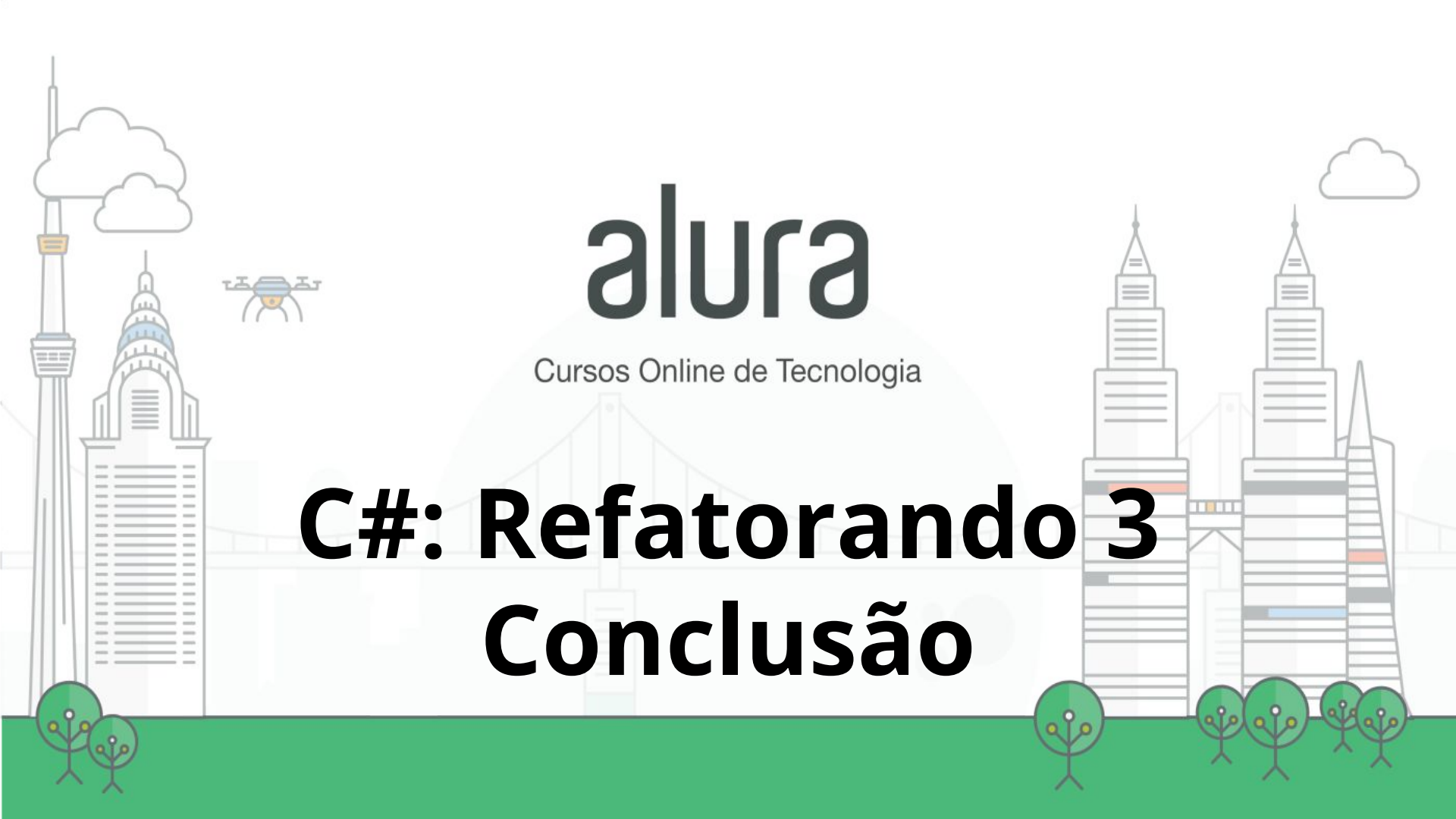
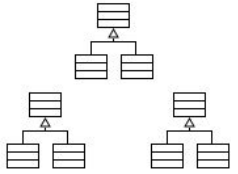


