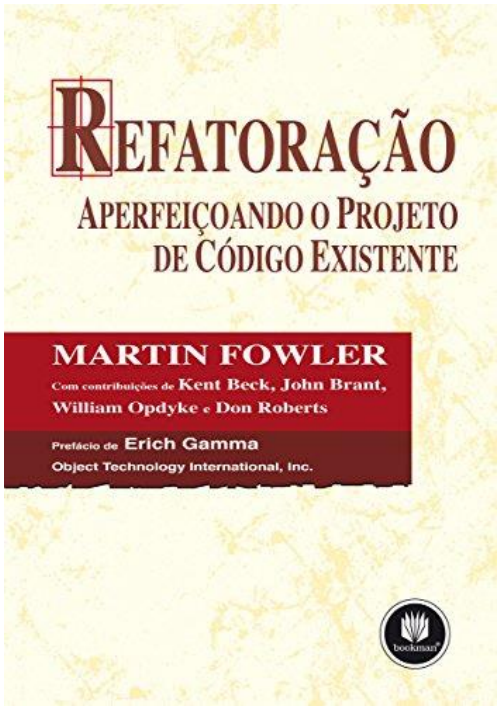
MÓDULO 4

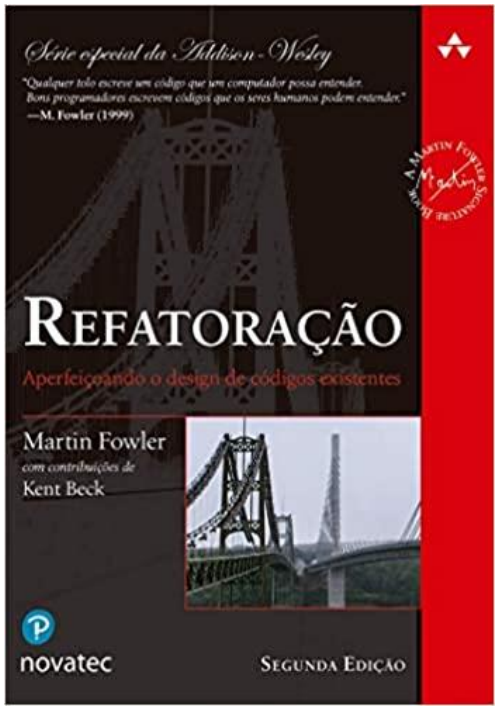
Refatoração

Referência:

- FOWLER, Martin. Refatoração: aperfeiçoando o projeto de código existente, 1ed. Porto Alegre: Bookman, 2004.

- FOWLER, Martin. Refatoração: aperfeiçoando o projeto de código existente, 2ed. Novatec. 2020.





Mover Método

Grupo: Movendo Recursos entre objetos

Resumo: Indicada para momentos em que se depara com métodos que estão fazendo uso excessivo de informações vindas de outras classes.

Permite que o método seja movido para outra classe, a fim de que ele passe a pertencer à classe que mais provem informações para que ele funcione.

Motivação: Classes que possuem diversas funcionalidades podem acabar carregando informações além das que naturalmente deveriam conter.

Alguns métodos existentes nestas classes acabam fazendo com que elas sejam usadas mais do que o normal.

Mover Método

Mecânica:

1) Primeiro passo consiste em analisar a classe que contem o método que será movido. Neste caso, busca-se por recursos da classe que estão sendo usados pelo método. Caso o recurso esteja sendo usado por outros métodos, talvez seja importante movê-los também. Neste caso analisa-se as particularidades de cada aplicação.
2) Aconselha-se analisar, caso haja uma hierarquia de classes, o impacto dessa mudança em outras classes do sistema, antes de mover o método.
3) Na classe que receberá o método a ser movido, declare um método com o mesmo nome ou não do método que se deseja mover.
4) Copia-se o código do corpo do método para o método recém declarado.
5) Faça os ajustes necessários para que o método funcione na classe onde está agora, cuidando para que ele não perca nenhuma funcionalidade que possuía quando estava na outra classe.
6) Execute seus testes para se certificar que tudo correu como planejado.

