ÉRIKA BARBOSA FRANCO MARRA

UMA INTERFACE DE ACOMPANHAMENTO DAS ETAPAS DA ENGENHARIA DE SOFTWARE NO SISTEMA DE INFORMAÇÃO DA ACADEMIA KEMPER FORMA.

Monografia de projeto orientado apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharelado em Ciência da Computação.

Orientadora

Profª. Olinda Nogueira Paes Cardoso

Co-orientador

Profº Antônio Maria Pereira Resende

LAVRAS

MINAS GERAIS – BRASIL

2002

ÉRIKA BARBOSA FRANCO MARRA

UMA INTERFACE DE ACOMPANHAMENTO DAS ETAPAS DA ENGENHARIA DE SOFTWARE NO SISTEMA DE INFORMAÇÃO DA ACADEMIA KEMPER FORMA.

Monografia de projeto orientado apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharelado em Ciência da Computação.

Orientadora

Profª. Olinda Nogueira Paes Cardoso

Co-orientador

Profº Antônio Maria Pereira Resende

LAVRAS

MINAS GERAIS – BRASIL

2002

DEDICATÓRIA

Dedico primeiramente a Deus, que esteve e continua sempre comigo, pois sem ele não encontraria força necessária para superar as dificuldades desta caminhada.

Dedico aos meus pais por terem acreditado e tornada realidade a realização desta conquista.

"O ideal da vida não é a esperança de chegar a ser perfeito e sim a vontade de ser cada dia melhor."

(Autor desconhecido)

AGRADECIMENTOS

Agradeço aos meus pais que investiram seu tempo e sonhos em meus estudos. Às minhas irmãs, amigos e a todas as pessoas que acreditaram em mim e que, direta ou indiretamente, influenciaram de forma positiva na minha formação.

Agradeço aos meus amigos Jerusa, Juliano, Vaninha, Juliana, Gláucia, Vanessa, Hesli e Rodrigo que não foram somente meus amigos, mas sim companheiros de uma caminhada que está apenas começando.

Agradeço também aos meus mestres, principalmente a Olinda e Antônio Maria, por repartir seus conhecimentos, colocando em minhas mãos as ferramentas com as quais consegui desenvolver meu projeto.

SUMÁRIO

ÍNDICE DE TABELAS

INDÍCE DE FIGURAS

1 INTRODUÇÃO

As empresas modernas estão percebendo que o gerenciamento da informação como um recurso é uma questão estratégica para a sobrevivência em mercados cada vez mais competitivos e frente a recursos cada vez mais escassos. Diante desta situação, os sistemas de informação tornaram-se peças-chave no gerenciamento da informação, e o sucesso no seu desenvolvimento, operação e manutenção são importantes para as empresas, a fim de proporcionar capacidade competitiva no mercado e permitir mais precisão e agilidade na tomada de decisão sobre o uso eficiente desses recursos.

Uma das grandes preocupações das organizações é a necessidade de se criar sistemas de informação rápidos, com qualidade e baixo custo. Por isto, tornou-se imprescindível a adoção de métodos que possam proporcionar mais estabilidade ao processo de desenvolvimento destes sistemas. Os métodos envolvem um amplo conjunto de tarefas que incluem planejamento, estimativa de projeto, análise de requisitos de sistemas, arquitetura de programa e algoritmo de processamento, codificação, teste e manutenção [7].

O planejamento do processo de gerenciamento do projeto de sistemas é fundamental para se obter um *software* bem-sucedido. Primeiramente é necessário compreender o escopo do trabalho a ser feito, os riscos que podem ocorrer, os recursos exigidos, as tarefas a serem executadas, os marcos de referência a serem acompanhados, o esforço despendido e a programação a ser seguida.

Não existe uma abordagem em particular do que seja melhor para a solução dos problemas que podem ocorrer durante o desenvolvimento de um sistema. Entretanto, ao combinarmos métodos abrangentes para todas as etapas de desenvolvimento do *software*, pode-se conseguir uma disciplina para o desenvolvimento de sistemas, disciplina esta chamada de Engenharia de Software [7].

Engenharia de Software (ES) é o ramo da Ciência da Computação que desenvolve e aplica métodos, técnicas e ferramentas na construção de sistemas confiáveis, eficientes e com custos aceitáveis [7].

A modelagem do sistema é uma parte central de todas as atividades, proporcionando a implantação de um bom *software* e criação da documentação, com a finalidade de satisfazer as necessidades do usuário. Esta é uma tarefa difícil, muitas vezes realizada de forma descuidada, por não considerarem uma etapa importante.

O objetivo deste trabalho é realizar as etapas de desenvolvimento da modelagem, aplicando as técnicas de ES, a fim de propiciar a criação de um sistema de informação confiável, eficiente, de fácil integração com o usuário utilizando-se dos recursos que a informatização dispõe e eliminar o trabalho manual com a atualização automática dos dados. Para isto, a **Academia Kemper Forma** foi escolhida como empresa-exemplo, visando ilustrar todo o desenvolvimento do trabalho.

Paralelamente, será desenvolvida uma interface, que ficará disponível na internet, da documentação *on-line* das etapas deste projeto desenvolvidas durante o trabalho usando UML (*Unified Modeling Langrage)*, descrevendo detalhadamente o sistema, uma inovação tecnológica de grande importância para o entendimento e manutenção do mesmo.

O capítulo 2 trata dos conceitos relativos ao método de desenvolvimento utilizando a UML, o Capítulo 3 apresenta a proposta de trabalho, o Capítulo 4 apresenta material e métodos utilizados no trabalho, no Capítulo 5 é apresentada a interface desenvolvida, o Capítulo 6 apresenta os resultados obtidos e os últimos Capítulos apresentam a conclusão, sugestões de trabalhos futuros e referências bibliográficas.

2 REVISÃO DE LITERATURA

2.1 Método de desenvolvimento utilizando a UML

Grande parte do desenvolvimento de sistemas fracassa devido, principalmente a má administração de riscos, construção de sistemas errados e por superestimar a tecnologia. Entre os riscos pode-se citar problemas de domínio de tecnologia, riscos financeiros, impossibilidade de fazer integração, inexistência de teste, ausência de interação com o usuário, entre outros.

