоставяне на удобен интерфейс от високо ниво към множество подсистеми или една сложна подсистема
- Използван в много Win32 API базови класове, за да се скрие сложността на Win32



- http://www.dofactory.com/net/facade-design-pattern

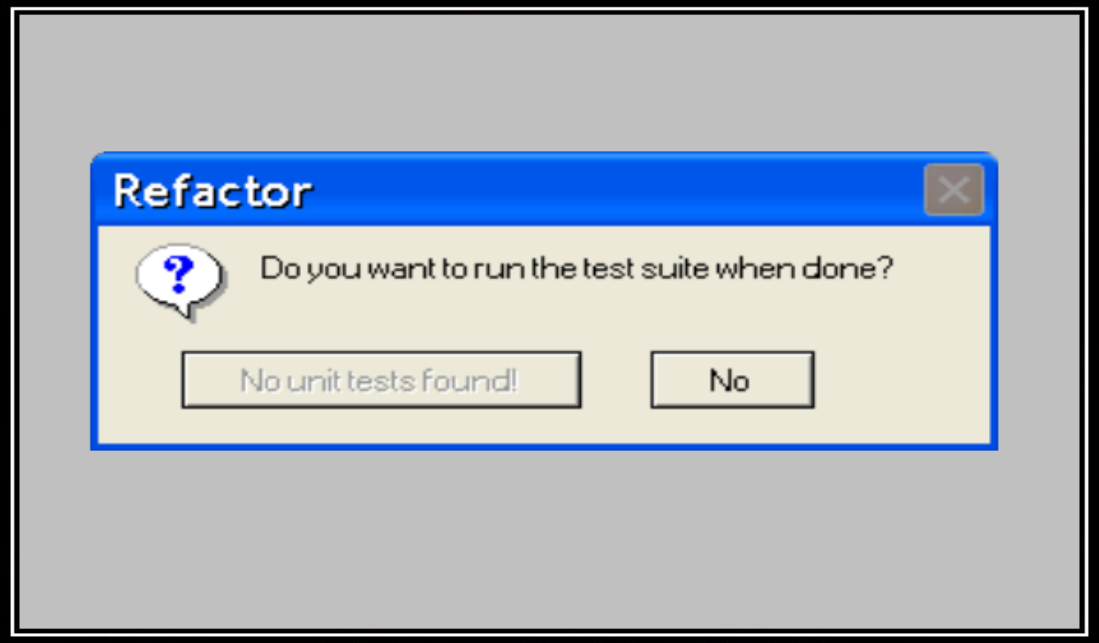
Шаблон Façade – пример

- По сложния начин:

```
speakers.On();
speakers.SetSurroundSound(true);
speakers.SetVolume(25/100);
speakers.SetOptions(SoundOptions.BlueRay);
environment.DimLights();
projector.On();
projector.SetMode(Modes.WideScreen);
dvd.On();
dvd.Play(movieName);
```

- С Façade:

```
homeTheater.WatchMovie(movieName)
```



IRONY

Is blaming frequent refactoring for not writing unit tests

Нива на преработка

Преработка на ниво данни

- Замяна на магически числа с именувани константи
- Преименуване на променлива с по-описателно име
- Замяна на израз с метод
  - За опростяване или за да се избегнат повторения
- Преместване на израз да е inline
- Добавяне на междинна променлива
  - Въвежда поясняваща променлива
- Замяна на променлива с много употреби в множество променливи, ползвани за едно нещо
  - Създава се отделна променлива за всяка употреба

Преработка на ниво данни (2)

- Създава локална променлива за локалните обработки вместо да ползва параметър за целта
- Преобразуване на прости данни в клас
  - Предлага и логика за валидиране (money)
- Преобразува набор от кодове на типове (константи) в `enum`
- Преобразува набор от кодове на типове в клас с подкласове с различно поведение
- Смяна на масив с обект
  - Когато се използва масив с различни типове в него
- Капсулиране на колекция

Преработка на ниво израз

- Разделяне на части на булев израз
- Местене на сложен булев израз в ясно именувана булева функция
- Използване на `break` или `return` вместо управляващата променлива на цикъл
- Връщане на резултат веднага щом е ясен отговора вместо да се присвоява връщаната стойност
- Обединяване на дублирания код в условни команди
- Замяна на условните команди с полиморфизъм
- Използване на null-object шаблона вместо проверки за `null`

