Reutilização de Soluções com *Patterns* e *Framework* na Camada de Negócio

Orientando Marcio Carlos Grott

Orientador Marcel Hugo

Roteiro de Apresentação

- Introdução
- Arquitetura de software
- Reutilização
- *Patterns*
- *Framework*
- Impostos
- Implementação do *Framework* e Protótipo

Introdução

- Aumento da utilização da OO;
- Requisitos de software mais complexos;
- Problemas semelhantes em sua essência;
- Semelhanças na resolução conduzem a padrões;
- Padrões formam *frameworks*.

Introdução

Objetivos do Trabalho

Implementar um *framework* para cálculo de impostos em projetos de automação comercial utilizando Padrões de Projeto.

- Exemplificar o uso do *pattern* e *frameworks;*
- Criar um *framework* com os *patterns* selecionados;
- Desenvolver um material que ajudará os arquitetos de sistemas a decidirem qual a melhor solução a ser tomada no desenvolvimento de um projeto.

Arquitetura de Software

A palavra arquitetura está associada à arte e ciência de projetar e construir, levando em consideração os métodos, ferramentas e padronizações utilizadas no projeto e na construção.

Arquitetura de Aplicações define a organização estática dos sistemas aplicativos em um alto nível.

Arquitetura de Software

- Inicialmente a arquitetura monolítica;
- Evoluindo para cliente/servidor;
- Atualmente sistemas construídos em várias camadas.

Arquitetura de software
Multicamadas (*multi-tier*)

- Divisão do sistema em camadas lógicas: interface, regras de negócio e acesso a banco de dados;
- Isolamento da aplicação em componentes facilitando a reutilização;
- Distribuição das camadas em diferentes nós físicos;
- Alocação de desenvolvedores para construção de camadas específicas.

Arquitetura de software
Multicamadas (*multi-tier*)

Interface

Responsável pela apresentação e recepção de dados do usuário.

Acesso ao banco de dados

Responsável pela persistência dos dados da Camada de Regra de Negócio em um ou mais bancos de dados.

Arquitetura de software
Multicamadas (*multi-tier*) Regra de Negócio

Camada de Regra de Negócio deve conter conhecimento pertinente ao negócio abordado pela aplicação.

Usadas para capturar e implementar uma lógica de negócio precisa em termos de processos, procedimentos e sistemas de informação nas organizações.

Arquitetura de software
Multicamadas (*multi-tier*) Regra de Negócio

Perkins (2003) relaciona o que regras de negócio podem ser:

- definições de termos do negócio;
- restrições de integridade de dados;
- funções matemáticas;
- inferências lógicas;
- seqüências de processo;
- relacionamentos entre fatos do negócio.

Arquitetura de software
Multicamadas (*multi-tier*)

União de Tecnologia de Framework e Regra de Negócio

Há uma tendência na união das tecnologias pela compatibilidade;

Necessidades das organizações em aprimorar o processo de desenvolvimento

A não adoção da tecnologia vem em parte da não padronização dessa metodologia.

Maiores níveis de reutilização.

Reutilização

Grande número de soluções implementadas todos os dias.

Objetivo e reaproveitar módulos de projetos que já tenham sido desenvolvidos, testados e implantados anteriormente com sucesso.

Sistemas não somente são feitos de código fonte; portanto, deve-se ser capaz de reutilizar mais do que código.

Reutilização

Tipos de Reutilização

- Código;
- Herança;
- *Templates*;
- Componentes;
- *Frameworks*;
- Artefatos;
- *Patterns*;
- Componentes de domínio.

Reutilização

Vantagens

- Maior confiabilidade;
- Redução dos riscos de processo;
- Uso efetivo de especialistas;
- Conformidade com padrões;
- Desenvolvimento acelerado.

Reutilização

Desvantagens

- Aumento nos custos de manutenção;
- Falta de ferramentas de apoio;
- Síndrome do "não-foi-inventado-aqui";
- Manutenção de uma biblioteca de componentes;
- Encontrar e adaptar componentes reutilizáveis.

Patterns

- Aumento da capacidade computacional;
- Sensação de reinvenção;
- Facilitar a comunicação do conhecimento entre entidades e indivíduos;
- Partilhar experiências e soluções.

Patterns

Definição

Cada padrão descreve um problema que ocorre repetidas vezes em nosso ambiente, e então descreve o núcleo da solução para aquele problema, de tal maneira que se pode usar essa solução milhões de vezes sem nunca fazê-lá da mesma forma duas vezes segundo Christopher Alexander (apud GAMMA, 2000).

Patterns
Categorias

- Padrões de Processo;
- Padrões Arquiteturais;
- Padrões de padrões;
- Padrões de projeto;
- Padrões de programação;
- Padrões persistência;
- Padrões de Hipertexto.

Patterns
Processo de aprendizagem

- Aceitação da importância;
- Reorganização para poder utilizar;
- Divulgação dos detalhes suficientes para poder utilizar;

Patterns

Componentes de um padrão

- Nome referencia concisa e significativa;
- Problema onde se deve aplicar e seus objetivos;
- Contexto pré-condição em que o problema e sua solução costumam ocorrer;
- Forças descrições dos impactos, influências e restrições;
- Solução conjunto de relacionamentos estáticos, responsabilidades, colaborações e regras

Patterns

Componentes de um padrão

- Exemplo ilustração da utilização;
- Resultado do contexto estado ou configuração do sistema após a aplicação;
- Conseqüências análises das vantagens e desvantagens;
- Relacionamento com outros padrões;
- Aplicações conhecidas

Patterns
AntiPadrões

Antipadrões representam uma "lição aprendida", ao contrário dos padrões, que representam a "melhor prática".

Descreve com ir de uma solução ruim para uma solução boa.

Patterns
Catálogos de Padrões

Subdivide os padrões num pequeno número de categorias segundo Gamma (2000).

		Propósito		
		1. Criação	2. Estrutura	3. Comportamento
Escopo	Classe	Factory Method	Class Adapter	Interpreter Template Method
	Objeto	Abstract Factory Builder Prototype Singleton	Object Adapter Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Patterns

Catálogos de Padrões

Catálogo de Patterns segundo Mestker(2002).