Uma das maneiras de evitar o fracasso é definir um processo de desenvolvimento de sistemas. O processo de desenvolvimento de sistemas de informação é formado por um conjunto padronizado e documentado de atividades para todas as etapas do ciclo de desenvolvimento.

A escolha de uma metodologia constitui um dos passos mais importantes no processo de desenvolvimento de sistemas, e é através dele que as equipes e seus membros mantêm uma linguagem comum durante todo o ciclo de desenvolvimento.

Os principais benefícios na adoção de um método são: [8]

- permitir o compartilhamento da mesma filosofia de trabalho entre os desenvolvedores de sistemas de informação;
- evitar trabalho desnecessário;
- aumentar a produtividade;
- estabelecer pontos de verificação para acompanhamento e controle de execução do projeto de desenvolvimento;
- facilitar o envolvimento do usuário no projeto;
- prover uma notação e linguagem comum.

A seguir será descrita cada etapa do desenvolvimento de um sistema em UML.

2.1.1 Etapas do desenvolvimento de um sistema em UML

Existem seis etapas no desenvolvimento de sistemas: análise de requisitos, análise, projeto, implementação, testes e manutenção, como mostra a Figura 1. Estas cinco etapas devem ser seguidas de forma que, quando algum problema for detectado numa certa fase, deve-se modificar e melhorar as fases desenvolvidas anteriormente a fim de que o resultado global gere um produto de alta qualidade e desempenho [7].

A modelagem ajuda o analista no planejamento e na determinação do produto final, na determinação das fases, etapas e tarefas e determinação de um planejamento, um cronograma, ferramentas e levantamento das equipes [7].

A UML (*Unified Modeling Language*) é uma linguagem de modelagem orientada a objetos designada para especificar o comportamento e a estrutura de um sistema, visualizar um sistema como ele é ou como se deseja que ele seja, auxiliar no desenvolvimento e documentar um sistema. Por isso, e muitas outras razões, o bom entendimento da UML não é apenas aprender a simbologia e o seu significado, mas também significa aprender a modelar orientado a objetos no estado da arte [7].

A UML pode ser usada para: [7]

- mostrar as fronteiras de um sistema e suas funções principais utilizando atores e casos de uso;
- ilustrar a realização de casos de uso com diagramas de interação;
- representar uma estrutura estática de um sistema utilizando diagramas de classe;
- modelar o comportamento de objetos com diagramas de transição de estado;
- revelar a arquitetura de implementação física com diagramas de comportamento e de implementação;

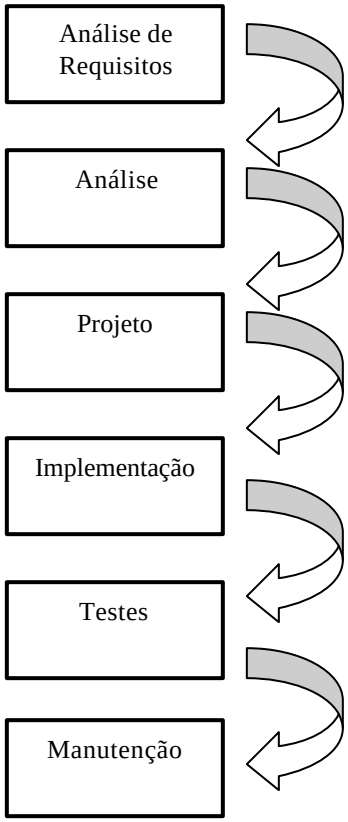


Figura 1: Etapas do projeto

2.1.1.1 Análise de requisitos

Esta etapa se caracteriza pela definição do comportamento do sistema, ou seja, consiste na entrevista com o cliente/usuário, no qual serão relatadas as intenções e necessidades dos usuários do sistema, a ser desenvolvidas através do uso de funções chamadas caso de uso (*use-cases*). Através do desenvolvimento do caso de uso, as entidades externas ao sistema (em UML chamados de "atores externos") que interagem e possuem interesse no sistema são modelados entre as funções que eles requerem. Os atores externos e os casos de uso são modelados

com relacionamentos que possuem comunicação associativa entre eles ou são desmembrados em hierarquia. Cada caso de uso modelado é descrito através de um texto onde são especificados os requerimentos do ator externo que utilizará este caso de uso [8].

2.1.1.2 Análise

Esta etapa se caracteriza pela identificação de classes de objetos com características e comportamento.

As classes são modeladas e ligadas através de relacionamentos com outras classes, e são descritas no Diagrama de Classe. As colaborações entre classes também são mostradas neste diagrama para desenvolver os casos de uso modelados anteriormente [7].

2.1.1.3 Projeto

Durante a análise, é dado o enfoque sobre o que precisa ser feito, independentemente de como deve ser feito. No projeto são encontradas as soluções de como o problema será resolvido.

O Diagrama de Classe é a principal estrutura em torno da qual é desenvolvido o projeto. O objetivo do projeto é refinar e otimizar os modelos de análise a fim de estabelecer a base para a implementação [7].

2.1.1.4 Implementação

Nesta etapa, as classes provenientes do projeto são convertidas para o código da linguagem orientada a objetos escolhida. A implementação deve ser a parte de menor importância do ciclo de desenvolvimento, porque todas as decisões difíceis devem ser tomadas durante o projeto .

2.1.1.5 Testes

Um sistema normalmente é executado em testes de unidade, integração, e aceitação. Os testes de unidade são para classes individuais ou grupos de classes e são geralmente testados pelo programador [8]. Os testes de integração são aplicados usando as classes e componentes integrados para confirmar se as classes estão cooperando umas com as outras como especificado nos modelos. Os testes de aceitação observam o sistema como uma “caixa preta" e verificam se o sistema está funcionando como o especificado nos primeiros diagramas de Casos de Uso [7].

