Programação Avançada 2021-22

Técnicas de Refatoring (1)

- Extract Method
- Form Template Method
- Extract Class

Bruno Silva, Patrícia Macedo

Sumário 📝

- Classificação das Tecnicas de Refatoring segundo Martin Fowler
- Extract Method
- Form Template Method
- Extract Class

Classificação das Técnicas de Refactoring

As tecnicas de refactoring servem para resolver os problemas identificados. A cada BAD smell está relacionado uma ou mais tecnicas de refactoring a aplicar.

- Martin Fowler categorizou as técnicas Refactoring em 6 categorias:
  - Composing Methods
  - Moving Features Between Objects
  - Organizing Data
  - Simplifying Conditional
  - Making Method Calls Simpler
  - Dealing with Generalization

Composing Methods

Estas técnicas de Refactoring tem como objectivo resolver problemas relacionado com métodos demasiado extensos ou curtos e código duplicado. Tal como o nome sugere os métodos são decompostos e compostos para melhorar o desenho da aplicação:

- Extract Method ⭐
- InlineMethod
- Replace Temp with Query
- Slipt Temporary Variables
- Replace Method with Method Object
- Remove Assignemnet to parameters
- Substitute Algoritm

Moving Features Between Objects

Estas técnicas de Refactoring tem como objectivos:
(i) a transferência de funcionalidade entre classes
(ii) a criação de novas classes e
(iii) o encapsulamento de detalhes de implementação.

- Move Method
- Move Field
- Extract Class ⭐
- Inline Class
- Hide Delegate
- Remove Middle Man
- Introduce Foreign Method
- Introduce Local Extension

Organizing Data

Estas técnicas de Refactoring têm como objetivo tornar a manipulação dos dados mais eficente.

- Self Encapsulate Field
- Replace Data Value with Object
- Change Value to Reference
- Change Reference to Value
- Replace Array with Object
- Duplicate Observed Data
- Change Unidirectional Association to Bidirectional
- Change Bidirectional Association to Unidirectional
- Replace Magic Number with Symbolic Constan
- etc...

Simplifying Conditional Expressions

Estas técnicas de Refactoring têm como objectivo melhorar a complexidade introduzida pelas expressões condicionais

- Decompose Conditional
- Consolidate Conditional Expression
- Consolidate Duplicate Conditional Fragments
- Remove Control Flag
- Replace Nested Conditional with Guard Clauses *
- Replace Conditional with Polymorphism
- Introduce Null Object
- Introduce Assertion

Making Method Calls Simpler

Estas técnicas de Refactoring têm como objectivo tornar as chamados aos métodos mais simples e fáceis de compreender.

- Rename Method
- Add Parameter
- Remove Parameter
- Separate Query from Modifier
- Preserve Whole Object
- Introduce Parameter Object
- Hide Method
- Replace Constructor with Factory Method
- Replace Error Code with Exception
- Replace Exception with Test
- etc...

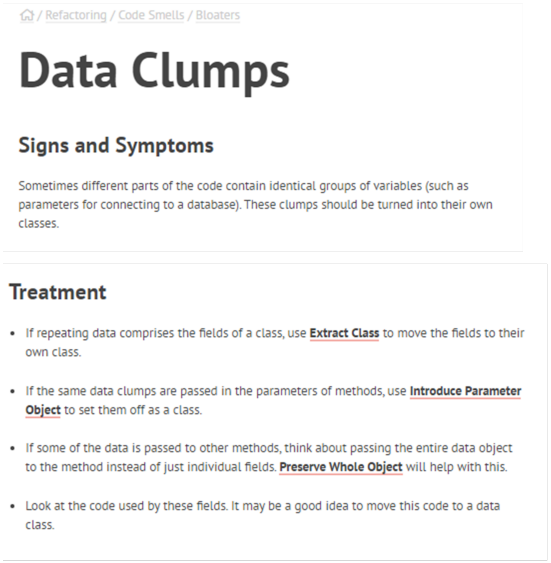
Dealing with Generalization

Estas técnicas de Refactoring têm como objetivo mover funcionalidades ao longo da hierarquia de classes,criando novas classes e interfaces e substituindo herança por delegação e vice-versa.

- Pull Up Field
- Push Down Method
- Push Down Field
- Extract Subclass
- Extract Superclass
- Collapse Hierarchy
- Form Template Method ⭐
- Replace Inheritance with Delegation
- Replace Delegation with Inheritance
- etc...

Qual a técnica a aplicar ?

- Para cada **BAD SMELL** existe a indicação de um conjunto de técnicas de refactoring associadas.
- Dentro das técnicas propostas seleciona-se a mais adequada à situação.



