IPS Instituto Politécnico de Setúbal

Escola Superior de Tecnologia de Setúbal

Programação Avançada 2021-22

Introdução ao Refactoring

Bruno Silva, Patrícia Macedo

Sumário 📝

- **Refatoring**
  - O que é o Refatoring
  - Bad Smells
  - Categorias de Bad Smells
  - Exemplo Motivacional
  - Tecnicas de Refatoring

Definição 📑

- **Refactoring** é a alteração de um programa de software, através da melhoria da sua estrutura interna (desenho) sem alterar o seu comportamento.
- O processo de **Refactoring**:
  - perserva a correção do programa
  - “trata” um problema de cada vez
  - recorre ao uso de testes unitários

Vantagens ✨

- Melhora o desenho do programa de software.
- Torna o código mais eficiente.
- Torna o código mais facil de manter.
- Facililita a compreensão do código.
  - Na maioria dos ambientes de desenvolvimeto de software, outras pessoas que não desenvolveram o código terão que o ler.
- Facilita a detetação mais facimente possiveis erros (bugs).

Quando fazer refatoring ?

- Sempre que detetamos uma porção do código em que o desenho pode melhorar.

Como saber que partes devemos melhorar ?

- Existem um conjunto de “sintomas” tipicos (exemplos: métodos longos,código duplicado, switchs, muitos condicionais encadeados, nomes desadequados) que refletem problemas mais profundos no desenho do programa. -> **BAD SMELLS on Code**

Exemplos de BAD SMELLS

Os BAD SMELLS encontram-se definidos e cada conjunto de "sintomas" típicos tem um nome especifico.

- Duplicated Code
- Long Method
- Large Class
- Data Clumps
- Primitive Obsession
- Switch Statements
- Message chain
- Refused Bequest
- etc...

Exemplo

Condidere o seguinte código

```
 public class Animal {
   enum TYPE_ANIMAL {MAMMAL,BIRD,REPTILE} ;
   TYPE_ANIMAL myKind;  // set in constructor
   ...
   String getSkin() {
      switch (myKind) {
         case MAMMAL: return "hair";
         case BIRD: return "feathers";
         case REPTILE: return "scales";
         default: return "integument";
      }
   }
}
```

Qual o BAD SMELL ? -> **SWITCH STATEMENT**

Exemplo

Qual a solução ? -> **Aplicar o polimorfismo**

```
public abstract class Animal {
  ....
   public abstract String getSkin();
}

public class Mammal extends Animal {
   String getSkin() { return "hair"; }
}

public class Bird extends Animal {
   String getSkin() { return "feathers"; }
}

public class Reptile extends Animal {
   String getSkin() { return "scales"; }
}
```

Exemplo

Porque é que se considerou uma melhoria, substituir o **switch statement** por **polimorfismo** ?

- Adicionar um novo tipo de animal, não implica alterar rever e recompilar o código existente.
- Como Mammals, Birds, e Reptiles são animais que podem ser diferenciados de outras formas, assim evitaremos outros Switchs.
- Ficamos a usar os objectos tal como devido.

Exemplo

Teste Unitários e Refatoring

- Sempre que se aplica "refatoring" deveremos usar os testes unitários.
- Deveremos correr os mesmos testes antes e depois de aplicar a alteração ao código para garantir que não houve alteração ao comportamento.

```
public void testGetSkin() {
        assertEquals("hair", myMammal.getSkin());
        assertEquals("feathers", myBird.getSkin());
        assertEquals("scales", myReptile.getSkin());
        assertEquals("integument", myAnimal.getSkin());
}
```

Catálogo de Refatoring

Á semelhança dos padrões de software, também existe uma lista de técnica de refatoring, que se denomina como catálogo de técnicas de refatoring (https://refactoring.guru/refactoring/catalog).

- Martin Fowler foi um dos percursores do refactoring, propondo inicialmente 70 técnicas de Refactoring, dividas nas seguintes categorias:
  - Composing Methods
  - Moving Features Between Objects
  - Organizing Data
  - Simplifying Conditional
  - Making Method Calls Simpler
  - Dealing with Generalization

Tecnica de Refatoring: Especificação

Cada técnica de Refatoring é especificada da seguinte forma:

- **Nome**: Nome da tecnica
- **Sumário**: Descrição sucinta da situação que faz com que seja preciso aplicar a tecnica e a explicação do que é que a tecnica consiste.
- **Mecanismo**:Descrição passo a passo da aplicação da técnica.
- **Exemplo**: Exemplo de Aplicação

Técnica Refatoring

- **Name**: Replace Conditional with Polymorphism
- **Summary**: You have a conditional that chooses different behavior depending on the type of an object.
- **Mechanism**: Move each leg of the conditional to an overriding method in a subclass. Make the original method abstract.
- **Example**: (apresentado anteriormente)

```
public class Animal {
  enum TYPE_ANIMAL {MAMMAL,BIRD,REPTILE} ;
  TYPE_ANIMAL myKind;  // set in constructor
  ...
  String getSkin() {
     switch (myKind) {
        case MAMMAL: return "hair";
        case BIRD: return "feathers";
        case REPTILE: return "scales";
        default: return "integument";
     }
  }
}
```

```
public abstract class Animal {
  ....
   public abstract String getSkin();
}

public class Mammal extends Animal {
   String getSkin() { return "hair"; }
}

public class Bird extends Animal {
   String getSkin() { return "feathers"; }
}

public class Reptile extends Animal {
   String getSkin() { return "scales"; }
}
```

Atividade (15 minutos + 20 minutos)

- A turma é dividida em grupos e a cada grupo é atribuido uma letra:
  - A - message chain, temporary field
  - B - data clumps, duplicate code
  - C - primitive obsession, data class
  - D - refused bequest; inapropriate intimacy
- Para cada uns dos **BAD SMELLS**
  - Procurar o seu significado
  - Escrever código que seja um exemplo do BAD SMELL
- Apresentar à turma

Referências: https://refactoring.guru/refactoring/smells

Rever: Algumas definições

- **Refactoring** - é o processo de modificar um programa de software para melhorar a estrutura interna do código sem alterar seu comportamento externo.
- **Refactoring** (substantivo): mudança feita na estrutura interna do programa de software para facilitar a sua compreensão e melhorar a sua estrutura interna.
- **Refactor** (verbo): acto de reestruturar o programa de software aplicando uma série de tecnicas de refactoring sem alterar seu comportamento observável.
- **Bad Smells on Code**: Sinais de aviso no código de má programação.