A Tabela 1 apresenta cada uma das etapas descritas com os respectivos diagramas utilizados, cujos conceitos serão apresentados na sessão 2.3.

Etapa	Diagramas utilizados
Análise de requisitos	Diagrama de Caso de Uso; Diagrama de Seqüência; Diagrama de colaboração; Documento de requisitos (descrição funcional, fluxo de eventos).
Análise e Projeto	Diagrama de Transição de Estados; Diagrama de Classe.
Implementação	Diagrama de Componente Código fonte Versão executável do sistema
Teste	Resultado da realização dos testes de integração

Tabela 1: Diagramas correspondentes às etapas do projeto

Na seção 2.2 será descrito como se pode realizar o levantamento de dados a fim de obter os requisitos necessários para a descrição do **escopo**.

2.2 Planejamento inicial

O projeto de desenvolvimento de qualquer sistema de informação deve começar com o plano de desenvolvimento. Neste plano deve constar o que o desenvolvedor deve entregar ao usuário, analisando todos os requisitos necessários para satisfazer as necessidades do usuário. Para os desenvolvedores, este plano é um dos documentos mais importantes porque define o **escopo do projeto** [7].

2.2.1 Processo de levantamento de dados

O processo de levantamento de dados consiste em estabelecer a abordagem que será utilizada no levantamento de dados de acordo com a complexidade da aplicação.

O levantamento poderá ser realizado através de entrevistas, reuniões, questionários, uso de formulários de descrição do processo de identificação dos eventos, etc. Esta entrevista pode ocorrer de várias maneiras. Provavelmente a entrevista é feita com os usuários, gerentes, auditores, programadores que fazem a manutenção de sistemas já existentes e várias outras pessoas.

Estas entrevistas são feitas por alguns motivos: [8]

- inicialmente cria-se uma lista de atividades e/ou tarefas que o cliente/usuário realiza diariamente, semanalmente, mensalmente e anualmente. Em resumo, todas as atividades pela qual o usuário é responsável devem ser registradas nesta lista de tarefas. Precisa-se coletar informações sobre o comportamento de um sistema atual ou sobre os requisitos de um novo sistema, das pessoas que têm essas informações armazenadas em algum lugar;
- precisa-se verificar nossa própria compreensão, como analista, do comportamento ou dos requisitos de um sistema atual ou dos

requisitos de um novo sistema. Essa compreensão, deve ter sido adquirida através de entrevistas prévias juntamente com informações coletadas de modo independente;

- precisa-se coletar informações sobre os sistemas atuais para ser executado um estudo de custo/benefício.

A forma especializada de entrevista que se tornou popular em algumas empresas é conhecida como JAD (*Joint Application Development*) ou projeto acelerado, ou análise em equipe, e por vários outros nomes. Ela consiste em uma rápida entrevista e um processo acelerado de coleta de dados em que todos os principais usuários e o pessoal da análise de sistemas agrupam-se em uma única e intensiva reunião para documentar os requisitos do usuário [8].

A maneira mais comum de entrevista é uma reunião pessoal e direta entre o analista e os usuários do sistema. Normalmente tomam-se apontamentos com papel e lápis por um dos entrevistadores, ou também a entrevista pode ser gravada.

Existem muitos problemas que podem ocorrer. Por exemplo, em projetos de alta tecnologia, a maioria dos problemas difíceis não envolve *hardware*, nem *software*, mas sim a comunicação entre os analistas e usuários. Alguns dos problemas são [8]:

- entrevistar a pessoa errada e/ou no momento errado;
- fazer perguntas erradas e obter respostas erradas: Os usuários e analistas de sistemas têm vocabulários diferentes, experiências básicas diferentes e muitas vezes um diferente conjunto de pressuposições, percepções, valores e prioridades. Deste modo, é fácil fazer ao usuário uma pergunta racional sobre os requisitos do sistema e o usuário não entender essas informações.
- criar ressentimentos recíprocos: Existem algumas razões pelas quais o usuário pode não se sentir à vontade ou mesmo colocar-se em

posição antagônica na entrevista com o analista. Por exemplo, ele pensa que o verdadeiro objetivo do novo sistema que o especialista está especificando é tomar-lhe o emprego.

É importante certificar-se que o analista tem autorização para falar com os usuários. Na maior parte dos casos, basta uma autorização verbal, mas se a organização for altamente burocrática, será preciso uma autorização por escrito.

Para fazer uso eficiente do tempo é necessário que a entrevista, seja planejada e preparada tão antecipadamente quanto possível para fazer uso eficiente da entrevista.

Depois de listadas todas as atividades e divididas em grupo, é elaborado um texto em português para descrever o que ocorre na empresa. Este documento denomina-se **escopo**. Considera-se esta etapa terminada quando o escopo está descrito num documento formal e tanto o cliente/usuário, quanto o engenheiro de *software* estão de acordo com o conteúdo.

2.3 Descrição dos diagramas

Além dos Diagramas de Caso de Uso, Seqüência e Classe que serão descritos e utilizados no desenvolvimento do projeto demonstrando a modelagem do sistema, existem os Diagramas de Estado, Atividade, Comportamento e de Implementação que não serão explicados.

2.3.1 Diagrama de Caso de Uso

Casos de uso são descrições de processos do domínio, ou seja, um caso de uso é um documento narrativo que descreve a seqüência de eventos de um ator (agente externo) que usa um sistema para completar um processo. É bom ressaltar que casos de uso não são exatamente especificação de requisitos ou

especificação funcional, mas ilustram e implicam requisitos necessários para satisfazer as exigências do cliente [4].

Um Diagrama de Caso de Uso ilustra um conjunto de casos de uso para um sistema, os atores e a relação entre os atores e os casos de uso, especificados na Tabela 2. A finalidade deste diagrama é apresentar um tipo de diagrama de contexto, através do qual pode-se compreender rapidamente quais são os atores externos de um sistema e as maneiras principais, segundo as quais eles o utilizam [5]. A seção 6.2.1 mostra um exemplo de Diagrama de Caso de Uso, no qual o ator administrador interage com os casos de uso descritos na seção 6.1.3.

