

## Construção de Software Refatoração

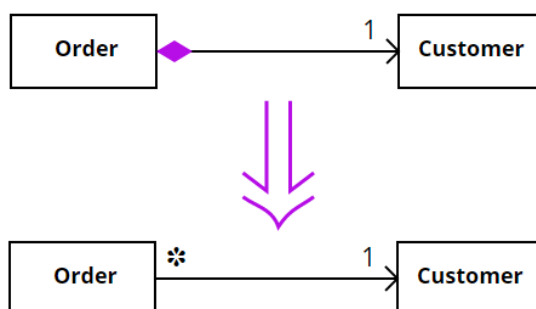
### Exemplo *Change Value to Reference*

Prof. Rubens de Castro Pereira, Me.  
rubens@inf.ufg.br



### Refatorações de Implementação no nível de Classe

- Transforme objetos de valor ou itens de dados em objetos de referência quando houver numerosas cópias de objetos grandes e complexos. Objetos de referência endereçam o objeto mestre.



2


## Change Value to Reference

```

class Cliente {
    private final String _nome;
    public Cliente(String nome) {
        _nome = nome;
    }
    public String getNome() {
        return _nome;
    }
}

class Pedido {
    private Cliente _cliente;
    public Pedido(String nomeCliente) {
        _cliente = new Cliente(nomeCliente);
    }
    public void setCliente(String nomeCliente) {
        _cliente = new Cliente(nomeCliente);
    }
    public String getNomeCliente() {
        return _cliente.getNome();
    }
}

```



```

classDiagram
    Order "1" *-- "1" Customer
    class Customer {
        name: String
    }

```

Prof. Rubens de Castro Pereira

3

## Change Value to Reference

```

private static int numeroPedidos(Collection pedidos,
                                String nomeCliente) {

    int resultado = 0;
    Iterator iteracao = pedidos.iterator();
    while (iteracao.hasNext()) {
        Pedido pedido = (Pedido)iteracao.next();
        if (pedido.getNomeCliente().equals(nomeCliente)) {
            resultado++;
        }
    }
    return resultado;
}

```

Prof. Rubens de Castro Pereira

4

## Change Value to Reference

Refatorando para ...



Prof. Rubens de Castro Pereira

5

## Substituição do construtor por uma fábrica

```
class Cliente {
    private final String _nome;
    public Cliente(String nome) {
        _nome = nome;
    }
    public String getNome() {
        return _nome;
    }
    public static Cliente criar(String nome) {
        return (new Cliente(nome));
    }
}
```

Prof. Rubens de Castro Pereira

6

## Ajuste na criação do pedido utilizando o método de fábrica

```
class Pedido {
    private Cliente _cliente;
    public Pedido(String nomeCliente) {
        _cliente = Pedido.criar(nomeCliente);
    }
    public void setCliente(String nomeCliente) {
        _cliente = new Cliente(nomeCliente);
    }
    public String getNomeCliente() {
        return _cliente.getNome();
    }
}
```

Prof. Rubens de Castro Pereira

7

## Tornar o construtor de cliente privado

```
class Cliente {
    private final String _nome;
    private Cliente(String nome) {
        _nome = nome;
    }
    public String getNome() {
        return _nome;
    }
    public static Cliente criar(String nome) {
        return (new Cliente(nome));
    }
}
```

Prof. Rubens de Castro Pereira

8

## Criar método de acesso a clientes via fábrica (Dictionary ou maps)

```
class Cliente {
    private static Dictionary _instancias = new Hashtable();
    private final String _nome;

    private Cliente(String nome) {
        _nome = nome;
    }
    public String getNome() {
        return _nome;
    }
    public static Cliente criar(String nome) {
        return (new Cliente(nome));
    }
}
```

Prof. Rubens de Castro Pereira

9

## Ajustar classe Cliente para manter todos os clientes

```
class Cliente {
    private static Dictionary _instancias = new Hashtable();
    private final String _nome;
    private Cliente(String nome) { _nome = nome; }
    public String getNome() { return _nome; }
    public static Cliente criar(String nome) {
        return (new Cliente(nome));
    }
    static void carregarClientes() {
        new Cliente("Lemon Car Hire").armazenar();
        new Cliente("Associated Coffee Machines").armazenar();
        new Cliente("Bilston Gasworks").armazenar();
    }
    private void armazenar() {
        _instancias.put(this.getNome(), this);
    }
}
```

Prof. Rubens de Castro Pereira

10

## Ajustar o método criar de Cliente, retornando sempre um cliente pré-existente

```
class Cliente {
    private static Dictionary _instancias = new Hashtable();
    private final String _nome;
    private Cliente(String nome) { _nome = nome; }
    public String getNome() { return _nome; }
    public static Cliente criar(String nome) {
        return (Cliente)_instancias.get(nome);
    }
    static void carregarClientes() {
        new Cliente("Estacionamento Bem Seguro").armazenar();
        new Cliente("Associacao de Maquinas de Cafe").armazenar();
        new Cliente("Seguro Vai Seguro").armazenar();
    }
    private void armazenar() {
        _instancias.put(this.getNome(), this);
    }
}
```

Prof. Rubens de Castro Pereira

11

## Ajustar o método criar de Cliente, retornando sempre um cliente pré-existente

```
class Cliente {
    private static Dictionary _instancias = new Hashtable();
    private final String _nome;
    private Cliente(String nome) { _nome = nome; }
    public String getNome() { return _nome; }
    public static Cliente getCliente(String nome) {
        return (Cliente)_instancias.get(nome);
    }
    static void carregarClientes() {
        new Cliente("Estacionamento Bem Seguro").armazenar();
        new Cliente("Associacao de Maquinas de Cafe").armazenar();
        new Cliente("Seguro Vai Seguro").armazenar();
    }
    private void armazenar() {
        _instancias.put(this.getNome(), this);
    }
}
```

Prof. Rubens de Castro Pereira

12