Intenção	*Padrões*
1. Interfaces	*Adapter, Facade, Composite, Bridge*
2. Responsabilidade	*Singleton, Observer, Mediator, Proxy, Chain of Responsibility, Flyweight*
3. Construção	*Builder, Factory Method, Abstract Factory, Prototype, Memento*
4. Operações	*Template Method, State, Strategy, Command, Interpreter*
5. Extensões	*Decorator, Iterator, Visitor*

Patterns
Processo de desenvolvimento

Complementam os métodos de análise e projetos e independentes do problema.

- Utilizar o método preferido;
- Utilizar um sistema de padrões adequado;
- Encontrar o padrão caso não exista;
- Caso não encontra um padrão, utilizar as diretrizes do processo.

Patterns
Projeto de Software

- Identificar;
- Selecionar;
- Verificar;
- Instanciar;
- Avaliar.

Patterns
Vantagens

- Linguagem comum;
- Reduzem a complexidade;
- Independência de SO, linguagem de programação.

Patterns
Desvantagens

- Não suportam reutilização rápida de código fonte;
- Dificuldade de encontrar um *pattern* que se adapte facilmente ao problema;
- Nem sempre é a melhor solução.

Framework

É o projeto de um conjunto de objetos que colaboram entre si para execução de um conjunto de responsabilidades.

Conjunto de classes abstratas de um domínio específico modelando a arquitetura principal da aplicação.

Framework
Inversão de controle

Em uma biblioteca convencional escreve-se o corpo da aplicação e chama-se as rotinas que se deseja reutilizar.

No *framework* reutiliza-se o corpo principal e desenvolve-se as rotinas para chama-las.

Framework
Classificação

Frameworks horizontais são voltados para implementações voltadas a comunicação de dados, interface e gerenciamento de dados.

Frameworks verticais são voltados para o domínio específico de aplicação.

Framework
Classificação

Framework caixa branca o usuário pode visualizar como foi implementado a partir de classes já existentes.

Framework de caixa preta o usuário não é totalmente livre para criar ou alterar funcionalidades a partir da criação de classes.

Framework

Desenvolvimento de aplicações a partir de um framework

- Reutilizar em diferentes aplicações;
- Entendimento do sistema, particularidades, adaptação das estruturas;
- Centenas de classes podem estar contidas em um *framework.*

Impostos

- ICMS;
- IPI;
- Retenção de Micro Empresa;
- Base de substituição;
- Margem de substituição;
- Cliente Especial;
- Retenção para estados diferentes alíquotas de ICMS.

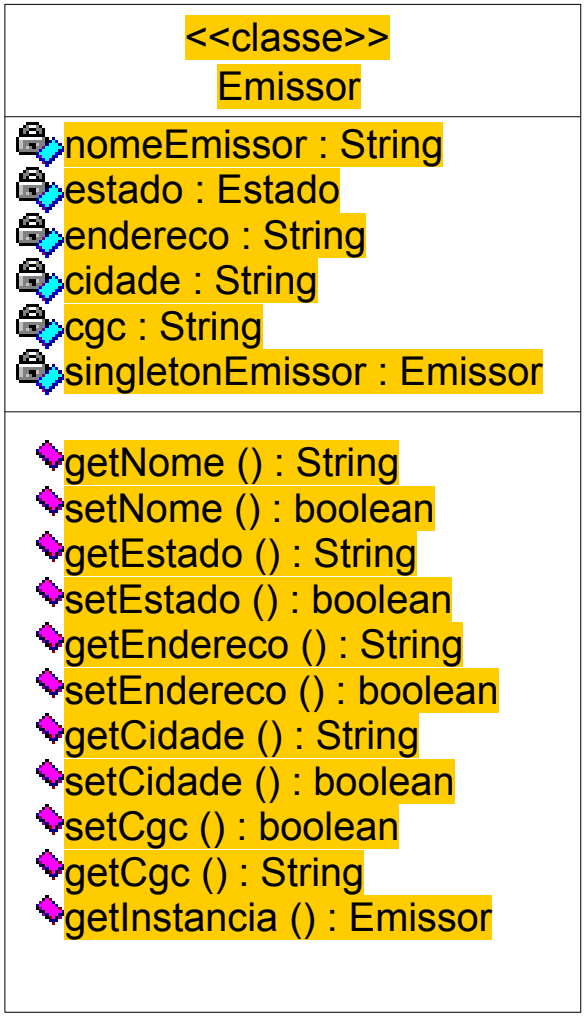
Desenvolvimento do Framework

Modelagem abrangente, concisa e específica do domínio do problema.

Aplicação dos *Patterns* extraídos do catálogo de padrões do Gamma (2000).