Caso de Uso:	Deve-se escrever o nome do caso de uso.
Atores:	Deve-se listar o nome do atores envolvidos neste caso de uso.
Descrição:	A descrição do caso de uso em si, ou seja, uma descrição superficial das ações que compõem o processo representado.

Tabela 2: Formato dos casos de uso

2.3.2 Diagrama de Interação

Este tipo de diagrama tem como objetivo mostrar o comportamento dos vários objetos relacionados, durante a realização do caso de uso. Ele é classificado em dois tipos de diagramas, de acordo com o foco que ele demonstra: Diagrama de Seqüência (foco na seqüência das mensagens trocadas entre os objetos) e Diagrama de Colaboração (foco no comportamento de cada objeto dentro do caso de uso). Ambos os diagramas representam os aspectos dinâmicos relativos à modelagem do caso de uso, indicando as interações dos

usuários com o sistema, e a troca de mensagens entre as diversas classes relacionadas ao caso de uso para produzir o resultado desejado.

A inspeção da existência de um Diagrama de Seqüência e/ou Colaboração relacionado ao caso de uso, obedece aos seguintes critérios [9]:

- deve haver um rastreamento direto entre o caso de uso e o Diagrama de Interação, ou seja, este diagrama deve pertencer a realização do caso de uso, ou estar imediatamente abaixo do caso de uso na hierarquia do projeto (caso se esteja utilizando alguma ferramenta que permita esse tipo de ligação, por exemplo, a *Rational Rose*);
- o diagrama deve possuir o mesmo nome de identificação do caso de uso;
- deve representar as diversas classes de análise envolvidas com a realização do caso de uso e que modelam o comportamento do mesmo;
- deve representar as classes de controle que monitoram as classes de análise, de forma a obter os resultados desejados;
- deve conter mensagens que permitem realizar os fluxos de eventos descritos para o caso de uso.

2.3.2.1 Diagrama de Seqüência

O Diagrama de Seqüência é uma forma de visualizar um cenário na ordem em que ele acontece em determinado tempo. Este diagrama é mais fácil de ser interpretado pelo usuário [5].

O Diagrama de Seqüência deve ser feito para a seqüência típica de eventos do caso de uso, e, possivelmente, outros diagramas de seqüência para as outras seqüências alternativas mais interessantes [5].

Objetos são desenhados, como retângulos, com nomes sublinhados e a linha de vida de cada objeto é representada por uma linha tracejada.

As interações (mensagens) são mostradas como setas horizontais, direcionadas da linha vertical, representado o objeto cliente (transmissor) para a linha representando o objeto servidor (receptor). As setas são denominadas mensagens. A ordenação no tempo dos eventos é indicada pela posição vertical, sendo que o primeiro evento aparece no topo [7].

A seção 6.2.2 mostra alguns Diagramas de Seqüência modelados, apresentando a seqüência típica de eventos de cada caso de uso. Esta seqüência é verificada nas tabelas da seção 6.1.3.

2.3.3 Diagrama de Classe

Esse diagrama representa a estrutura das diversas classes relacionadas a realização do caso de uso, indicando também os relacionamentos entre tais classes. A inspeção desse diagrama, visa identificar as diversas classes necessárias para realização do caso de uso e obedece aos seguintes critérios [9]:

- as classes de análise definidas no Diagrama de Interação devem estar representadas no Diagrama de Classe contendo: atributos (possíveis parâmetros passados nas mensagens do diagrama de interação, ou ainda atributos relativos a análise do caso de uso), métodos (correspondem as mensagens trocadas nos diagramas de interação e métodos de acesso aos atributos do sistema) e relacionamentos com outras classes (resultantes do comportamento da classe para atingir a funcionalidade do caso de uso);
- as classes de controle definidas no Diagrama de Seqüência do caso de uso devem estar representadas no Diagrama de Classe, contendo os atributos e métodos (mensagens que ela envia) necessários, para controlar a execução do caso de uso;

- as classes de fronteira relacionadas com a prototipação do caso de uso devem estar representadas no Diagrama de Classe, contendo atributos, métodos relevantes e relacionamentos com outras classes bem definidos.

A seção 6.2.3 mostra o Diagrama de Classe modelado, dividido em cinco Figuras, cada uma representando uma parte do diagrama.

3 PROPOSIÇÃO

Este trabalho visa desenvolver as etapas da modelagem dos principais requisitos do sistema usando UML, descrevendo toda a documentação, a fim de propiciar a criação de um sistema de informação para **Academia Kemper Forma**.

No final deste estudo pretende-se disponibilizar uma interface de acesso às informações das etapas do projeto desenvolvidas neste trabalho, a fim de demonstrar através dos Diagramas de Caso de Uso, Seqüência e Classe a documentação do funcionamento do sistema.

Esta academia foi escolhida como empresa-exemplo, por 3 motivos:

- houve uma demonstração de interesse por parte do proprietário em desenvolver um novo sistema, pois os atuais não estavam suprindo suas necessidades;
- havia uma facilidade de acesso tanto à empresa, quanto às pessoas e aos sistemas, devido a uma confiança por parte do administrador;
- é um tipo de empresa que vem crescendo muito no mercado, possibilitando que não só o sistema para esta academia seja desenvolvido, como também para as demais que tenham interesse na informatização de suas atividades.

4 MATERIAL E MÉTODOS

Neste capítulo será apresentada a metodologia utilizada para a modelagem do sistema para obtenção dos resultados posteriores.

A modelagem dos principais requisitos do sistema administrativo a fim de obter uma visualização do seu funcionamento foi realizada a partir da ferramenta *Rational Rose 2000*, utilizando os Diagramas de Casos de Uso, Seqüência e de Classe.