Mover Método

```
package principal;

import java.util.Scanner;

public class Main {
  public static void main (String[] args) {
    Conta conta = new Conta();
    Scanner input = new Scanner(System.in);

    System.out.println("Informe o número da conta: ");
    conta.setNumero(input.next());

    System.out.println("Informe o valor do depósito: ");
    conta.setSaldo(conta.getSaldo() + input.nextDouble());

    imprimirSaldo(conta.getNumero(), conta.getSaldo());
  }

  public static void imprimirSaldo (String numero, double valor) {
    System.out.println("---Saldo:---");
    System.out.println("Numero: " + numero);
    System.out.println("Valor: " + valor);
    System.out.println("------------");
  }
}
```

Mover Método

```
package principal;

public class Conta {
  String numero = "";
  double saldo = 0;

  public String getNumero () {
    return numero;
  }

  public void setNumero (String numero) {
    this.numero = numero;
  }

  public double getSaldo () {
    return saldo;
  }

  public void setSaldo (double saldo) {
    this.saldo = saldo;
  }
}
```

Mover Método

```
package principal;

import java.util.Scanner;

public class Main {
  public static void main (String[] args) {
    Conta conta = new Conta();
    Scanner input = new Scanner(System.in);

    System.out.println("Informe o número da conta: ");
    conta.setNumero(input.next());

    System.out.println("Informe o valor do depósito: ");
    conta.setSaldo(conta.getSaldo() + input.nextDouble());

    conta.imprimirSaldo();
  }
}
```

Mover Método

```
package principal;

public class Conta {
  String numero = "";
  double saldo = 0;

  public String getNumero () {
    return numero;
  }
  public void setNumero (String numero) {
    this.numero = numero;
  }
  public double getSaldo () {
    return saldo;
  }
  public void setSaldo (double saldo) {
    this.saldo = saldo;
  }
  public void imprimirSaldo () {
    System.out.println("---Saldo:---");
    System.out.println("Numero: " + this.numero);
    System.out.println("Valor: " + this.saldo);
    System.out.println("------------");
  }
}
```

Extrair Método

Grupo: Compondo Métodos

Resumo: A refatoração Extrair Método permite a criação de novos métodos com nomes compatíveis com a função do método.

Motivação:

Encontrar alguns métodos que estão com mais responsabilidades do que deveriam ter.

Torna-se complicado entender qual é o real objetivo do método.

Extrair método sugere que um novo método seja extraído deste trecho de código e que o mesmo passe a possuir parte do código do método que se está refatorando, permitindo que cada método possua uma responsabilidade.

Extrair Método

Mecânica:

1)Identifique um método que possua mais de uma responsabilidade. Crie um novo método com um nome compatível com a responsabilidade extra do método que se está analisando.

2)O código que possui a responsabilidade que se está retirando do método analisado, agora deve ser copiado para o método recém criado.

3)Analise o código movido e veja se há variáveis que estão declaradas no método original. Se sim, faça com que o método criado tenha parâmetros como os das variáveis encontradas.

4)No método novo, verifique a existência de declaração de variáveis. Se existir, modifique-as para que sejam destruídas após a execução do método.

5)No lugar do código que foi extraído do método original, agora deve haver uma chamada ao novo método. Terminado todo o processo, compile o código e teste a aplicação.