Desenvolvimento do Framework *Singleton*

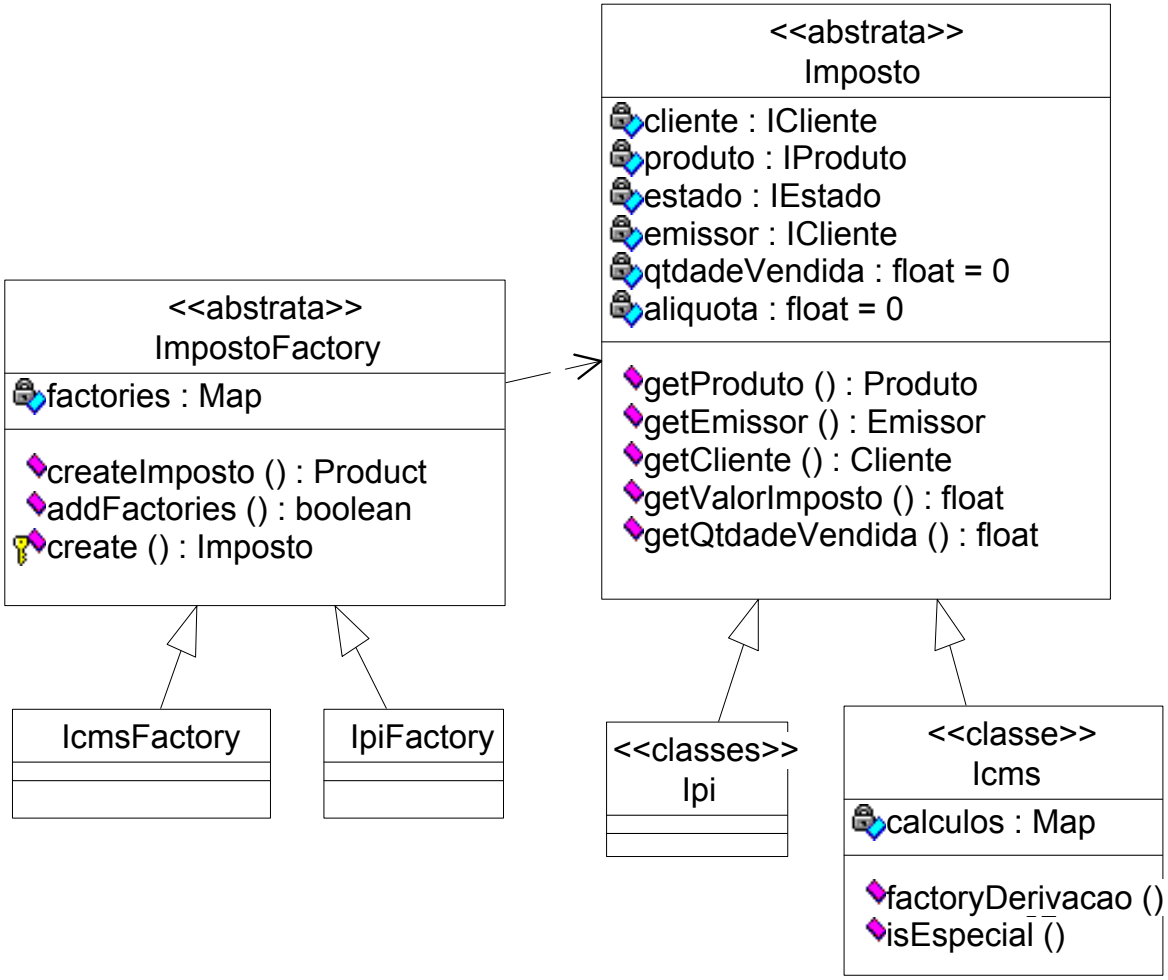
Garantir somente uma instância da classe e um único ponto de acesso.



Desenvolvimento do Framework
Factory Method

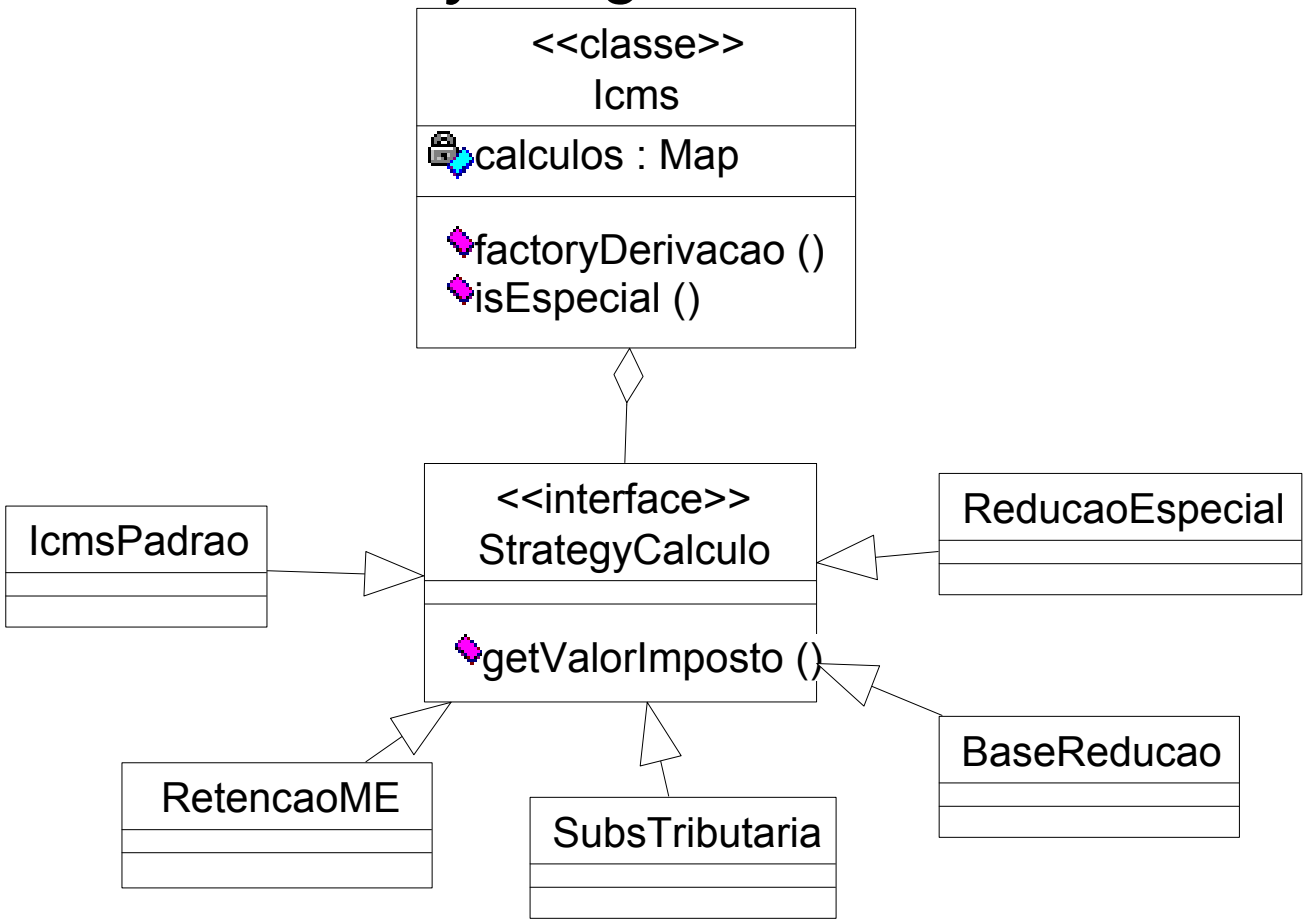
Interface para instanciação de objetos que mantém isoladas as classes concretas usadas na requisição da criação destes objetos.

