



Programação Avançada 2021-22

Técnicas de Refactoring (2)

- Hide Delegate
- Move Method

Bruno Silva, Patrícia Macedo

Sumário



- Técnicas de refactoring | 2
 - Classificação das técnicas segundo Martin Fowler (repetição para contextualizar)
 - Hide Delegate
 - Move Method

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
- Simplifying Conditional
- Making Method Calls Simpler
- Dealing with Generalization


Moving Features Between Objects

Estas técnicas de refactoring podem envolver:

1. A transferência de funcionalidade entre classes;
2. A criação ou remoção classes, ou;
3. 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

Hide Delegate

 Aplica-se principalmente aos *bad smells* **message chain** e **inappropriate intimacy**.

- **Sumário:** O cliente obtém um objeto B de um método ou atributo de uma classe A; o cliente então invoca um método da classe B.
- **Mecanismo:** Criar um novo método na classe A que delega a chamada para o objeto B. Agora o cliente não sabe, nem depende, da classe B.

Esconde a delegação do cliente; quanto menos o cliente necessitar de saber sobre relações entre classes, mais fácil será fazer alterações ao programa.

Se resultar num número excessivo de delegações, poderá dar origem ao *bad smell* **middle man**.

Exemplo

Antes

```
public class Circle {  
    private Point center;  
    private int radius;  
  
    public Circle() {  
        this.center = new Point(0,0);  
        this.radius = 1;  
    }  
    public Point getCenter() {  
        return center;  
    }  
}  
  
public static void main(String[] args) {  
    Circle c= new Circle();  
    c.getCenter().setX(1);  
}
```

Após aplicar o Hide Method : moveXCenter

```
public class Circle {  
    private Point center;  
    private int radius;  
    public Circle() {  
        this.center = new Point(0,0);  
        this.radius = 1;  
    }  
    public void moveXCenter(int dx) {  
        center.setX(dx);  
    }  
}  
  
public static void main(String[] args) {  
    Circle c= new Circle();  
    c.moveXCenter(1);  
}
```

Move Method ★

🤧 Aplica-se principalmente aos *bad smells* message chain, inappropriate intimacy, data class e feature envy. Também utilizado em conjunção com outras técnicas, e.g., extract class.

- **Sumário:** Quando a lógica contida num método de uma classe A é mais apropriada numa classe B.
 - O método deve pertencer à classe que contém a maioria da informação necessária pelo método. Isto aumenta a coesão interna das classes.
- **Mecanismo:** ➡ ... (continua)

👍 Reduz interdependências e aumenta a coesão de classes.

Move Method

- Mecanismo:
 1. Declarar o novo método na classe B. Poderá ser dado um nome diferente ao método que seja mais apropriado à classe onde se encontra.
 2. Decidir como a classe A vai "chamar" este método. Se já houver um atributo do tipo B, então utiliza a referência; caso contrário, adicionar atributo do tipo B. Se se tratar agora de um método *estático*, invocá-lo diretamente.
 3. Voltar a analisar o código. O método original da classe A contém funcionalidade adicional necessária ou pode ser removido e substituído pela invocação do novo método em B?

Exemplo: (Antes)

```
class Project {
    Person[] participants;
}
class Person {
    int id;
    public boolean participate(Project p) {
        for(int i=0; i<p.participants.length; i++) {
            if (p.participants[i].id == id) return(true);
        }
        return(false);
    }
}
class MainText{
    public static void main(String[] args){
        Project project;
        Person person;
        // initialization
        if (person.participate(project))
            // code
    }
}
```

Exemplo (Após)

Move Method `participate` da classe `Person` para a classe `Project`

```
class Project {
    Person[] participants;
    boolean participate(Person x) {
        for(int i=0; i<participants.length; i++) {
            if (participants[i].id == x.id) return(true);
        }
        return(false);
    }
}

class Person {
    int id;
}

class MainText{
    public static void main(String[] args){
        Project project;
        Person person;
        // initialization
        if (project.participate(person))
            // code
    }
}
```

Atividade

Aplicar as técnicas **hide delegate** e **move method** a código existente.

- Faça *clone* do projeto em: https://github.com/estsetubal-pa-geral/JavaRefactoring_HideDelegateMoveMethod
 - Cada *package* contém um programa diferente.
- 1. No package/programa `A_inappropriate_intimacy` identifique este *bad smell* e aplique a técnica **hide delegate**.
- 2. No package/programa `B_message_chains` identifique este *bad smell* e aplique a técnica **hide delegate**.
- 3. No package/programa `C_identifysmells` identifique os *bad smells* que ocorrem e aplique as técnicas **hide delegate** e **move method**.

Ver mais em:

Bad smells:

- <https://refactoring.guru/smells/inappropriate-intimacy>
- <https://refactoring.guru/smells/message-chains>
- <https://refactoring.guru/smells/data-class>
- <https://refactoring.guru/smells/feature-envy>

Técnicas:

- <https://refactoring.guru/move-method>
- <https://refactoring.guru/hide-delegate>