C#: Refatorando 3
Conclusão

Parte 3

Simplificar Chamadas



Generalização

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

Técnicas Para Simplificar Chamadas a Métodos

40) Renomear Método

Problema: O nome do método não explica o que ele faz OU existe método semelhante

Solução: Renomear o método

Refatorações similares

- Adicionar Parâmetro
- Remover Parâmetro

Elimina os odores

- Classes Alternativas com Interfaces Diferentes
- Comentários

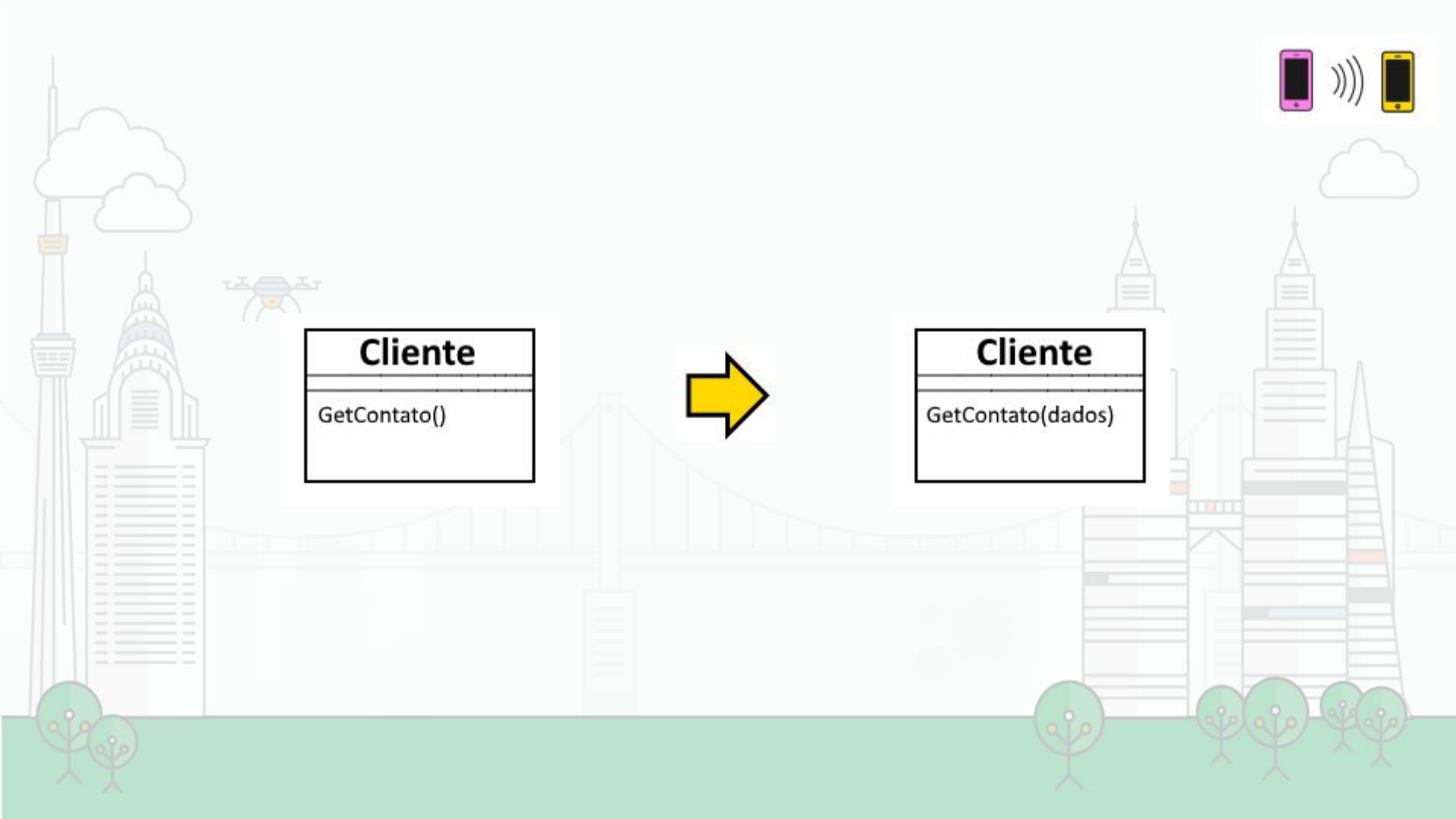
41) Adicionar Parâmetro

Problema: Um método não tem todos os dados de que precisa para fazer seu trabalho (mas o chamador tem esses dados!)

Solução: Crie parâmetros para fornecer os dados necessários ao método

Refatoração inversa
Remover Parâmetro

Auxilia outras refatorações
Introduzir Objeto-Parâmetro


Cliente
GetContato()
Cliente
GetContato(dados)