Criação de famílias de objetos.

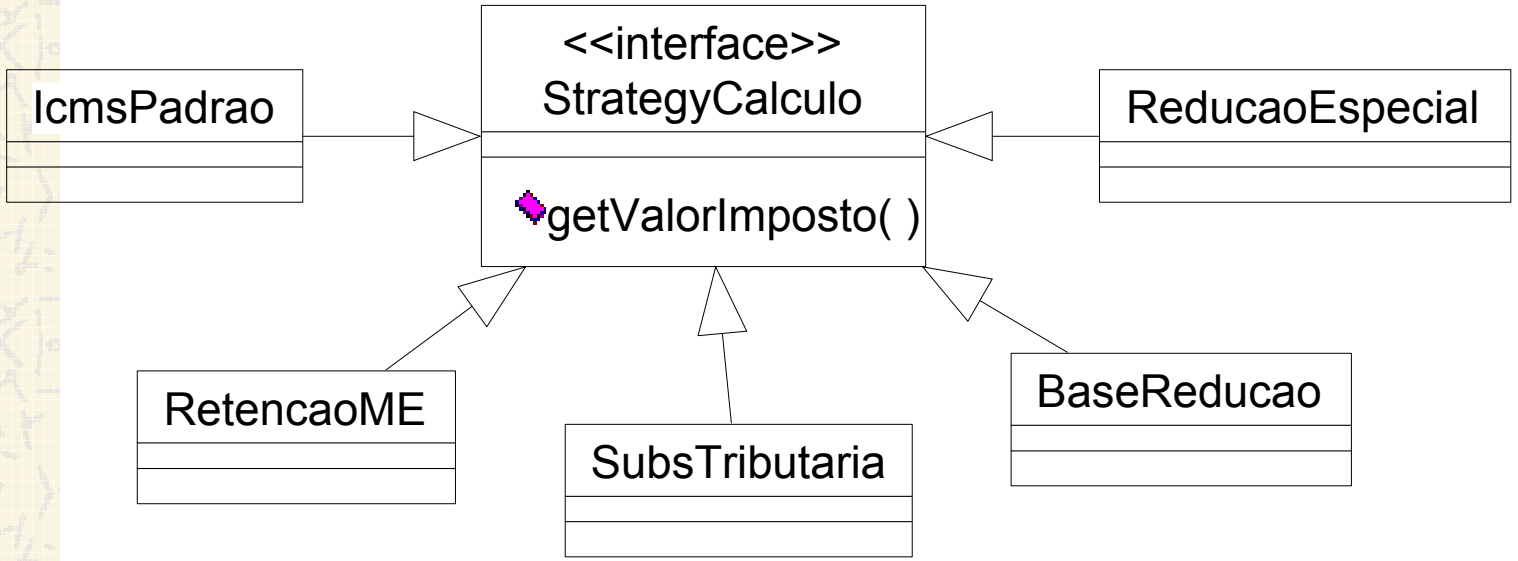


Desenvolvimento do Framework *Flyweight*

Suportar grande quantidade de objetos de granularidade fina onde o custo de armazenamento é alto devido à grande quantidade de objetos que podem ser agrupados.



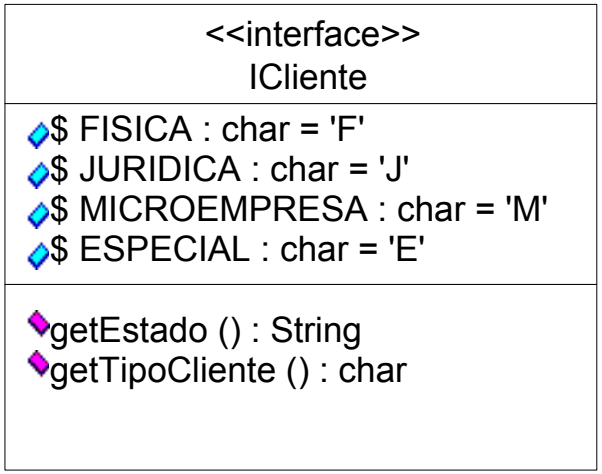
Desenvolvimento do Framework Strategy

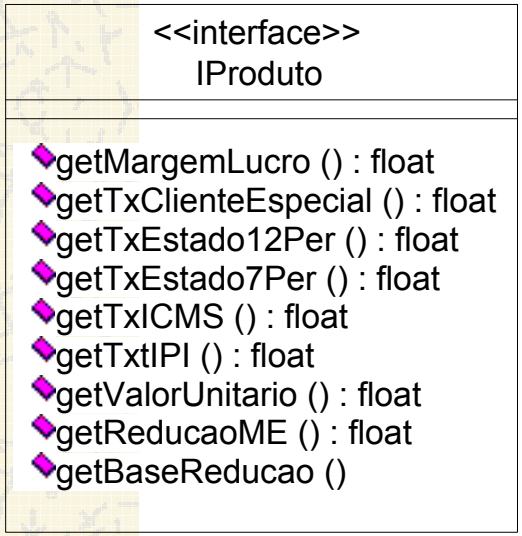


Definição de uma família de algoritmos, encapsula cada um e faz com que eles possam ser permutáveis.