/ Refactoring / Code Smells / Bloaters

Data Clumps

Signs and Symptoms

Sometimes different parts of the code contain identical groups of variables (such as parameters for connecting to a database). These clumps should be turned into their own classes.

Treatment

- If repeating data comprises the fields of a class, use **Extract Class** to move the fields to their own class.
- If the same data clumps are passed in the parameters of methods, use **Introduce Parameter Object** to set them off as a class.
- If some of the data is passed to other methods, think about passing the entire data object to the method instead of just individual fields. **Preserve Whole Object** will help with this.
- Look at the code used by these fields. It may be a good idea to move this code to a data class.

https://refactoring.guru/refactoring/smells

Técnica: Extract Method

⭐Aplica-se para resolver vários tipos de problemas: código duplicado, métodos longos, switch statements etc...
⭐É usado como um passo intermédio em técnicas mais complexas (Form Template Method, Hide Method etc..)

- **Sumário**: Um método tem um fragmento de código que pode ser agrupado, então um novo método é criado com esta porção de código e referenciado no método que original.
- **Mecanismo**:
  - Criar um novo método, e dar-lhe o nome da intenção do método.
  - Copie o código extraído do método da fonte para o novo método de destino.
  - Verificar as referencias às variaiveis.
  - Substituir o código extraído no método de origem por uma chamada ao método de destino.

Extract Method: Exemplo

Inicial

```
void printOwing() {
  printBanner();

  // Print details.
  System.out.println("name: " + name);
  System.out.println("amount: " + getOutstanding());
}
```

Após

```
void printOwing() {
  printBanner();
  printDetails(getOutstanding());
}

void printDetails(double outstanding) {
  System.out.println("name: " + name);
  System.out.println("amount: " + outstanding);
}
```

Atividade 1

- Aplique a técnica de Refactoring **Extract Method**, de forma a remover o **BAD SMELL** Duplicated Code
- Ver código base em: https://github.com/estsetubal-pa-geral/ExtractMethod_Template