42) Remover Parâmetro

Problema: Um parâmetro não é usado dentro do corpo do método

Solução: Remova o parâmetro não utilizado

Refatoração inversa

Adicionar Parâmetro

Auxilia outras refatorações

Substituir Parâmetro por Chamada a Método

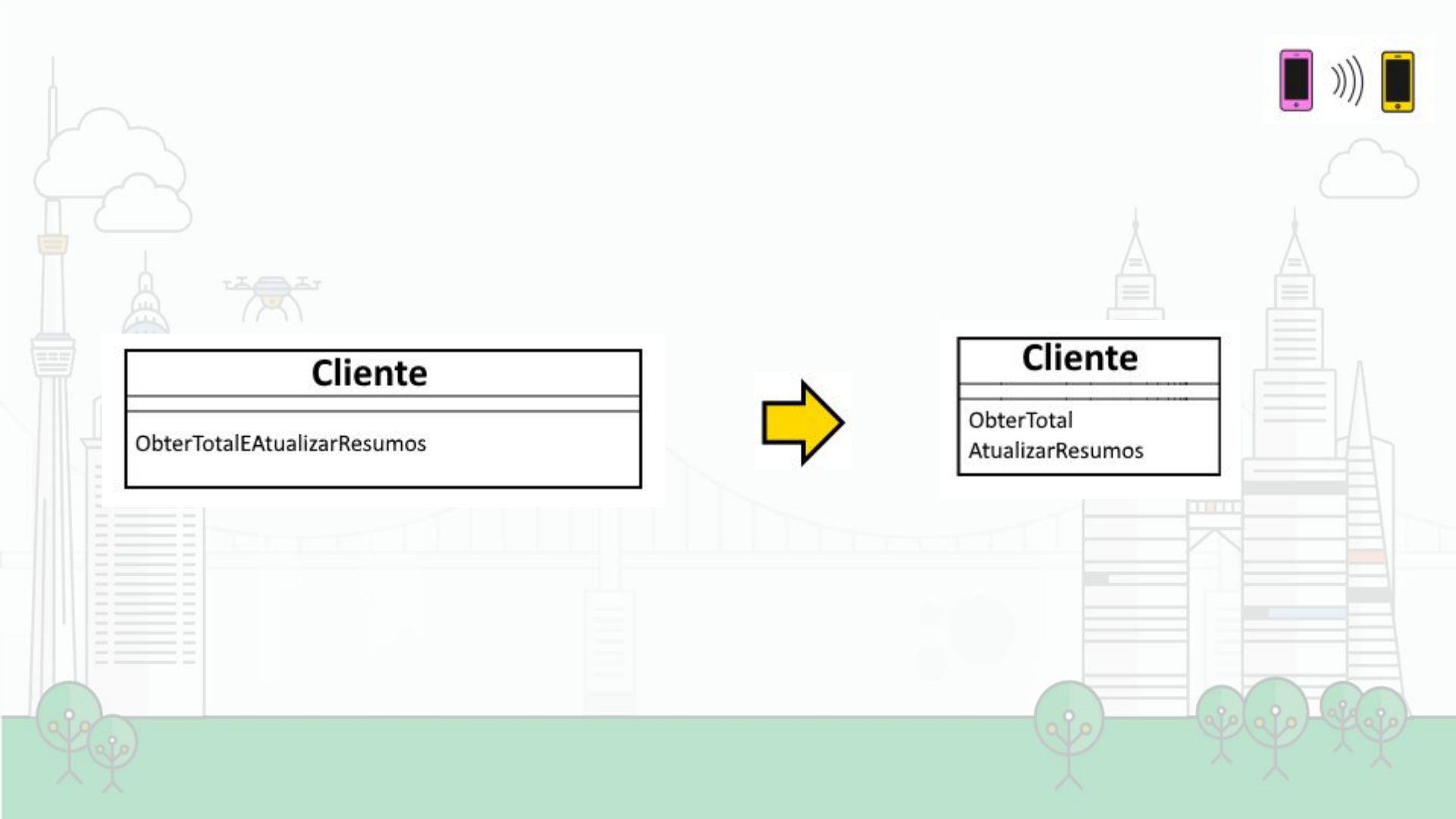
Elimina os odores

Generalidade Especulativa


Cliente
GetContato(dados)
Cliente
GetContato()

43) Separar Consulta do Modificador

Problema: Um método faz duas coisas: retorna dados mas também executa algum comando
Solução: Quebre o método em dois. Um método vai retornar dados e o outro vai executar comandos.
Auxilia outras refatorações
Substituir Variável Temporária por Consulta a Método


Cliente
ObterTotalEAtualizarResumos
Cliente
ObterTotal
AtualizarResumos

44) Parametrizar Método

Problema: Vários métodos executam ações similares que só diferem em valores, números ou operações

Solução: Combine esses métodos utilizando um parâmetro que passará os valores necessários

Refatoração inversa

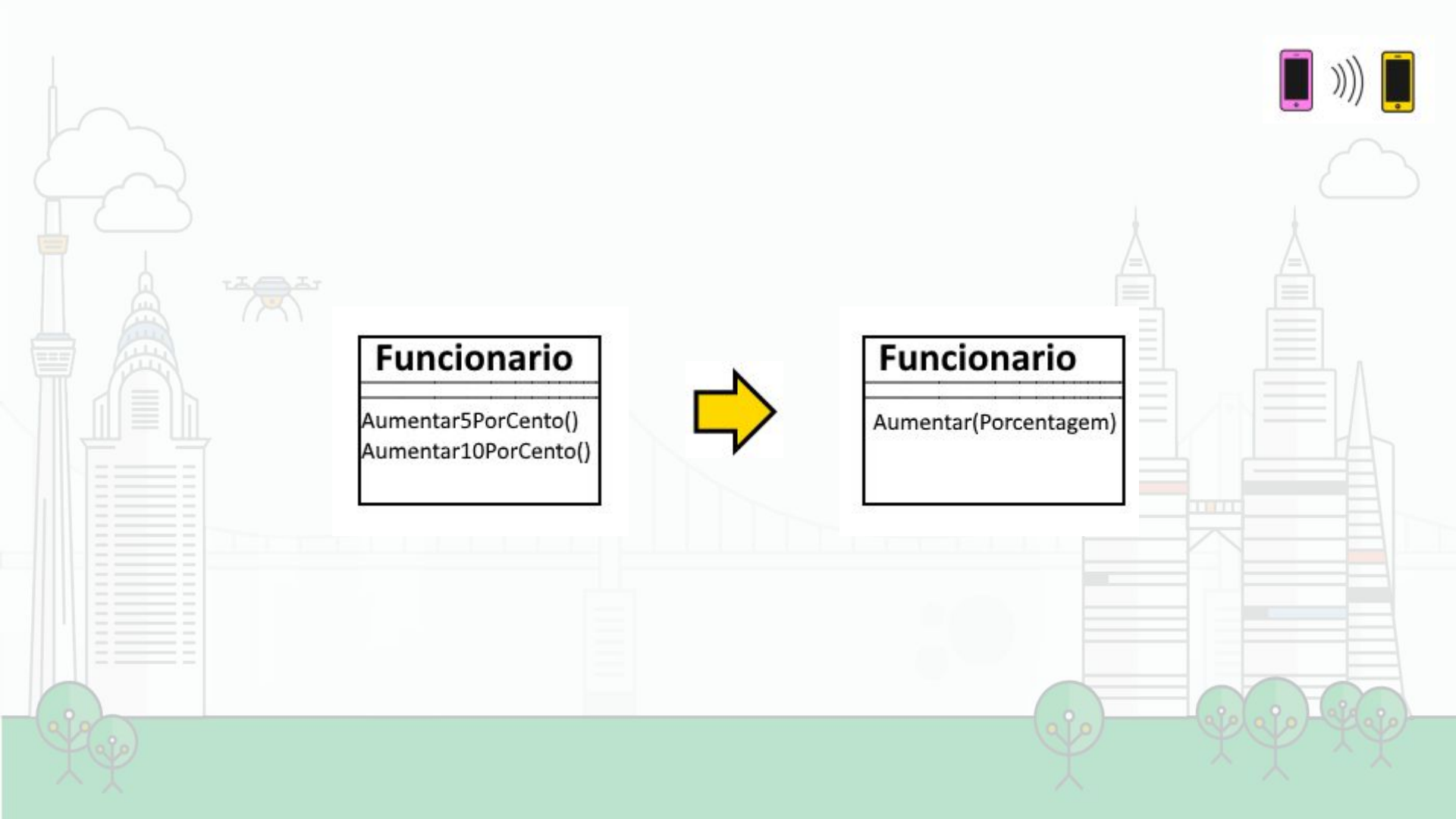
Substituir Parâmetro por Métodos Explícitos