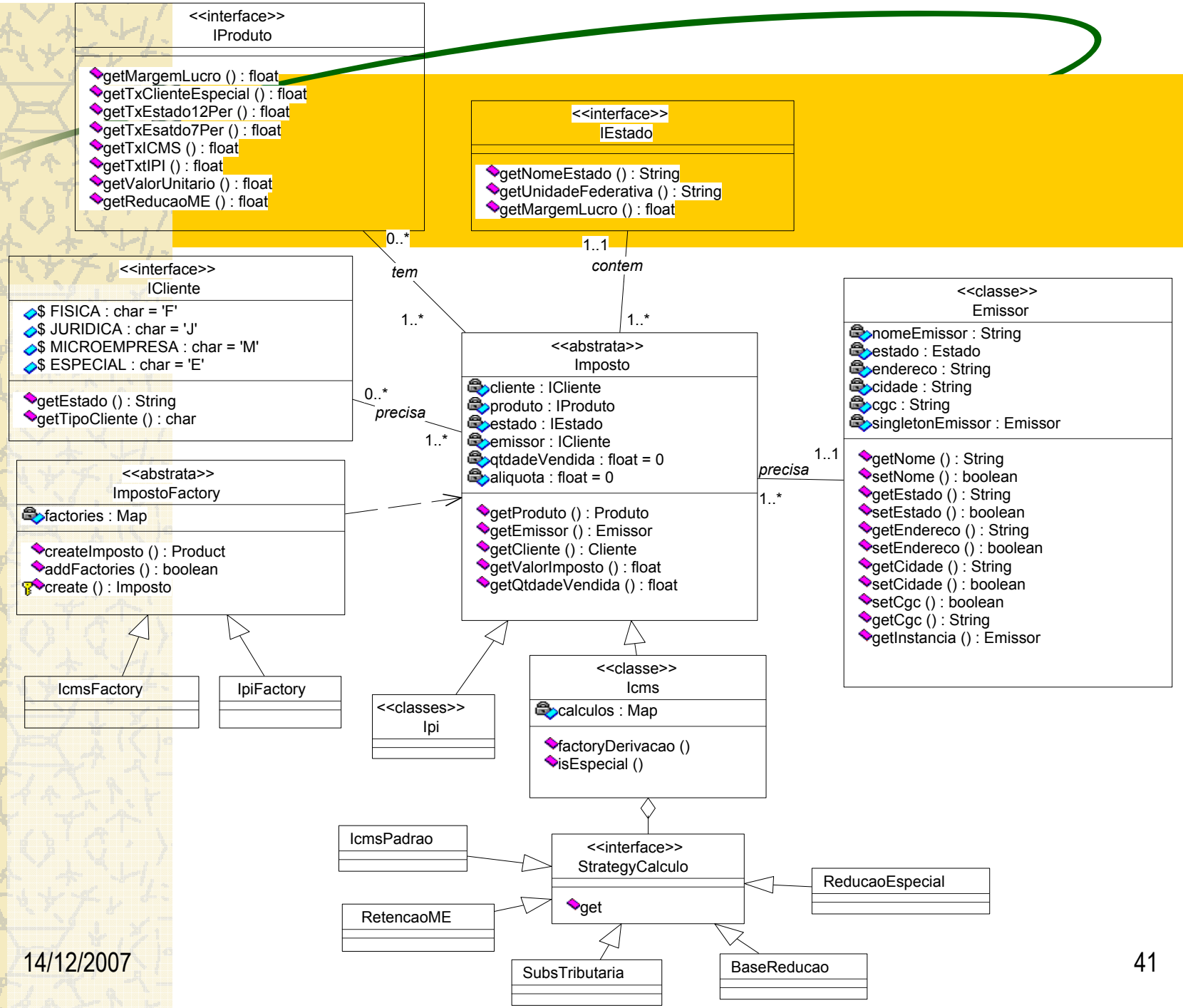
Desenvolvimento do Framework
Demais interfaces

Inteface `ICliente` responsável por implementar o tipo de Cliente.



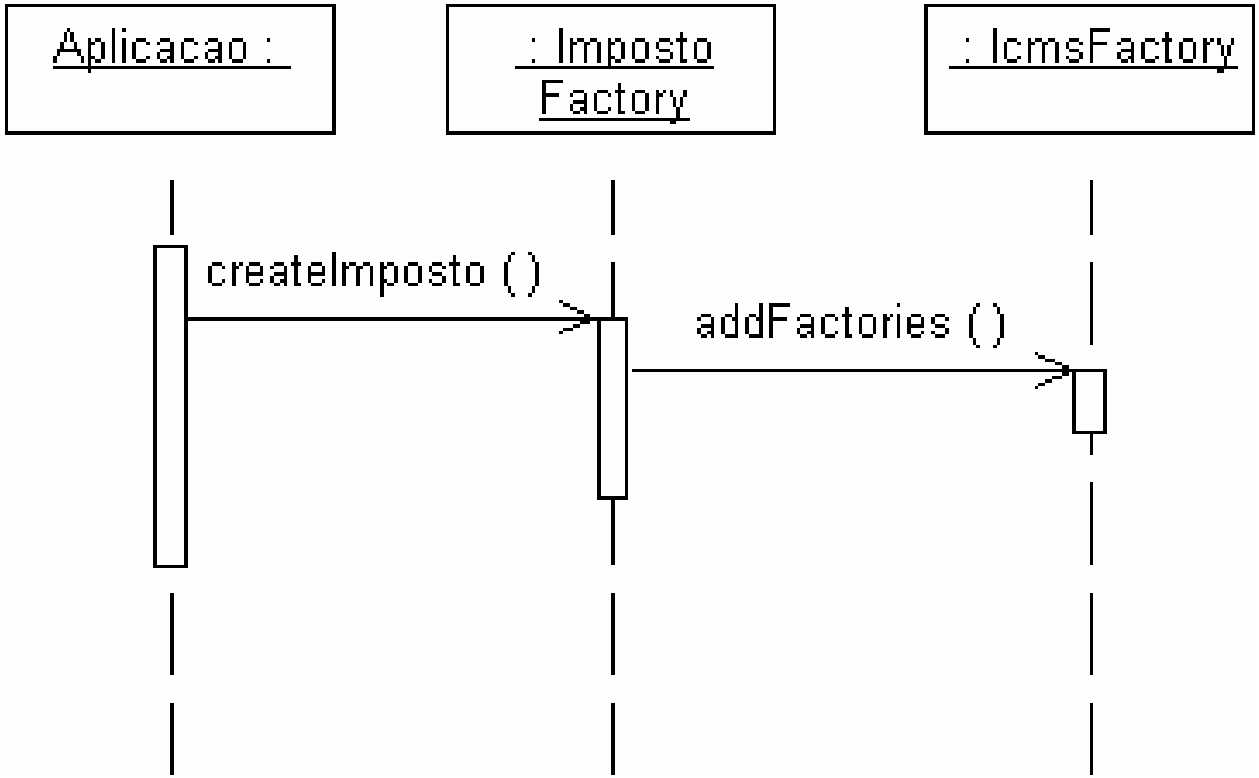


Inteface `IProduto` responsável por implementar os métodos de acesso as taxa de impostos.