4.1 Descrição do levantamento dos dados

O método utilizado para obter o levantamento dos requisitos necessários na descrição do escopo da academia foi o seguinte:

- primeiramente realizou-se uma reunião pessoal e direta com o gerente/proprietário da academia, na qual ele demonstrou qual era a complexidade do problema;
- na segunda reunião pôde-se perceber qual era o porte da academia, assim como o número de funcionários, alunos e oferta de atividades, realizada através de um questionário formulado antecipadamente;
- depois das duas reuniões realizou-se uma série de entrevistas com o gerente administrador, atual usuário do sistema, tomando-se apontamentos com papel e lápis, onde se criou uma lista de tarefas, registrando todas as atividades realizadas diariamente por ele;
- analisou-se o manual do sistema *Pyisical Control for Windows*, disponibilizado pelo proprietário;
- analisou-se o comportamento dos sistemas atuais, verificando juntamente com o administrador, os requisitos necessários para a criação de um novo sistema.

Após serem relatados todos os requisitos necessários para o desenvolvimento do sistema da **Academia Kemper Forma**, descreveu-se o **escopo,** a seguir.

4.2 Escopo

A **Academia Kemper Forma** localizada na cidade de Lavras-MG, à rua Francisco Sales, número 343, possui em seu quadro de funcionários três professores e um monitor na categoria musculação, seis professores na categoria ginástica, dois avaliadores físicos, um professor responsável pela ginástica olímpica, uma pessoa responsável pela limpeza e duas secretárias. Atualmente uma média de 300 alunos freqüenta as atividades oferecidas diariamente, tais como: Body Pump, Body Combat, Power Local, Step, Alongamento, Swing, Swing/Local, Step/Abdome, Combat/Attack, Super Local, Super Step, The Best of Body Combat, Body Systems, Capoeira, Forró, Condicionamento Físico e Musculação.

A academia possui dois sistemas independentes, desenvolvidos em *FoxPro* para Windows. Um dos sistemas é o *Physical Control 3.1 for Windows*, um programa que se propõe a automatizar algumas das mais importantes e trabalhosas tarefas administrativas de uma academia. Com este sistema é possível manter um cadastro (inclusão e alteração de dados) de clientes, acessando e manuseando os cadastros utilizados. Nele encontram-se as rotinas da academia que acontecem diariamente como matrículas, cancelamento de matrículas, venda de produtos ou serviços, agendamento/consulta/cancelamento de compromissos, consulta à situação de contas a pagar e a receber.

O sistema pode ser configurado para impedir automaticamente o acesso dos clientes na academia por atraso de pagamento, vencimento de exame médico, avaliação física ou nutricional, etc.

Há também o cadastro dos fornecedores que são identificados no lançamento das contas a pagar. Além da identificação do fornecedor, também é possível indicar a que categoria de despesa se relaciona aquela compra, possibilitando a emissão posterior de gráficos e relatórios das despesas por centro de custo (administração, folha de pagamento, manutenção do prédio, manutenção do funcionamento das atividades, etc).

Há outra opção que exibe gráficos gerenciais que auxiliam a tomada de decisão por parte do administrador, como: despesas por categoria no período (análise da procura das atividades oferecidas pela academia, agrupadas por categoria), análise da situação financeira e análise da situação geral dos clientes.

É possível solicitar a impressão da agenda de compromissos, da relação dos aniversariantes do mês, de clientes com exame vencidos, de clientes com matrículas vencidas e não renovadas ou listagem dos clientes com acesso impedido. Encontram-se também opções para emissão da relação das contas a pagar, já pagas e da relação das contas a receber ou já recebidas. Outra função do sistema é permitir o controle bancário do sistema, realizando lançamentos, transferências e fazer consultas. É interessante emitir relatório do controle de freqüência dos alunos, quantidade de pessoas por horário, a fim de obter dados estatísticos, tais como: meses do ano com maior e/ou menor freqüência, dias da semana que possuem maior e/ou menor freqüência num determinado horário e atividades mais procuradas.

O outro sistema responsável pela avaliação física, *o Physical Test for Windows*, permite ao cliente obter o conhecimento de seu condicionamento físico, a fim de realizar um trabalho consciente, bem orientado e adequado à sua realidade. Os testes contidos neste sistema foram formulados por profissionais de alto nível através de pesquisas constantes e é continuamente aperfeiçoado com o objetivo de fornecer ao cliente melhor condição de vida.

Primeiramente, o cliente mostra quais são seus objetivos com relação à atividade física, como estética, condicionamento físico, lazer, convívio social e terapêutico. Depois de expor o objetivo, o avaliador faz várias perguntas a fim de conhecer o estilo de vida do cliente, como: se ele já praticou algum tipo de atividade física e quanto tempo de prática, se há algum tipo de restrição à prática de atividade física, alergia, se utiliza algum tipo de medicamento e há quanto tempo, se há dores no corpo, se já sofreu algum acidente ou lesão, se tem o hábito de fumar e se faz dieta.

São necessários também medir a composição corporal (peso atual, altura, dobra toráxica, dobra abdominal, dobra triciptal, dobra supra_ilíaca, dobra da coxa) e as circunferências (tórax, cintura, abdome, quadril, antebraços, braços, coxas, panturrilhas).

Estas circunferências correspondem aos perímetros máximos de um segmento corporal. A atividade praticada pode alterar estas medidas, por exemplo, a musculação promove aumento da massa muscular e, conseqüentemente, o aumento do perímetro do segmento. Atividades cardiorrespiratórias podem reduzir as medidas de determinados perímetros corporais através da maior mobilização de gordura localizada.

É extremamente importante a avaliação da capacidade cardiorrespiratória, quantidade de oxigênio que um indivíduo consegue captar e metabolizar durante a atividade física, pois esta alteração de freqüência pode ser causa de várias lesões.

Neste sistema também é verificada a flexibilidade, força abdominal, força dos braços e avaliação postural (coluna vertebral, cintura escapular, cintura pélvica e membros inferiores).

Há também o sistema *MicroPoint S*, versão 1.08, responsável pela comunicação entre um equipamento eletrônico de controle ao acesso à academia, e os demais sistemas, porém este não está sendo utilizado.

Na sessão 4.3 serão feitas algumas considerações observadas a respeito dos sistemas atuais.