Refatorações similares

- Extrair Método
- Formar Método Template

Elimina os odores

Código Duplicado


Funcionario
Aumentar5PorCento()
Aumentar10PorCento()
Funcionario
Aumentar(Porcentagem)

45) Substituir Parâmetro por Métodos Explícitos

Problema: Um método retorna valores diferentes dependendo do valor de um parâmetro

Solução: Extraia as partes do método em métodos distintos, cada um com sua lógica independente

Refatoração inversa

Parametrizar Método

Elimina os odores

- Comando Switch
- Método Longo

46) Preservar Objeto Inteiro

Problema: Você está obtendo valores de um objeto e em seguida passando esses valores para um método

Solução: Passe o objeto inteiro como parâmetro

Refatorações similares

- Introduzir Objeto-Parâmetro
- Substituir Parâmetro por Chamada a Método

Elimina os odores

- Obsessão por Primitivos
- Longa Lista de Parâmetros
- Método Longo
- Massa de Dados

47) Substituir Parâmetro por Chamada a Método

Problema: Fazer uma consulta a um método e usar o resultado como parâmetro de um outro método, sendo que este poderia chamar a consulta diretamente

Solução: Faça o segundo método chamar a consulta

48) Introduzir Objeto-Parâmetro

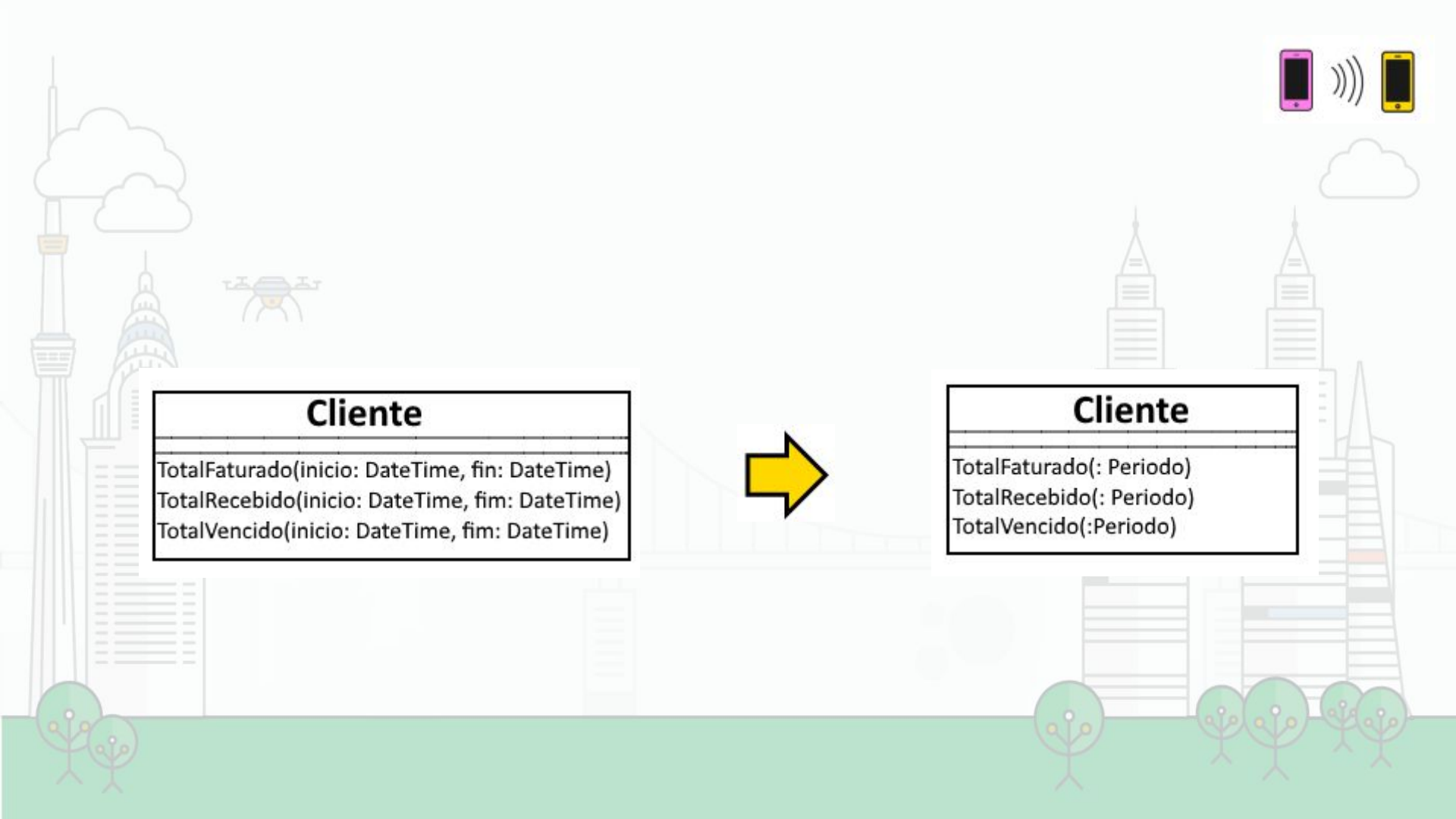
Problema: Métodos que exigem sempre o mesmo grupo de parâmetros