<<interface>>
IProduto
getMargemLucro () : float
getTxClienteEspecial () : float
getTxEstado12Per () : float
getTxEsatdo7Per () : float
getTxICMS () : float
getTxtIPI () : float
getValorUnitario () : float
getReducaoME () : float
<<interface>>
IEstado
getNomeEstado () : String
getUnidadeFederativa () : String
getMargemLucro () : float
0..*
tem
1..*
1..1
contem
1..*
<<interface>>
ICliente
$ FISICA : char = 'F'
$ JURIDICA : char = 'J'
$ MICROEMPRESA : char = 'M'
$ ESPECIAL : char = 'E'
getEstado () : String
getTipoCliente () : char
0..*
precisa
1..*
<<abstrata>>
Imposto
cliente : ICliente
produto : IProduto
estado : IEstado
emissor : ICliente
qtdadeVendida : float = 0
aliquota : float = 0
getProduto () : Produto
getEmissor () : Emissor
getCliente () : Cliente
getValorImposto () : float
getQtdadeVendida () : float
<<classe>>
Emissor
nomeEmissor : String
estado : Estado
endereco : String
cidade : String
cgc : String
singletonEmissor : Emissor
getNome () : String
setNome () : boolean
getEstado () : String
setEstado () : boolean
getEndereco () : String
setEndereco () : boolean
getCidade () : String
setCidade () : boolean
setCgc () : boolean
getCgc () : String
getInstancia () : Emissor
1..1
precisa
1..*
<<abstrata>>
ImpostoFactory
factories : Map
createImposto () : Product
addFactories () : boolean
create () : Imposto
IcmsFactory
IpiFactory
<<classes>>
Ipi
<<classe>>
Icms
calculos : Map
factoryDerivacao ()
isEspecial ()
IcmsPadrao
<<interface>>
StrategyCalculo
get
ReducaoEspecial
RetencaoME
SubsTributaria
BaseReducao

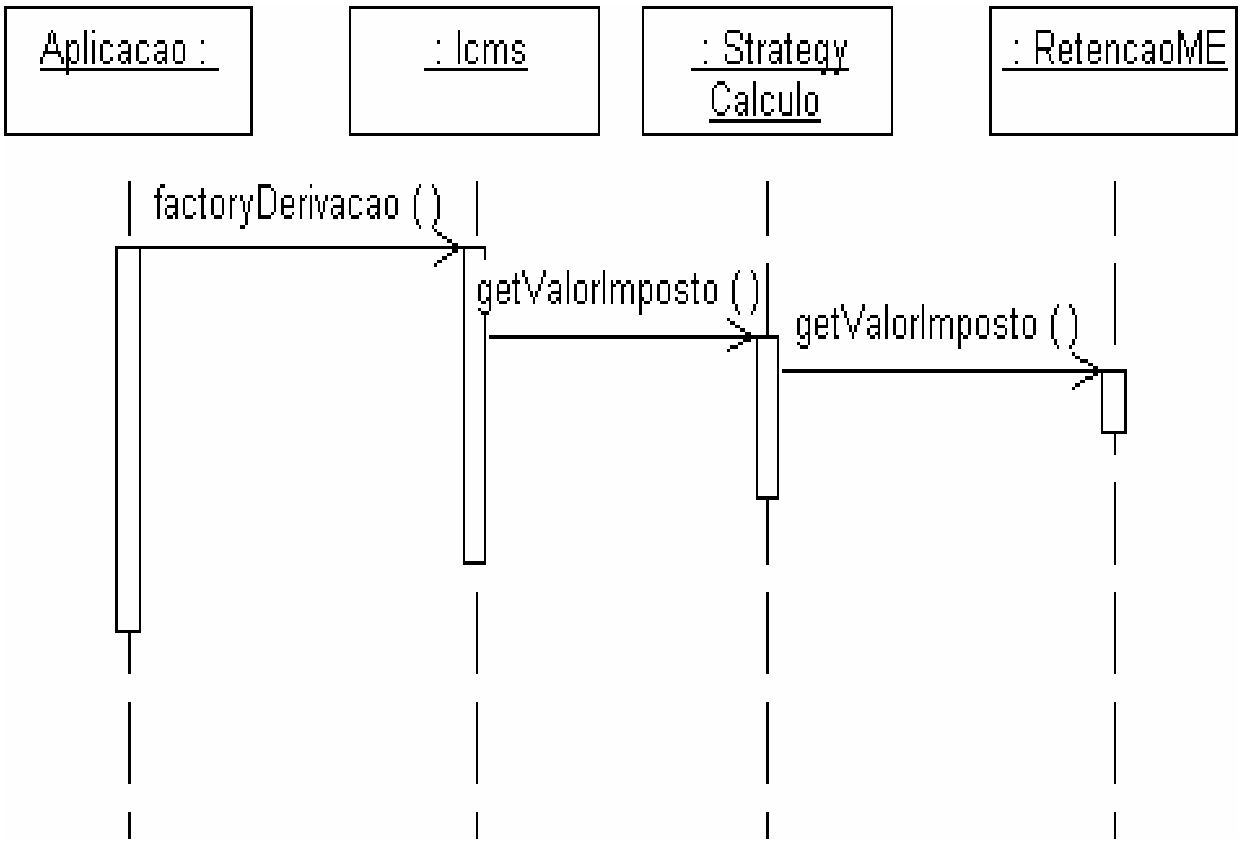
Desenvolvimento do Framework

Diagrama de seqüência demonstrando a criação de um imposto.