Extrair Método

```
package principal;

public class Conta {
  String numero = "";
  double valor = 0;

  public void imprimirSaldo (String nome) {
    // Imprime o cabeçalho:
    System.out.println("Saldo da conta : " + this.numero);
    System.out.println("Cliente VIP");

    // Imprime detalhes:
    System.out.println("Nome: " + nome);
    System.out.println("Valor: " + this.valor);
  }
}
```

Extrair Método

```
package principal;

public class Conta {
  String numero = "";
  double valor = 0;

  public void imprimirSaldo (String nome) {
    imprimirCabecalho();

    imprimirDetalhes(nome);
  }

  public void imprimirDetalhes (String nome) {
    System.out.println("Nome: " + nome);
    System.out.println("Valor: " + this.valor);
  }

  public void imprimirCabecalho () {
    System.out.println("Saldo da conta : " + this.numero);
    System.out.println("Cliente VIP");
  }
}
```

Internalizar Método

Grupo: Compondo Métodos

Resumo: é aplicado quando temos um método que é utilizado apenas uma vez ou porque é resultado de uma refatoração mal feita.

Aconselha-se a remoção desse método e juntar suas funcionalidades ao método que o invoca. Logo, devemos colocá-lo novamente no método que o originou ou então apenas colocá-lo dentro do método que o utiliza, nesse caso não necessariamente a origem dele.

Com isso extrairemos o código desse método não necessário e colocaremos seu código no local da chamada a ele no método cliente, que deve recebê-lo.

Motivação: Alguns desenvolvedores têm o habito de criar pequenos métodos para resolver determinado problema, quando na verdade um método poderia facilmente conter essas funcionalidades sem o carregar de responsabilidades além daquelas que já deveria naturalmente (Internalizar Método é o contrario de Extrair Método).

Internalizar Método

Mecânica:

1)Analisa-se o método a fim de se certificar que ele não é modificado polimorficamente.

2)Busca-se por todas as chamadas a este método pelo sistema e as substitui pelo corpo do método que se deseja Internalizar (remover).

3) Remove-se o método.

4)Executam-se os testes para se certificar que a aplicação ainda possui as mesmas funcionalidades que possuía antes.

Internalizar Método

```
package principal;
public class Conta {
  String numero = "";
  double valor = 0;

  public void imprimirSaldo (String nome) {
    imprimirCabecalho1();
    imprimirDetalhes1(nome);
  }

  public void imprimirCabecalho1 () {
    System.out.println("Saldo da conta : " + this.numero);
    imprimirCabecalho2();
  }

  public void imprimirCabecalho2 () {
    System.out.println("Cliente VIP");
  }

  public void imprimirDetalhes1 (String nome) {
    System.out.println("Nome: " + nome);
    imprimirDetalhes2();
  }

  public void imprimirDetalhes2 () {
    System.out.println("Valor: " + this.valor);
  }
}
```

Internalizar Método

```
package principal;

public class Conta {
  String numero = "";
  double valor = 0;

  public void imprimirSaldo (String nome) {
    imprimirCabecalho();
    imprimirDetalhes(nome);
  }

  public void imprimirDetalhes (String nome) {
    System.out.println("Nome: " + nome);
    System.out.println("Valor: " + this.valor);
  }

  public void imprimirCabecalho () {
    System.out.println("Saldo da conta : " + this.numero);
    System.out.println("Cliente VIP");
  }
}
```

Extrair Interface

Grupo: Lidando com Generalização

Resumo: Em pontos de um sistema, pode-se perceber que há uma grande utilização de alguns métodos de várias classes.

Cria-se, portanto uma interface que permita aos clientes da aplicação apontarem para a interface, ao invés de apontar para diversas classes.

Motivação: Analisando algumas classes de um sistema, pode-se perceber que algumas de suas funcionalidades em comum com as outras classes estão sendo muito utilizadas em diversos pontos da aplicação. Isso dificulta a tarefa do programador que, neste caso, tem que verificar classe por classe buscando essas responsabilidades.

Um recurso é a criação de uma Interface que possua a declaração desse grupo de responsabilidades, permitindo que o programador referencie a interface, ao invés de um grupo de classes.

Extrair Interface

Mecânica:

1) O primeiro passo consiste em criar uma interface vazia.

2) Segundo, declara-se o grupo de operações em comum das diversas classes na interface.

3) Modifique as classes envolvidas para que passem implementar a interface.

4) Faça com que os clientes passem a referenciar a interface ao invés das várias classes.