Solução: Substitua esses parâmetros por um objeto

Refatorações similares
Preservar Objeto Inteiro

Elimina os odores

- Longa Lista de Parâmetros
- Massa de Dados
- Obsessão por Primitivos
- Método Longo


Cliente
TotalFaturado(inicio: DateTime, fin: DateTime)
TotalRecebido(inicio: DateTime, fim: DateTime)
TotalVencido(inicio: DateTime, fim: DateTime)
Cliente
TotalFaturado(: Periodo)
TotalRecebido(: Periodo)
TotalVencido(:Periodo)

49) Remover Método Setter

Problema: O valor de um campo deveria ser definido somente durante sua criação, e nunca mais alterado
Solução: Remova métodos e setters que alteram o campo
Auxilia outras refatorações
Change Reference to Value

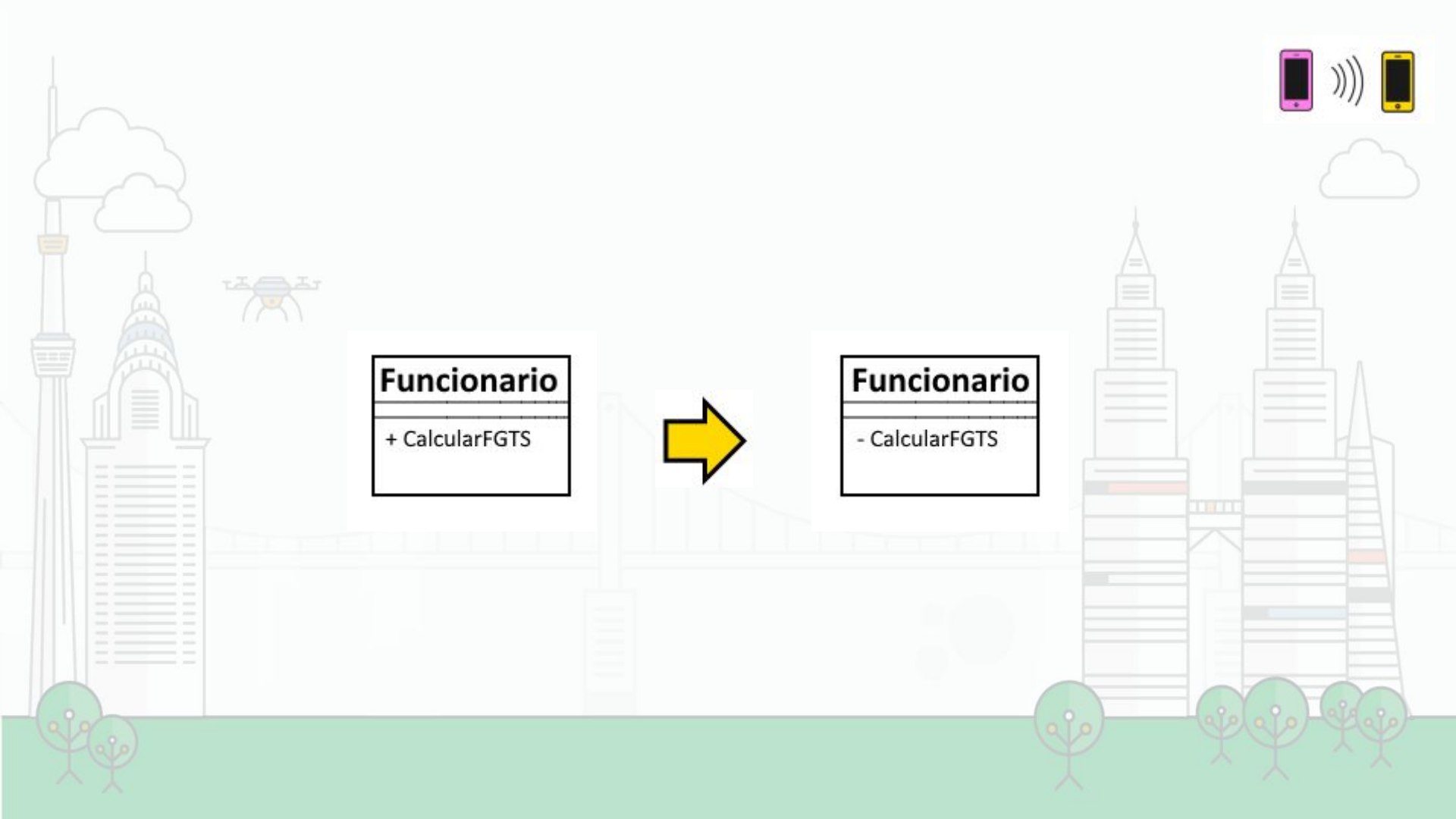
50) Ocultar Método

Problema: Um método não é usado por outras classes ou fora de sua hierarquia

Solução: Marque o método como private ou protected.