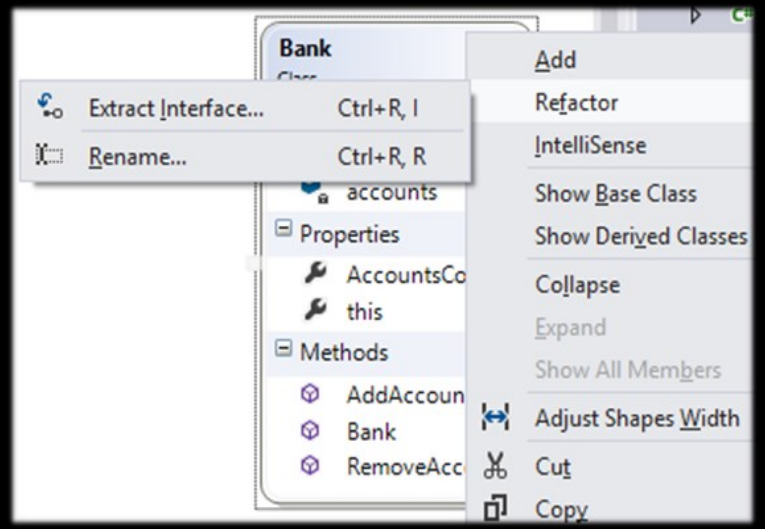
Преработка на ниво метод

- Извличане / вграждане на метод
- Преименуване на метод
- Превръщане на дълга процедура в клас
- Добавяне / махане на параметри

- Комбиниране на подобни методи чрез параметризиране
- Замяна на сложен алгоритъм с по-прост
- Разделяна на методи, чието поведение зависи от подадените параметри (създаване на нови методи)
- Подаване на целия обект вместо само негови отделни полета
- Капсулиране надолу / връщане на интерфейсни типове

Преработка на ниво клас

- Промяна на структура в клас или обратно
- Местене на членове нагоре / надолу по йерархията
- Извличане на специализирания код в подклас
- Комбиниране на подобния код в общ родителски клас
- Свиване на йерархията
- Замяна на наследяване с делегиране
- Замяна на делегиране с наследяване



Преработка на ниво интерфейс на класа

- Извличане на интерфейс(и) / запазване на изолацията им
- Преместване на метод в друг клас
- Разцепване на клас / сливане на класове / изтриване на клас
- Скриване на делегиращ клас
  - **A** вика **B** и **C** когато **A** трябва да извиква **B** и **B** да извиква **C**
- Премахване на посредника
- Въвеждане (и използване) на клас за разширение
  - Когато няма достъп до оригиналния клас
  - Или като алтернатива да се ползва Decorator шаблона

Преработка на ниво интерфейс на класа (2)

- Капсулиране на член-променлива, до която има достъп
  - Винаги ползвайте свойства
  - Подсигурете подходящ достъп до getters и setters
    - Премахнете setter-ите на данните само за четене
- Скриване на данни и процедури, които не са предназначени да бъдат ползвани извън класа / йерархията
  - private -> protected -> internal -> public
- Ползване на стратегия за избягване на голяма йерархия от класове
- Прилагане на други шаблони в проектирането за често срещани казуси с класове и йерархии от класове (Façade, Adapter, т.н.)