4.3 Considerações sobre os sistemas atuais

Através de informações levantadas com o administrador do sistema, e, além disso, por constatação própria, pode-se destacar inicialmente algumas considerações a serem feitas sobre os sistemas atuais:

- apesar do sistema satisfazer a maioria das necessidades do administrador, não possui nenhuma segurança. O acesso ao sistema de arquivos seqüenciais está totalmente disponível a qualquer usuário;
- além da falta de segurança, os sistemas atuais não se comunicam com o sistema da catraca eletrônica. Esta comunicação com a base de dados é extremamente importante, pois este controle não está sendo feito, motivo de vários prejuízos, como: alunos continuam freqüentando a academia, mesmo com mensalidades atrasadas, ou freqüentando musculação e ginástica, estando matriculados em apenas uma categoria;
- é interessante disponibilizar consultas do controle de freqüência dos alunos, quantidade de pessoas por horário, a fim de obter dados estatísticos, tais como: meses do ano com maior e/ou menor freqüência, dias da semana que possuem maior e/ou menor freqüência num determinado horário e atividades mais procuradas. Estas consultas ajudam na tomada de decisões, contribuindo com o gerenciamento da academia.

5 A INTERFACE

Carlos Alberto Sowek afirma “Documentação tem sido, ao longo do tempo, o calcanhar de Aquiles de muitas organizações de Tecnologia de Informação. Tão distante para obter, quanto para manter a documentação, o que parece ser um sonho impossível” [4].

Um dos maiores problemas do desenvolvimento de sistemas é a falta da documentação. Esta não é importante apenas no momento de desenvolvimento, como também na fase de manutenção, pois os requisitos do projeto modificam-se continuamente, mas as mudanças podem ser facilmente acomodadas, porque o *software* é flexível. Devido à falta da documentação estas mudanças podem tornar-se difíceis. Algumas vantagens de disponibilizar-se de uma boa documentação podem ser:

- O desenvolvedor que está fazendo a manutenção pode não ser a mesma pessoa que desenvolveu o sistema, com a documentação facilita o entendimento do seu funcionamento;
- Um problema ocorrido, ou mesmo a necessidade de um requisito solicitado pelo usuário, sem a documentação seria inviável a manutenção depois de um logo tempo;
- Com a modelagem documentada, o desenvolvedor evita um “desperdício” do tempo, com entrevistas fracassadas.

Visando alcançar a solução de problemas foi criada uma interface *on-line* para armazenar toda a documentação do sistema da **Academia Kemper Forma,** onde pessoas envolvidas no desenvolvimento têm acesso a documentação. Este acesso será controlado por uma senha, disponibilizada pelo administrador. Nesta interface estará disponível o escopo do sistema, a documentação de todos os diagramas desenvolvidos, mostrando suas especificações, operações e atributos.

A interface foi desenvolvida em *Front Page* e a página estará *on-line* para posterior observação da documentação disponível. Ela contém a descrição de 63 casos de uso, 16 Diagramas de Caso de Uso, 63 Diagramas de Seqüência, um Diagrama de Classe contendo 29 classes com todos os atributos e operações de cada classe documentados. Além da documentação da funcionalidade dos requisitos do sistema, há o código gerado de todas as classes. O Anexo A contém o código gerado da classe Cliente. Pela extensão da documentação, a interface é um excelente meio para se extrair as informações necessárias para o posterior desenvolvimento do sistema.

A interface está disponível no endereço www.comp.ufla.br/~marra/Interface/Index.htm.

6 RESULTADOS

Como foi mencionado na proposição este projeto contém a descrição das principais funcionalidades do sistema administrativo da **Academia Kemper Forma**, ou seja, nem todos os requisitos contidos no escopo foram descritos.

6.1 Especificação funcional do sistema da academia

6.1.1 Visão geral das funcionalidades do sistema

As Figuras 2 e 3 mostram as principais funcionalidades do ator Administrador do sistema.