Elimina os odores

Classe de Dados


Funcionario
+ CalcularFGTS
Funcionario
- CalcularFGTS

51) Substituir Construtor por Método Factory

Problema: Você tem um construtor que realiza tarefas complexas

Solução: Crie um método factory e use-o no lugar das chamadas ao construtor

Implementa os padrões de projeto

Método Factory

52) Substituir Código de Erro por Exception

Problema: Um método retorna um valor especial para indicar um erro
Solução: Lance uma exceção no lugar de retornar o valor especial

53) Substituir Exceção por Teste

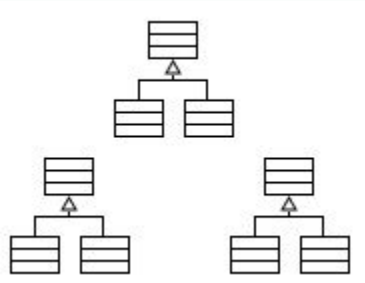
Problema: Em vez de fazer um simples teste, você está lançando uma exceção

Solução: Substitua a exceção por uma condição de teste

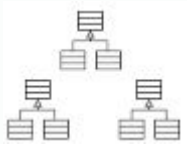
Refatorações similares

Substituir Código de Erro por Exceção

Técnicas de Generalização



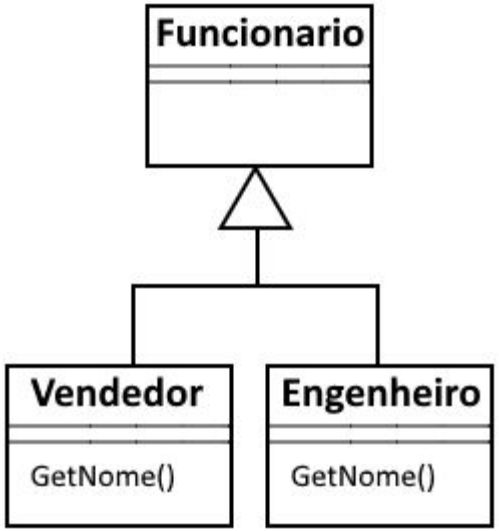
54) Subir Método

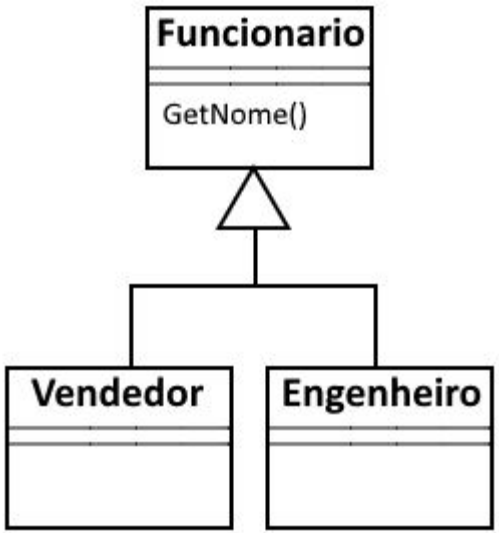


Problema: Duas classes possuem métodos que fazem a mesma coisa (ou quase)

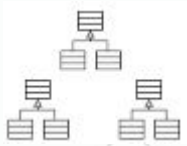
Solução: Remova os métodos das subclasses e mova-o para a superclasse

Refatoração inversa
Descer Método
Elimina os odores
Código Duplicado


Funcionario
Vendedor
GetNome()
Engenheiro
GetNome()


Funcionario
GetNome()
Vendedor
Engenheiro

55) Subir Corpo do Construtor



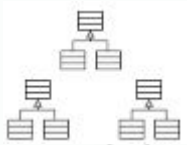
Problema: As subclasses possuem construtores quase idênticos

Solução: Crie um construtor na superclasse e mova o código comum para lá. Chame o construtor da superclasse a partir das subclasses

Elimina os odores

Código Duplicado

56) Descer Método



Problema: Comportamento implementado na superclasse, mas usado em apenas uma subclasse

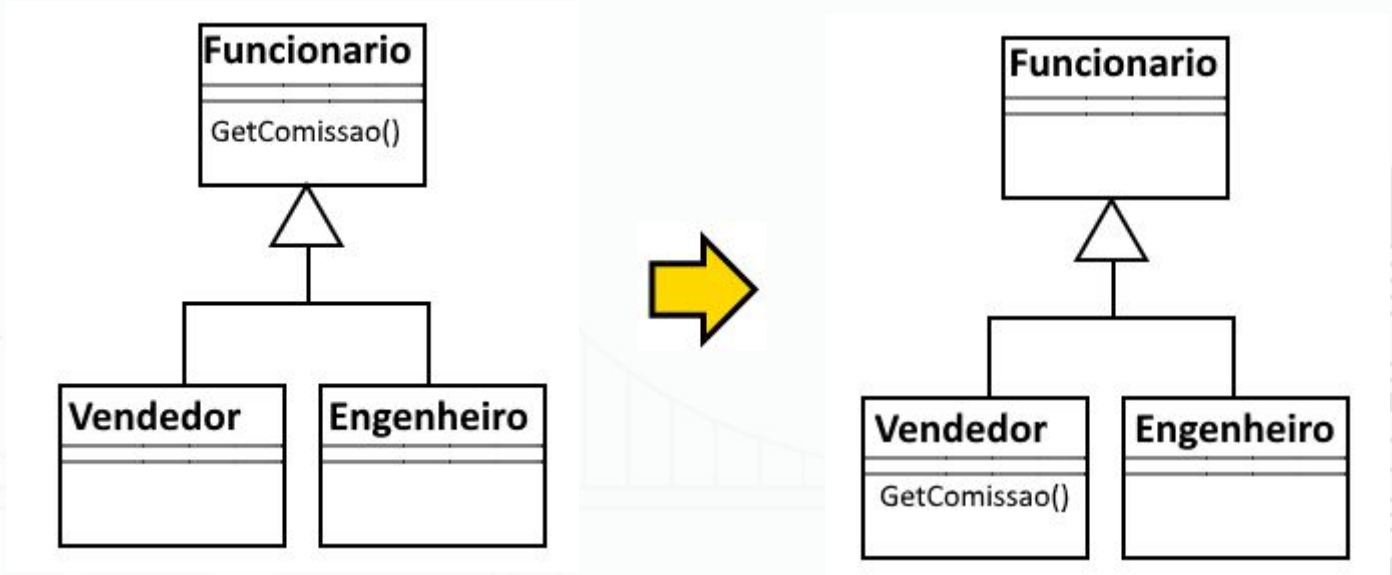
Solução: Mova este comportamento para a subclasse