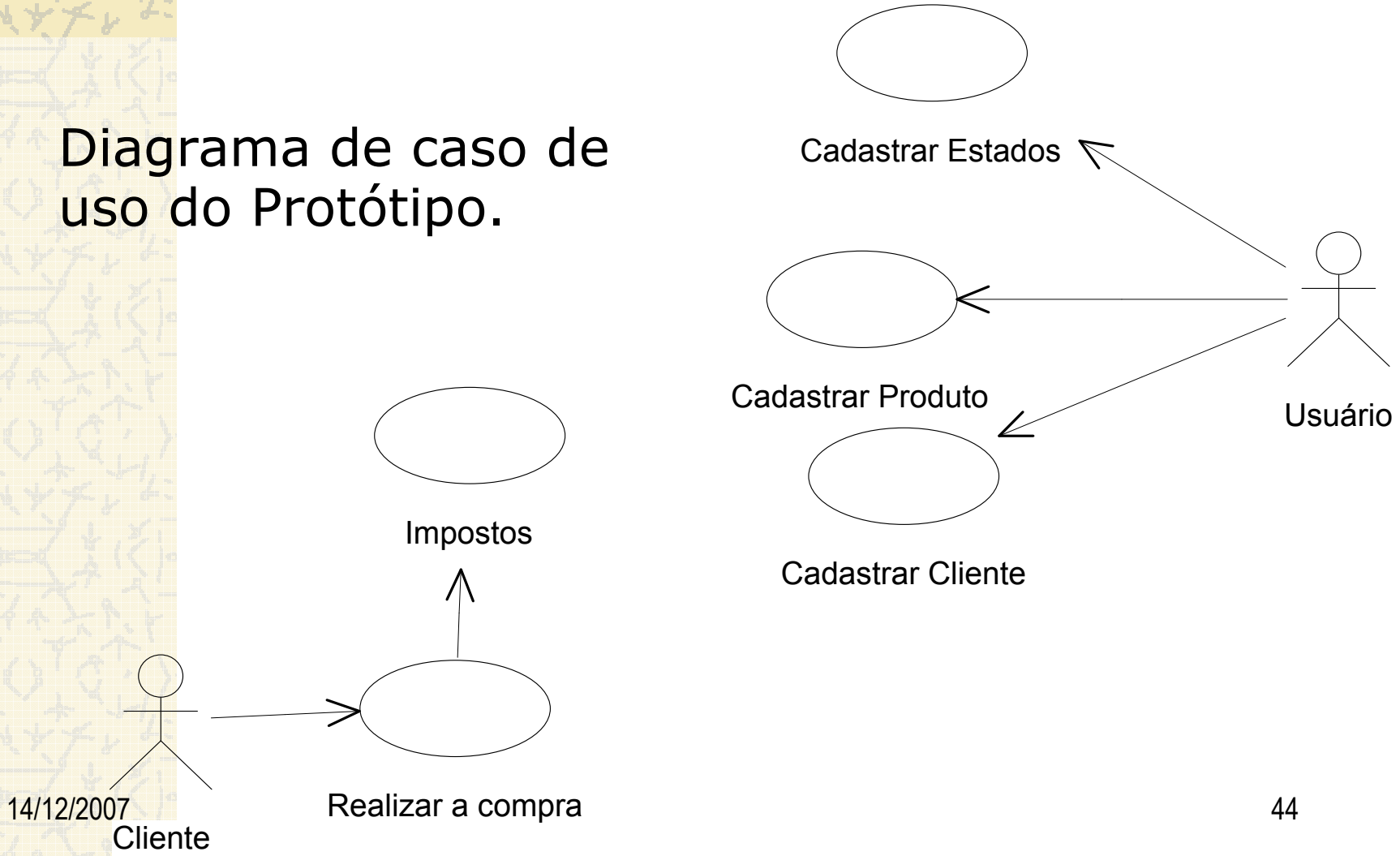
Desenvolvimento do *Framework*

Diagrama de seqüência do cálculo de um imposto.