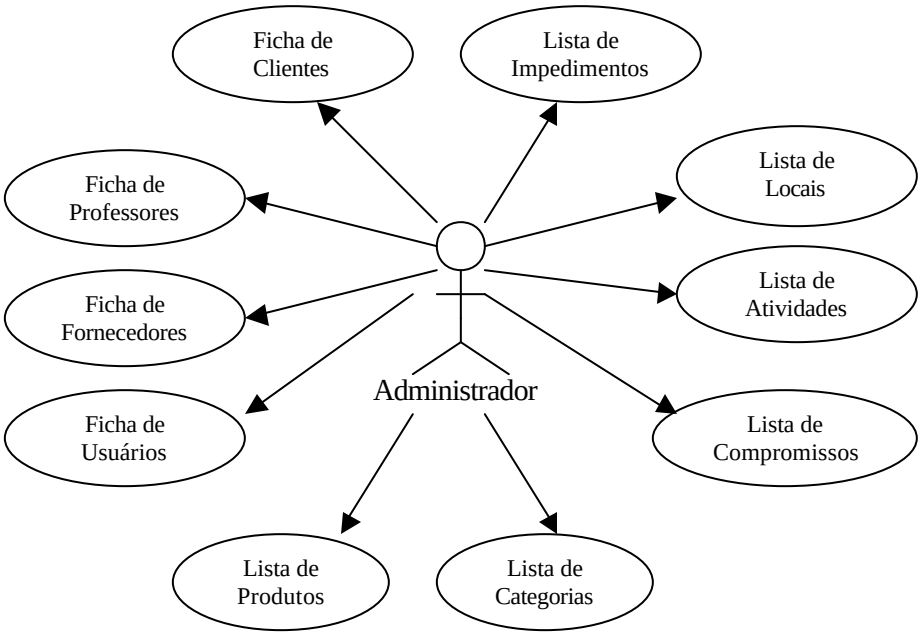


Figura 2: Principais funcionalidades do sistema

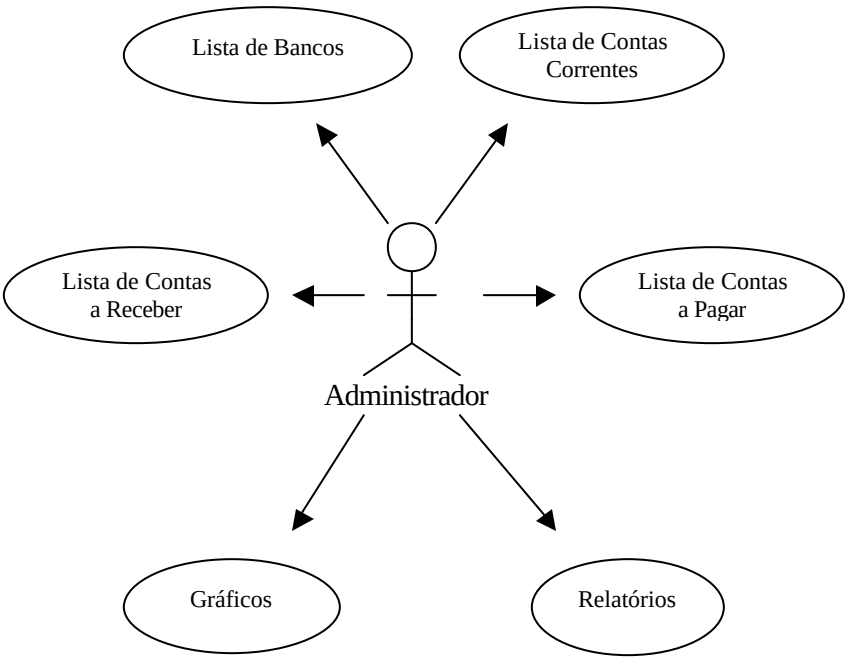


Figura 3: Principais funcionalidades do sistema

6.1.2 Casos de Uso

As Tabelas 3 e 4 mostram a relação existente dos casos de usos representados na Figura 2 e 3.

RELAÇÃO DOS CASOS DE USO		
Fun	**Nome da Funcionalidade**	**Relação dos Casos de Uso**
1	Ficha de Clientes	Cadastrar cliente; Cancelar matricula do cliente; Impedir acesso do cliente; Excluir cliente; Consultar clientes; Matricular cliente; Verificar situacao do cliente; Mostrar historico do cliente.

Continuação da tabela na próxima página.

2	Ficha de Professores	Cadastrar professor; Excluir professor; Consultar professores; Controlar aulas.
3	Ficha de Fornecedores	Cadastrar fornecedor; Excluir fornecedor; Consultar fornecedores.
4	Ficha de Usuários	Cadastrar usuario; Excluir usuario; Consultar usuarios.
5	Lista de Produtos	Cadastrar produto; Excluir produto; Vender produto; Consultar produtos.
6	Lista de Categorias	Cadastrar categoria; Excluir categoria; Consultar categorias.
7	Lista de Atividades	Cadastrar atividade; Excluir atividade; Consultar atividades.
8	Lista de Locais	Cadastrar local; Exclui local; Consultar locais.
9	Lista de compromissos	Cadastrar compromisso; Excluir compromisso; Consultar compromissos; Agendar compromisso; Cancelar compromisso agendado; Consultar compromissos agendados.
10	Lista de impedimentos	Cadastrar impedimento; Excluir impedimento; Consultar impedimentos.

Tabela 3: Relação dos casos de uso da Figura 2

RELAÇÃO DOS CASOS DE USO		
Fun	**Nome da Funcionalidade**	**Relação dos Casos de Uso**
11	Gráficos	Visualizar grafico.
12	Lista de Contas Correntes	Transferir fundos; Movimentar conta corrente; Consultar conta corrente.
13	Lista de Bancos	Cadastrar banco; Excluir banco; Consultar bancos;
14	Lista de Contas a Pagar	Cadastrar conta a pagar; Excluir conta a pagar; Consultar contas a pagar; Informar pagamento.
15	Lista de Contas a Receber	Cadastrar conta a receber; Excluir conta a receber; Consultar contas a receber; Informar recebimento.
16	Relatórios	Emitir relatorio da folha de pagamento; Emitir relatorio das atividades; Emitir relatorio dos clientes; Emitir relatorio das contas a pagar; Emitir relatorio das contas a receber; Emitir relatorio dos aniversariantes; Emitir relatorio dos compromissos agendados; Emitir relatorio de freqüência dos alunos; Emitir relatorio do extrato de conta corrente.

Tabela 4: Relação dos casos de uso da Figura 3

6.1.3 Descrição dos casos de uso

Nesta sessão serão mostradas através de tabelas as descrições dos casos de uso relacionados nas Tabelas 3 e 4.

- **Caso de Uso: Cadastrar cliente**

Nome do Caso de Uso: Cadastrar cliente	**Funcionalidade:** Ficha de Clientes
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: O administrador cadastra todos os clientes da academia, ou seja, inclui ou altera os dados de todas as pessoas que praticam ou consomem algum tipo de produto ou serviço da academia.	
PRÉ-CONDIÇÃO	
1- Dados do cliente disponíveis.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Verifica se o cliente não está cadastrado;	
2- Se estiver cadastrado, alterar dados modificados.	
3- Caso contrário, inserir os dados pessoais do cliente;	
4- Armazena os dados na base de dados.	
PÓS-CONDIÇÃO	
1- Cliente cadastrado.	

Tabela 5: Cadastrar cliente

- **Caso de Uso: Cadastrar professor**

Nome do Caso de Uso: Cadastrar professor	**Funcionalidade:** Ficha de Professores
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: O administrador cadastra todos os professores da academia, ou seja, todas as pessoas capacitadas a orientar pessoas que praticam alguma atividade na academia ou mesmo em qualquer outro lugar.	
PRÉ-CONDIÇÃO	
1- Dados do professor disponíveis.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Verifica se o professor não está cadastrado;	
2- Se estiver cadastrado, alterar dados modificados.	
3- Caso contrário, inserir os dados do professor;	
4- Armazena os dados na base de dados;	
PÓS-CONDIÇÃO	
1- Professor cadastrado.	