Refatoração inversa

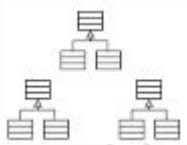
Subir Método

Elimina os odores

Herança Rejeitada


Funcionario
GetComissao()
Vendedor
Engenheiro
Funcionario
Vendedor
GetComissao()
Engenheiro

57) Descer Campo



Problema: Um campo é usado apenas em uma subclasse

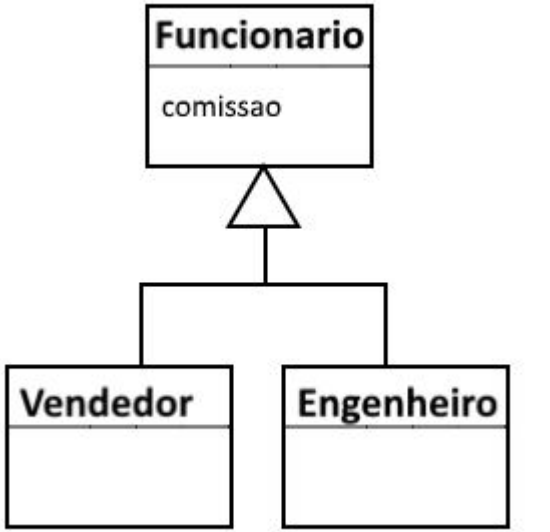
Solução: Mova este campo para essa subclasse

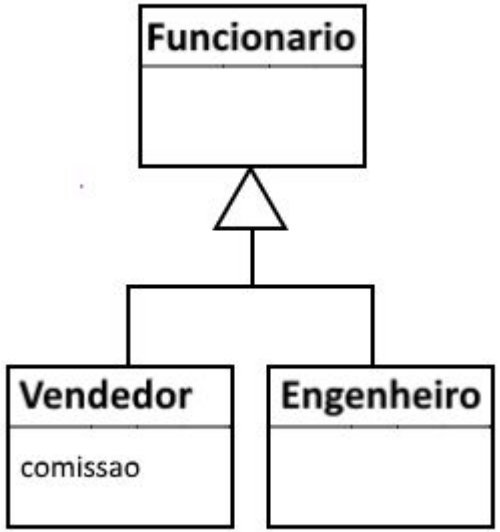
Refatoração inversa

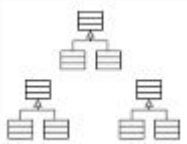
Subir Campo

Elimina os odores

Herança Rejeitada


Funcionario
comissao
Vendedor
Engenheiro


Funcionario
Vendedor
comissao
Engenheiro



58) Extrair Subclasse

Problema: Uma classe tem um conjunto de dados e comportamentos só usado em alguns casos

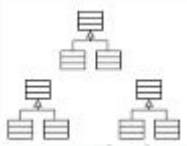
Solução: Crie uma subclasse para essas situações

Elimina os odores

Classe Grande


ItemServico
GetPrecoTotal()
GetPrecoUnitario()
GetFuncionario()
ItemServico
GetPrecoTotal()
GetPrecoUnitario()
MaoDeObra
GetPrecoUnitario()
GetFuncionario()

59) Extrair Superclasse



Problema: Duas ou mais classes possuem mesmos métodos ou dados

Solução: Crie uma superclasse e mova para lá os dados e comportamentos comuns das subclasses

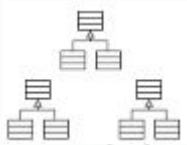
Refatorações similares

Extrair Interface

Elimina os odores

Código Duplicado

60) Extrair Interface



Problema: Código “cliente” usa a mesma interface de algumas classes

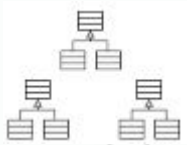
Solução: Extraia a interface comum dessas classes para uma interface

Refatorações similares

Extrair Superclasse


Funcionario
TemHabilidadeEspecial()
GetAvaliacao()
GetNome()
GetDepartamento()
«interface»
Faturavel
TemHabilidadeEspecial()
GetAvaliacao()
Funcionario
TemHabilidadeEspecial()
GetAvaliacao()
GetNome()
GetDepartamento()

61) Colapsar Hierarquia

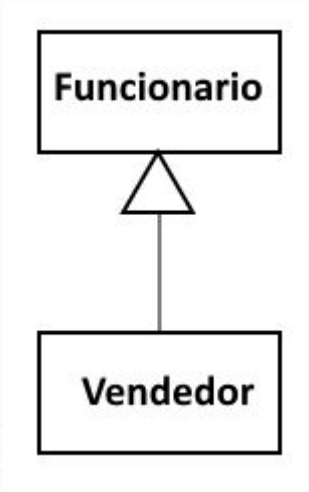


Problema: Subclasse e superclasse são praticamente iguais

Solução: Use somente uma das classes. Migre a diferença e remova a outra classe.