Extrair Interface

```
public class Cachorro {
    public void comida () {
        System.out.println("ração");
    }
    public void som () {
        System.out.println("latido");
    }
}

public class Gato {
    public void comida () {
        System.out.println("leite");
    }
    public void som () {
        System.out.println("miado");
    }
}

public class Galinha {
    public void comida () {
        System.out.println("milho");
    }
    public void som () {
        System.out.println("cacarejo");
    }
}
```

Extrair Interface

```
public class Main {
public static void main (String[] args) {
      Cachorro cachorro = new Cachorro();
      Gato gato = new Gato();
      Galinha galinha = new Galinha();

      cachorro.comida();
      cachorro.som();

      gato.comida();
      gato.som();

      galinha.comida();
      galinha.som();
   }
}
```

Extrair Interface

```
public interface Animal {
    public void comida ();
    public void som ();
}
```

```
public class Cachorro implements Animal {
    public void comida () {
        System.out.println("ração");
    }
    public void som () {
        System.out.println("latido");
    }
}

public class Gato implements Animal {
    public void comida () {
        System.out.println("leite");
    }
    public void som () {
        System.out.println("miado");
    }
}

public class Galinha  implements Animal {
    public void comida () {
        System.out.println("milho");
    }
    public void som () {
        System.out.println("cacarejo");
    }
}
```

Extrair Interface

```
public class Main {
   public static void main (String[] args) {
      Animal[] animais = new Animal[3];

      animais[0] = new Cachorro();
      animais[1] = new Gato();
      animais[2] = new Galinha();

      for (Animal animal : animais) {
         animal.comida();
         animal.som();
      }
   }
}
```

Remover Parâmetro

Grupo: Tornando as Chamadas de Métodos Mais Simples

Resumo: Alguns métodos possuem parâmetros que não estão sendo usados.
Aplica-se esta refatoração, removendo-se o parâmetro.

Motivação: Alguns métodos apresentam vários parâmetros. Após algumas mudanças no código, pode ser que algum desses parâmetros não esteja mais sendo utilizado, mas está presente porque alguns desenvolvedores julgam que removê-lo não é necessário, dado que posteriormente precisarão dele.

Remover Parâmetro

Mecânica:

1) Analisando o método que contem o parâmetro que deseja remover, certifique-se que realmente ele não está sendo usado, inclusive por algum código que o altere em uma hierarquia de classes.

2) Altere o método, removendo o parâmetro desnecessário.

3) Execute seus testes.

Remover Parâmetro

```
public class Cachorro {
    public void comida (String alimento) {
        System.out.println("ração");
    }
    public void som (String voz) {
        System.out.println("latido");
    }
}

public class Main {
    public static void main (String[] args) {
        String alimento = "osso",
               voz = "ganido";
        Cachorro cachorro = new Cachorro();

        cachorro.comida(alimento);
        cachorro.som(voz);
    }
}
```

Remover Parâmetro

```
public class Cachorro {
    public void comida () {
        System.out.println("ração");
    }
    public void som () {
        System.out.println("latido");
    }
}

public class Main {
    public static void main (String[] args) {
        Cachorro cachorro = new Cachorro();

        cachorro.comida();
        cachorro.som();
    }
}
```

Renomear Método

Grupo: Tornando as Chamadas de Métodos Mais Simples

Resumo: Um método possui um nome que não deixa claro qual é a sua função.

Refatore-o, alterando para um nome que reflita seu objetivo.

Motivação: Alguns métodos possuem nomes que dificultam, à primeira vista, a identificação da sua função, que nestes casos só é descoberta quando se analisa o corpo do método.

Isso dificulta o trabalho de quem está analisando o código.

Indicada para melhorar a legibilidade do código.

Renomear Método

Mecânica:

1) Certifique-se de que o método não tem sua assinatura implementada em uma super ou subclasse. Caso tenha, cada passo da mecânica deve ser repetido em todas as implementações existentes.