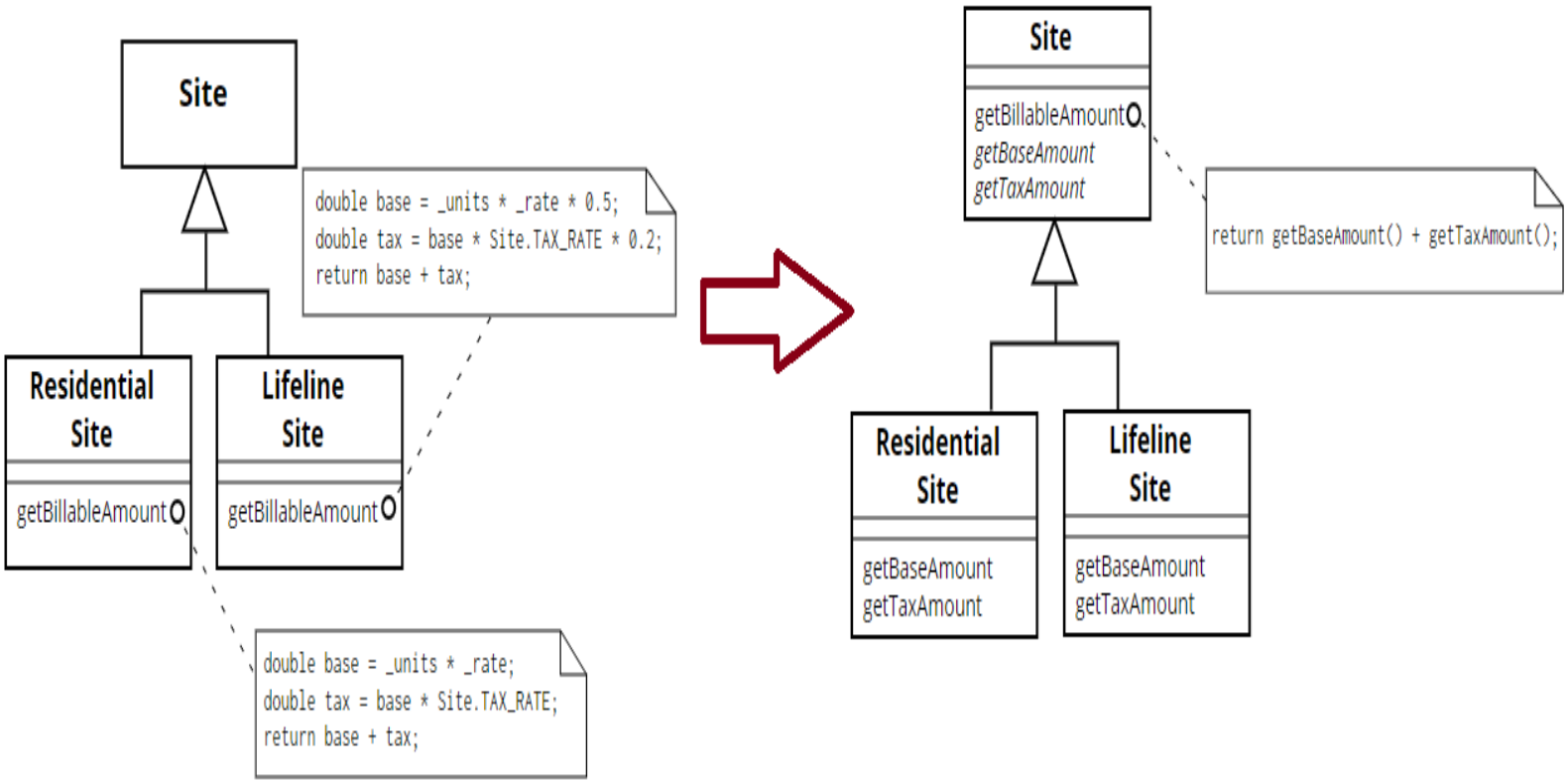
Técnica: Form Template Method

Sumário: Duas subclasses implementam algoritmos que contêm passos semelhantes na mesma ordem. Move-se a estrutura do algoritmo e passos idênticos para uma superclasse e deixa-se a implementação dos diferentes passos nas subclasses.
Mecanismo:

1. Dividir os algoritmos das subclasses nas suas partes constituintes em métodos separados - **Extract Method**
2. Os métodos resultantes que são idênticos para todas as subclasses podem ser movidos para uma superclasse através **Pull Up Method**.Os métodos não semelhantes devem receber nomes consistentes - **Rename Method**.
3. Mover as assinaturas de métodos não semelhantes para uma superclasse como métodos abstractos, utilizando **Pull Up Method**. Deixar as suas implementações nas subclasses.
4. **"Pull Up"** o método principal do algoritmo para a superclasse.

Form Template Method: Exemplo

Código repetido no método `getBillableAmount` nas subclasses `ResidentialSite` e `LifelineSite`



Atividade 2

- Crie um projeto baseado no template: https://github.com/estsetubal-pa-geral/TemplateMethod_Template
- Aplique a técnica **Form Template Method** de forma a resolver a duplicação de código (**Duplicated Code**) no método `formatMessage` nas subclasses `B` e `C`.

Técnica: Extract Class

Sumário: Tem-se uma classe faz o trabalho de duas, então cria-se uma nova classe e coloca-se nela os atributos e métodos responsáveis pela funcionalidade relevante.
Mecanismo:

1. Decida sobre a forma exacta como pretende dividir as responsabilidades da classe.
2. Criar uma nova classe para conter a responsabilidade relevante.
3. Criar uma relação entre a classe antiga e a nova.
4. Use **Move Field** e **Move Method** para cada atributo e método que tenha decidido mudar para a nova classe.
5. Pense também na acessibilidade ao novo par de classes a partir do exterior. Pode esconder a classe do cliente por completo, tornando-a privada ou pode torná-la pública, permitindo ao cliente alterar directamente os valores.

Extract Class: Exemplo

Inicial - Classe `Person`

```
public class Person {
  private String  name, streetAdress, cityName;
  public String getName() { return this.name; }
  public String getCityName() { return this.cityName; }
  public String getStreetAdress() { return this.streetAdress; }
}
```

Após - Extraiu-se a classe `Address` da classe `Person`

```
public class Person {
  private String  name;
  private Address address;
  public String getName() { return this.name; }
  public String getCityName() { return this.address.getCityName(); }
  public String getStreetAdress() { return this.adress.getStreetAddress(); }
}
public class Address {
  private String  streetAddress, cityName;
  public String getCityName() { return this.cityName; }
  public String getStreetAdress() { return this.streetAdress; }
}
```

Atividade 3

- Crie um projeto baseado no template: https://github.com/estsetubal-pa-geral/ExtractClass_Template
- Aplique a Técnica **Extract Class** de forma a fazer uma correta separação das responsabilidades.
  No fim deverá ter 3 classes:

- `AppMainCalculator` - classe responsável por lançar a aplicação JAVAFX
- `CalculatorUI` - classe responsável por implemenatr a interação com o Utilizador
- `Calculator` - classe responsavel por implemenatr a lógica do calculador

Ver mais em:

- https://refactoring.guru/smells/long-method
- https://refactoring.guru/smells/large-class
- https://refactoring.guru/smells/data-clumps
- https://refactoring.guru/smells/duplicate-code
- https://refactoring.guru/form-template-method
- https://refactoring.guru/extract-method
- https://refactoring.guru/extract-class