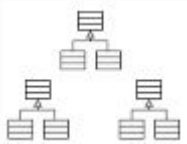
Elimina os odores

- Classe Preguiçosa
- Generalidade Especulativa


Funcionario
Vendedor


Funcionario

62) Formar Método Template



Problema: Duas ou mais subclasses possuem métodos diferentes, mas que executam um algoritmo na mesma ordem

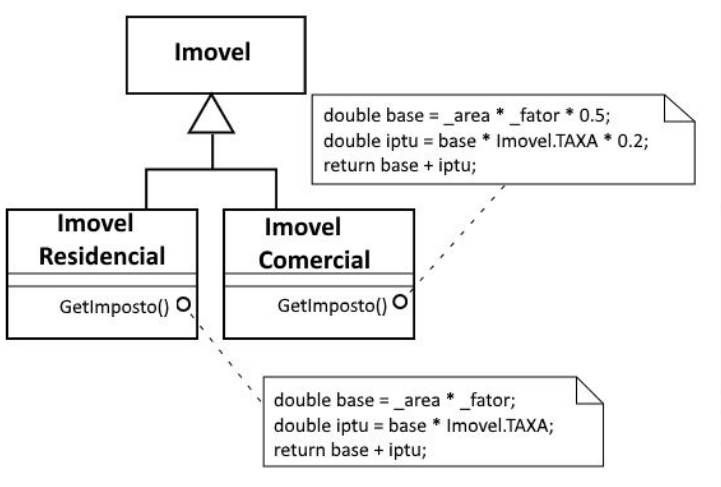
Solução: Crie um método na superclasse que contenha o “esqueleto” com a sequência de passos. Para cada passo, chame um método da subclasse.

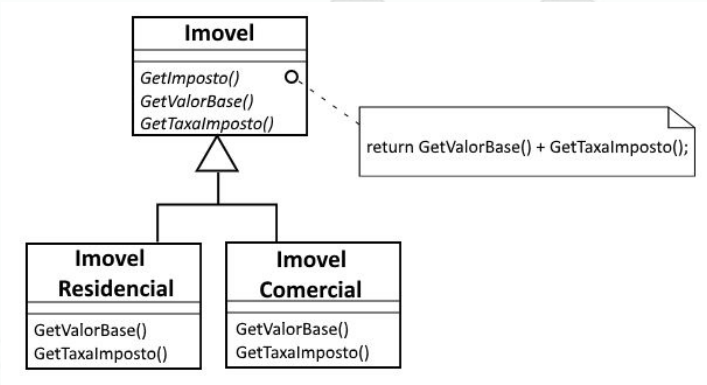
Implementa os padrões de projeto

Método Template

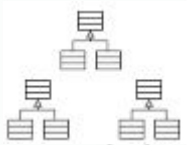
Elimina os odores

Código Duplicado

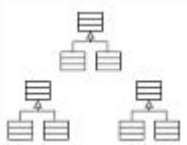

Imovel
double base = _area * _fator * 0.5;
double iptu = base * Imovel.TAXA * 0.2;
return base + iptu;
Imovel Residencial
GetImposto()
Imovel Comercial
GetImposto()
double base = _area * _fator;
double iptu = base * Imovel.TAXA;
return base + iptu;


Imovel
GetImposto()
GetValorBase()
GetTaxaImposto()
return GetValorBase() + GetTaxaImposto();
Imovel Residencial
GetValorBase()
GetTaxaImposto()
Imovel Comercial
GetValorBase()
GetTaxaImposto()

63) Substituir Herança por Delegação



Problema: Subclasse só usa uma parte dos dados ou comportamentos da superclasse (herança mal utilizada)
Solução: Utilize a superclasse como campo da subclasse. Chame os métodos e dados da superclasse. Elimine a herança.
Refatoração inversa
Substituir Delegação por Herança



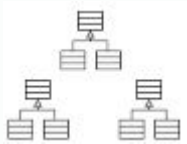
```
class Stack<T> : List<T>
{
    public void Push(T element)
    {
        Insert(0, element);
    }

    public T Pop()
    {
        var result = this[0];
        RemoveAt(0);
        return result;
    }
}
```

```
class Stack<T> : List<T>
{
    private List<T> elements = new List<T>();

    public void Push(T element)
    {
        elements.Insert(0, element);
    }

    public T Pop()
    {
        var result = elements[0];
        elements.RemoveAt(0);
        return result;
    }
}
```

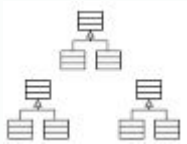


Para saber mais

Composition over inheritance

https://en.wikipedia.org/wiki/Composition_over_inheritance

64) Substituir Delegação por Herança

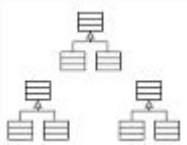


Problema: Uma classe tem um campo que referencia outra classe, e tem métodos simples, que fazem nada além de chamar os métodos dessa outra classe

Solução: Faça uma herança, remova o campo e passe a utilizar os métodos da superclasse

Elimina os odores

Intimidade Inapropriada



```
class Person
{
    public string Name { get; set; }

    public string LastName
    {
        get { return Name.Substring(Name.LastIndexOf(' ') + 1); }
    }

    public string PhoneNumber { get; set; }
}

class Employee
{
    private Person person = new Person();

    public string Name
    {
        get { return person.Name; }
        set { person.Name = value; }
    }

    public string PhoneNumber
    {
        get { return person.PhoneNumber; }
        set { person.PhoneNumber = value; }
    }
```

```
class Person
{
    public string Name { get; set; }

    public string LastName
    {
        get { return Name.Substring(Name.LastIndexOf(' ') + 1); }
    }

    public string PhoneNumber { get; set; }
}

class Employee : Person
{
    public override string ToString()
    {
        return "Emp: " + LastName;
    }
}
```