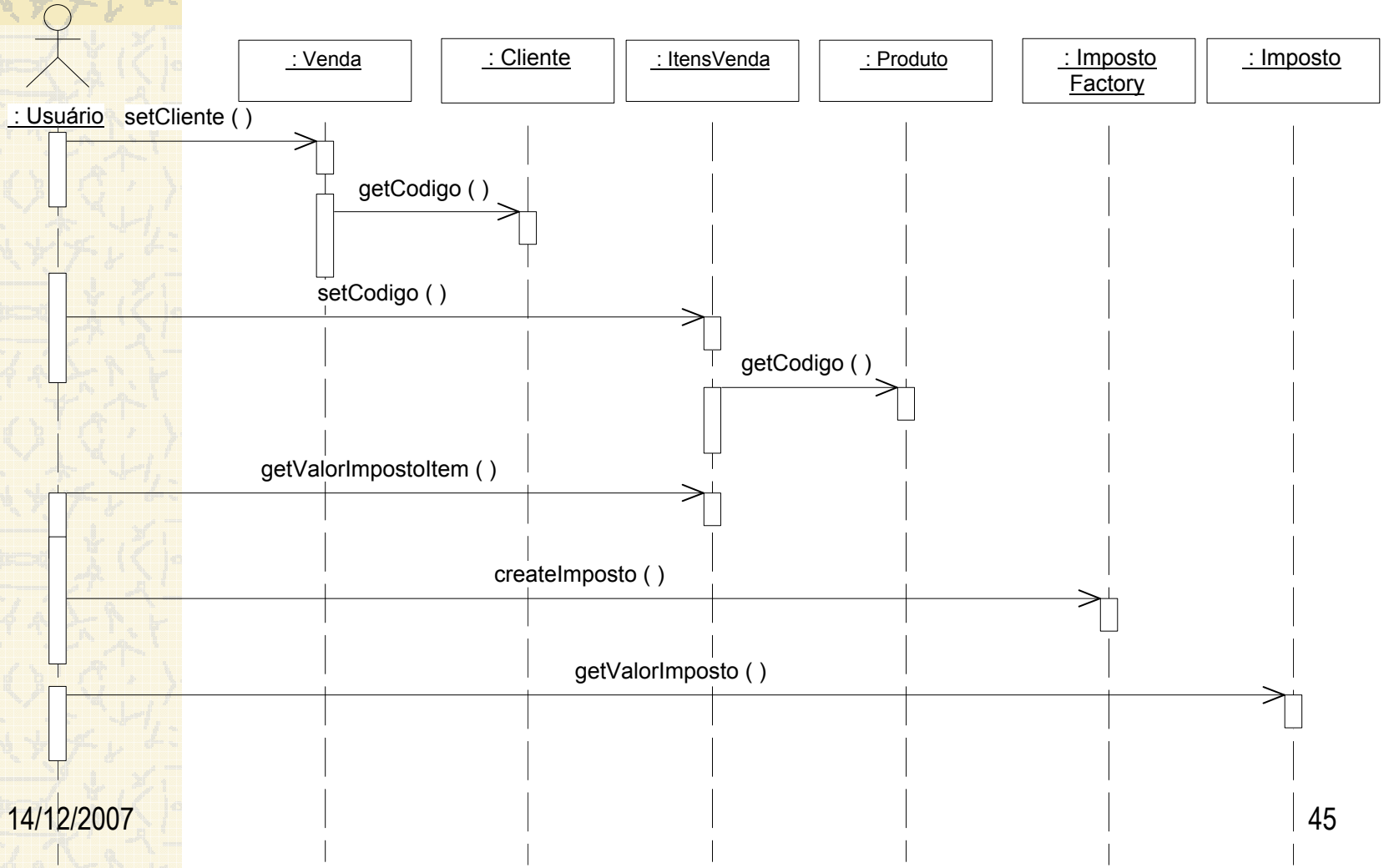
Implementação do Protótipo

Diagrama de caso de uso do Protótipo.

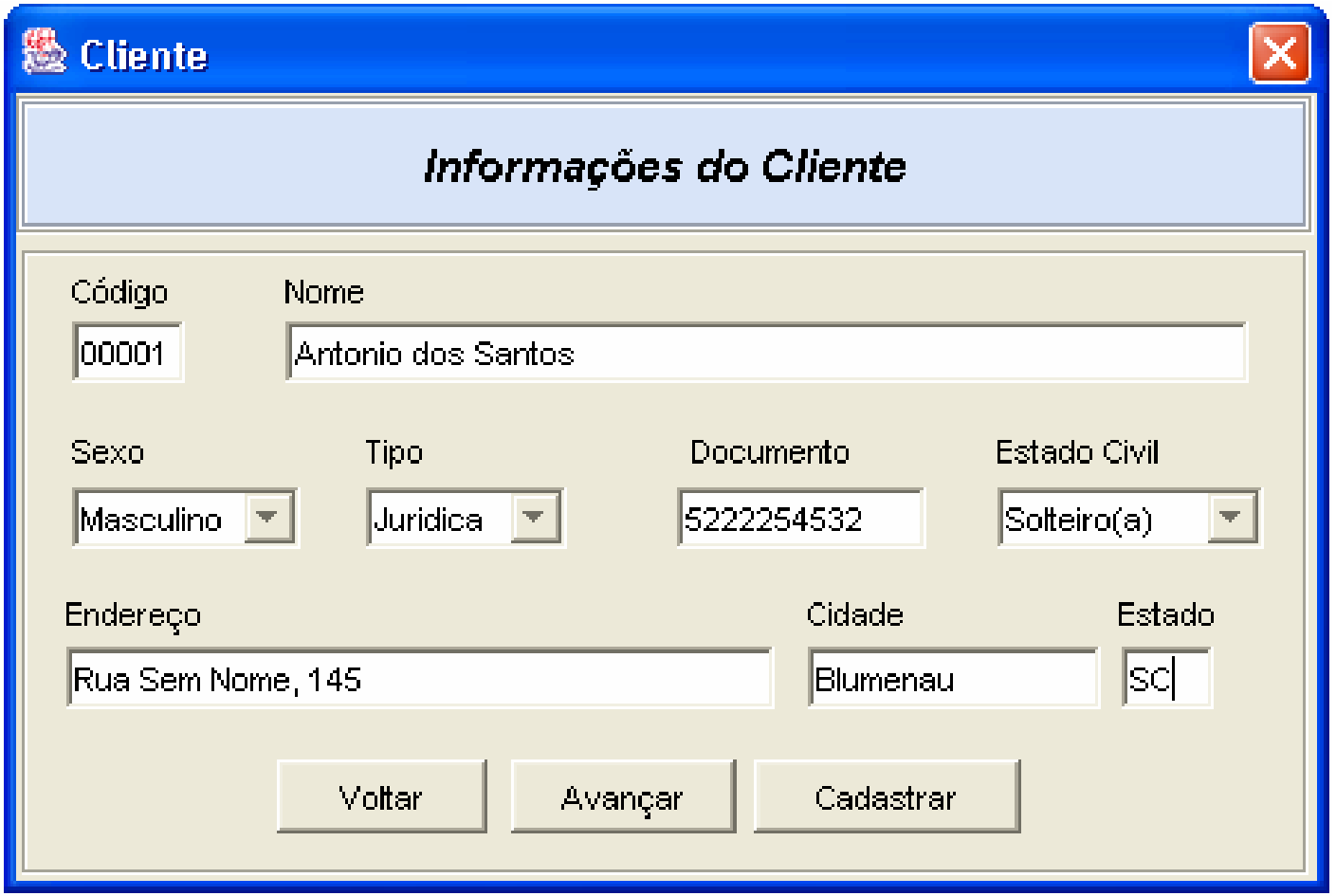


Implementação do Protótipo

Diagrama de seqüência



Implementação do Protótipo
Cadastro de Clientes



Conclusão

- Aprimoramento dos conceitos e prática da modelagem OO;
- Software em camadas proporciona uma melhor acomodação das funcionalidades na qual o sistema se destina;
- Maior quantidade de itens a serem reutilizados;
- Criação de micro unidades de fácil manutenção;

Conclusão

- Criação do *framework* – facilitando a tarefa do desenvolvedor quanto a reutilização e manutenção e do desenvolvedor da aplicação que abstrai a complexidade dos cálculos de impostos.

Demonstração do Protótipo implementado utilizando o *Framework* de cálculo de Impostos