2) Renomeio o método, empregando um nome significativo.

3) Modifique os pontos que chamam o método, para que passem a referenciar o novo nome.

4) Execute seus testes.

Renomear Método

```
public class Pessoa {
  String nome;

  public Pessoa(String nome) {
    this.nome = nome;
  }

  public String getN() {
    return this.nome;
  }
}
```

```
public class Main {
   public static void main(String[] args) {
      Pessoa pessoa = new Pessoa("Ana");

      System.out.println(pessoa.getN());
   }
}
```

Renomear Método

```
public class Pessoa {
  String nome;

  public Pessoa(String nome) {
    this.nome = nome;
  }

  public String getNome() {
    return this.nome;
  }
}
```

```
public class Main {
   public static void main(String[] args) {
      Pessoa pessoa = new Pessoa("Ana");

      System.out.println(pessoa.getNome());
   }
}
```

Substituir Algoritmo

Grupo: Compondo Métodos

Resumo: Alguns trechos de código parecem ser complexos de entender e há uma forma mais simples de representar o problema.
Substitui-se o trecho de código complexo por algo mais simples.

Motivação: É comum um desenvolvedor criar um trecho de código que, mais tarde ao lê-lo novamente, encontre uma forma mais simples de resolver o mesmo problema. Permite que o desenvolvedor esteja sempre substituindo código complexo por código mais fácil de entender.

Substituir Algoritmo

Mecânica:

1) Comente o trecho de código que pretende substituir.

2) Crie um novo trecho de código que possua a mesma função do trecho comentado, mas agora cuidando para que seja mais simples que aquele a ser substituído.

3) Teste a aplicação para se certificar que o código criado tem a mesma função que o código a ser substituído.

4) Apague o código comentado.

5) Execute seus testes.

Substituir Algoritmo

```java
public class Main {
  public static void main(String[] args) {
    String[] pessoas = { "Ana", "Bruna", "Carla" };

    for (int i = 0; i < pessoas.length; i++) {
      String pessoa = pessoas[i];
      System.out.println(pessoa);
    }
  }
}
```

```java
public class Main {
  public static void main(String[] args) {
    String[] pessoas = { "Ana", "Bruna", "Carla" };

    for (String pessoa : pessoas)
      System.out.println(pessoa);
  }
}
```

Encapsular Campo

Grupo: Organizando Dados
Resumo: Campos públicos devem ter seus moderadores alterados para privados e métodos de acesso (*gets* e *sets*) devem ser criados para possibilitar o acesso a esses campos.

Motivação: Permitir que campos (ou atributos) de uma classe passem a ter métodos de acesso, permitindo assim que o encapsulamento dos dados seja mantido, o que é um dos princípios da orientação a objetos.

Encapsular Campo

Mecânica:

1) Crie métodos get e set para os atributos que não possuem.

2) Busque pelo código fonte referências a esses atributos e modifique-os para que, a partir deste momento passem a referenciar os métodos de acesso, e não mais os atributos.

3) Execute os testes e certifique-se que tudo está correto, e então altere os modificadores de acesso dos atributos para private.

Encapsular Campo

```
public class Pessoa {
  public String nome;

  public Pessoa(String nome) {
    this.nome = nome;
  }
}


public class Main {
  public static void main (String[] args) {
    Pessoa pessoa = new Pessoa("Ana");

    System.out.println(pessoa.nome);
    pessoa.nome = "Bruna";
    System.out.println(pessoa.nome);
  }
}
```

Encapsular Campo

```
public class Pessoa {
  private String nome;

  public Pessoa(String nome) {
    this.nome = nome;
  }

  public void setNome(String nome) {
    this.nome = nome;
  }

  public String getNome() {
    return this.nome;
  }
}

public class Main {
  public static void main (String[] args) {
    Pessoa pessoa = new Pessoa("Ana");

    System.out.println(pessoa.getNome());
    pessoa.setNome("Bruna");
    System.out.println(pessoa.getNome());
  }
}
```