



Programação Avançada 2021-22

Técnicas de Refactoring 3

- Replace Magic Number with Symbolic Constant
- Replace Inheritance with Delegation
- Replace Array with Object
- Bidirectional Association to Unidirectional

Bruno Silva, Patrícia Macedo

Sumário

- Técnicas de refactoring | 3
 - Classificação das técnicas segundo Martin Fowler (repetição para contextualizar)
 - Replace Inheritance with Delegation
 - Replace Array with Object
 - Bidirectional Association to Unidirectional
 - Replace Magic Number with Symbolic Constant

Na classificação estarão assinaladas as técnicas cobertas nesta aula com o símbolo ★

Classificação

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

Martin Fowler categorizou as técnicas refactoring em 6 categorias:

- Composing Methods
- Moving Features Between Objects
- **Organizing Data** ★
 - 3 técnicas apresentadas são desta categoria.
- Simplifying Conditional
- Making Method Calls Simpler
- **Dealing with Generalization** ★
 - 1 técnica apresentada é desta categoria.

Organizing Data

Estas técnicas de refactoring têm como objetivo tornar a manipulação dos dados mais eficiente, e.g.:

- 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 Constant** ★

Dealing with Generalization


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


- 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

Replace Inheritance with Delegation

 Aplica-se ao *bad smell* refused bequest.

- **Sumário:** uma subclasse B utiliza apenas um subconjunto de atributos e/ou métodos da superclasse A.
- **Mecanismo:**
 1. substituir a herança por *composição*, adicionando um atributo do tipo A à classe B;
 2. Criar os métodos respetivos desse comportamento e delegar para os métodos de A.

 se a herança for desejável, rever a hierarquia estabelecida. A superclasse não conterà mais do que deveria?

 Altera uma relação "is-a" para uma relação "has-a", coerente com o que foi id

Replace Array with Object



Aplica-se ao *bad smell* primitive obsession.

- **Sumário:** existe um array que contém diferentes tipos de dados

```
String[] row = new String[2];  
row[0] = "Liverpool";  
row[1] = "15";
```

- **Mecanismo:** substituir o array por um objeto com atributos para cada elemento:

```
Performance row = new Performance();  
row.setName("Liverpool");  
row.setWins(15);
```

! é um caso particular de **extract class**, mas diz respeito à organização da informação em vez de responsabilidade de classes.



na classe resultante pode ser adicionada funcionalidade respetiva.

Change Bidirectional Association to Unidirectional

🤔 Aplica-se a uma situação particular do *bad smell* inappropriate intimacy.

- **Sumário:** Existe uma relação bidirecional entre duas classes, mas uma das classes não utiliza a funcionalidade da outra ou a relação é desnecessária.



- **Mecanismo:** ➡ ... (continua)

Change Bidirectional Association to Unidirectional

- Mecanismo:

1. Remover a associação não utilizada, e.g.:
2. Utilizar argumentos em métodos da classe de onde foi removida a associação para lhe passar a informação necessária.
3. Determinada funcionalidade da classe que perde a associação pode ter de ser movida para a outra classe.

👍 simplifica a classe que não necessita da relação;

👍 reduz a dependência de classes. Classes independentes são mais fáceis de manter.

Antes

```
public class Customer {  
  
    private String name;  
    private int id;  
    private double discount;  
    private HashSet<Order> orders;  
  
    public Customer(String name, int id, double discount) {  
        this.name = name;  
        this.id = id;  
        this.discount = discount;  
        orders= new HashSet();  
    }  
  
    public class Order {  
        private int id;  
        private Date date;  
        private double grossPrice;  
        private Customer customer;  
  
        public Order( int id, double grossPrice, Customer customer) {  
            this.id = id;  
            this.date = new Date();  
            this.grossPrice = grossPrice;  
            this.customer=customer;  
            this.customer.addOrder(this);  
        }  
    }  
}
```

Após

```
public class Customer {  
  
    private String name;  
    private int id;  
    private double discount;  
    private HashSet<Order> orders;  
  
    public Customer(String name, int id, double discount) {  
        this.name = name;  
        this.id = id;  
        this.discount = discount;  
        orders= new HashSet();  
    }  
    public void addOrder(Order order){  
        orders.add(order);  
    }  
}  
public class Order {  
    private int id;  
    private Date date;  
    private double grossPrice;  
  
    public Order( int id, double grossPrice) {  
        this.id = id;  
        this.date = new Date();  
        this.grossPrice = grossPrice;  
    }  
}
```

Replace Magic Number with Symbolic Constant

Sumário: O código utiliza um número que tem um significado associado, e.g.:

```
double potentialEnergy(double mass, double height) {  
    return mass * height * 9.81;  
}
```

Mecanismo: substituir esse número por uma constante cujo nome é explanatório, e.g.:

```
static final double GRAVITATIONAL_CONSTANT = 9.81;  
double potentialEnergy(double mass, double height) {  
    return mass * height * GRAVITATIONAL_CONSTANT;  
}
```

👍 é mais fácil substituir o valor de uma constante do que todas as ocorrências desse valor no código.

Atividade 1

Aplicar as técnicas **replace inheritance with delegation** e **replace magic number with symbolic constant**.

1. Faça *clone* do projeto em: https://github.com/estsetubal-pa-geral/JavaRefactoring_ReplaceInheritanceDelegation
2. Com as técnicas solicitadas, trate o *bad smell* **refused bequest** e o "número mágico".

Atividade 2

Aplicar a técnica **replace array with object**.

1. Faça *clone* do projeto em: https://github.com/estsetubal-pageral/JavaRefactoring_ReplaceArrayObject
2. Com a técnica solicitada, substitua o array que pode ser convertido num objeto.

Atividade 3

Aplicar a técnica **change bidirectional association to unidirectional**.

1. Faça *clone* do projeto em: https://github.com/estsetubal-pa-geral/JavaRefactoring_ChangeBi2Uni
2. Com a técnica solicitada, remova a associação da classe `Order` com a classe `Customer`.
 - Poderão ser necessárias alterações no "cliente", e.g.:

```
order3.changeCustomer(johhny);  
// to  
beatrice.transferOrder(order3, johhny);
```

Ver mais em:

Bad smells:

- <https://refactoring.guru/smells/inappropriate-intimacy>
- <https://refactoring.guru/smells/primitive-obsession>
- <https://refactoring.guru/smells/refused-bequest>

Técnicas:

- <https://refactoring.guru/replace-inheritance-with-delegation>
- <https://refactoring.guru/replace-array-with-object>
- <https://refactoring.guru/change-bidirectional-association-to-unidirectional>
- <https://refactoring.guru/replace-magic-number-with-symbolic-constant>