Преработка на системно ниво

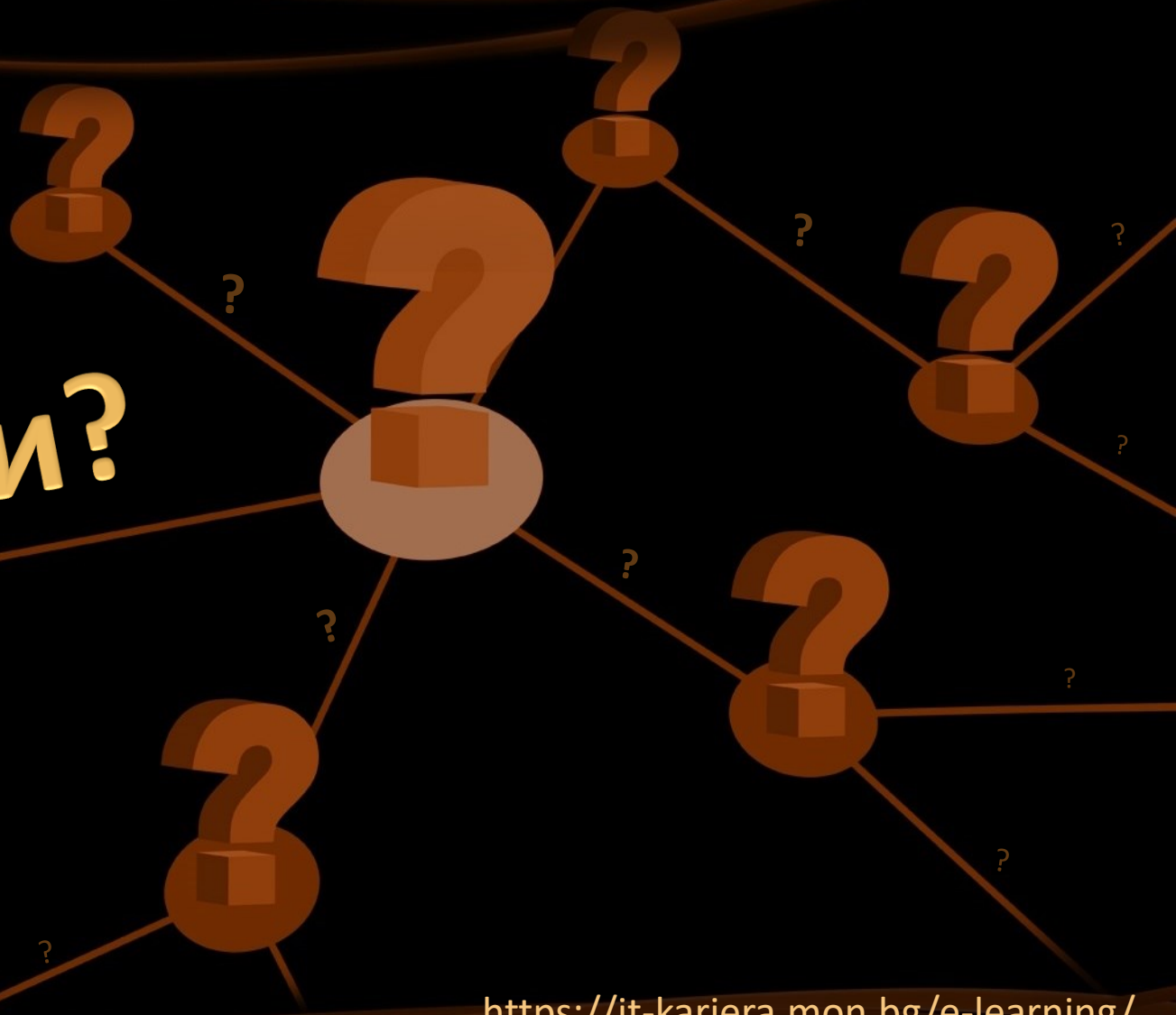
- Преместване на клас (или няколко класа) в друго пространство от имена или асембли
- Предоставяне на factory метод вместо прост конструктор / използване на fluent API
- Смяна на код за грешки с изключения
- Изнасяне на низове в ресурсни файлове
- Ползване на dependency injection
- Прилагане на архитектурни шаблони

Обобщение

1. Шаблони за преработка
2. Видове преработка
   - На ниво данни
   - На ниво изрази
   - На ниво метод
   - На ниво клас
   - На системно ниво

Шаблони за преработка на кода

Въпроси?



Министерство на образованието и науката (МОН)

- Настоящият курс (презентации, примери, задачи, упражнения и др.) е разработен за нуждите на Национална програма "**Обучение за ИТ кариера**" на МОН за подготовка по професия "Приложен програмист"

- Курсът е базиран на учебно съдържание и методика, предоставени от **фондация "Софтуерен университет"** и се разпространява под свободен лиценз **CC-BY-NC-SA**