Tabela 6: Cadastrar professor

- **Caso de Uso: Cadastra local**

Nome do Caso de Uso: Cadastrar local	**Funcionalidade:** Lista de Locais
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: O administrador cadastra todas as dependências da academia, por exemplo: piscina, sala de musculação, sala de ginástica.	
PRÉ-CONDIÇÃO	
1- Dados do local disponíveis.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Verifica se o local não está cadastrado;	
2- Se estiver cadastrado, alterar os dados modificados.	
3- Caso contrário, inserir os dados do local;	
4- Armazena os dados na base de dados.	
PÓS-CONDIÇÃO	
1- Local cadastrado.	

Tabela 7: Cadastra local

- **Caso de Uso: Cadastrar atividade**

Nome do Caso de Uso: Cadastrar atividade	**Funcionalidade:** Lista de Atividades
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Cadastra todas as atividades que a academia oferece a seus clientes. Exemplos de atividades: Body Pump, Body Combat, Power Local, Step, Alongamento, Swing, Swing/Local, Step/Abdome, Combat/Attack, Super Local, Super Step, The Best of Body Combat, Body Systems, Capoeira e Forró.	
PRÉ-CONDIÇÃO	
1- Dados da atividade disponíveis.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Verifica se a atividade não está cadastrada;	
2- Se estiver cadastrada, alterar os dados modificados.	
3- Caso contrário, inserir os dados da atividade;	
4- Armazena os dados na base de dados.	
PÓS-CONDIÇÃO	
1- Atividade cadastrada.	

Tabela 8: Cadastrar atividade

- **Caso de Uso: Cadastrar produto**

Nome do Caso de Uso: Cadastrar produto	**Funcionalidade:** Lista de Produtos
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: O administrador cadastra todos os produtos e serviços que a academia oferece aos seus clientes, por exemplo: roupa de ginástica, óculos para natação, tornozeleira, salgados, luvas, etc;	
PRÉ-CONDIÇÃO	
1- Dados do produto disponíveis.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Verifica se o produto não está cadastrado;	
2- Se estiver cadastrado, alterar os dados modificados.	
3- Caso contrário, inserir os dados do produto;	
4- Armazena os dados na base de dados.	
PÓS-CONDIÇÃO	
1- Produto cadastrado.	

Tabela 9: Cadastrar produto

- **Caso de Uso: Cadastrar impedimento**

Nome do Caso de Uso: Cadastrar impedimento	**Funcionalidade:** Lista de Impedimentos
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: O administrador cadastra os códigos de impedimento. Estes códigos restringem o acesso de clientes que tenham alguma pendência com a academia. Por exemplo: matrícula vencida, pendência de pagamento, exame médico vencido, avaliação física vencida, avaliação nutricional vencida, não matriculado, entre outros.	
PRÉ-CONDIÇÃO	
1- Dados do impedimento disponíveis.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Verifica se o impedimento não está cadastrado;	
2- Se estiver cadastrado, alterar os dados modificados.	
3- Caso contrário, inserir os dados do impedimento;	
4- Armazena os dados na base de dados.	
PÓS-CONDIÇÃO	
1- Impedimento cadastrado.	

Tabela 10: Cadastrar impedimento

- **Caso de Uso: Cadastrar banco**

Nome do Caso de Uso: Cadastrar banco	**Funcionalidade:** Lista de Bancos
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: O administrador cadastra todas os bancos em que o a academia possui uma conta corrente.	
PRÉ-CONDIÇÃO	
1- Dados do banco disponíveis.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Verifica se o banco não está cadastrado;	
2- Se estiver cadastrado, alterar os dados modificados.	
3- Caso contrário, inserir os dados do banco;	
4- Armazena os dados na base de dados.	
PÓS-CONDIÇÃO	
1- Banco cadastrado.	

Tabela 11: Cadastrar banco

- **Caso de Uso: Cadastrar fornecedor**

Nome do Caso de Uso: Cadastrar fornecedor	**Funcionalidade:** Ficha de Fornecedores
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: O administrador cadastra todos os fornecedores e prestadores de serviço da academia referente a produtos e matérias fornecidas por eles.	
PRÉ-CONDIÇÃO	
1- Dados do fornecedor disponíveis.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Verifica se o fornecedor não está cadastrado;	
2- Se estiver cadastrado, alterar os dados modificados.	
3- Caso contrário, inserir os dados do fornecedor;	
4- Armazena os dados;	
PÓS-CONDIÇÃO	
1- Fornecedor cadastrado.	

Tabela 12: Cadastrar fornecedor

- **Caso de Uso: Cadastrar usuario**

Nome do Caso de Uso: Cadastrar usuario	**Funcionalidade:** Ficha de Usuarios
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: O administrador cadastra os usuários do sistema e disponibiliza o que cada usuário já definido poderá utilizar dentro do sistema.	
PRÉ-CONDIÇÃO	
1- Dados do usuário disponíveis.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Verifica se o usuário não está cadastrado;	
2- Se estiver cadastrado, alterar os dados modificados.	
3- Caso contrário, inserir os dados do usuário;	
4- Armazena os dados na base de dados.	
PÓS-CONDIÇÃO	
1- Usuário cadastrado.	

Tabela 13: Cadastrar usuario

- **Caso de Uso: Cadastrar compromisso**

Nome do Caso de Uso: Cadastrar compromisso	**Funcionalidade:** Lista de Compromissos
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: O administrador cadastra todos os compromissos que podem ser marcados referente a academia. Alguns exemplos de compromisso podem ser exame médico, avaliação física, avaliação nutricional, torneio interno, palestra sobre atividade física.	
PRÉ-CONDIÇÃO	
1- Dados do compromisso disponíveis.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Verifica se o compromisso não está cadastrado;	
2- Se estiver cadastrado, alterar os dados modificados.	
3- Caso contrário, inserir os dados do compromisso;	
4- Armazena os dados;	
PÓS-CONDIÇÃO	
1- Compromisso cadastrado.	

Tabela 14: Cadastrar compromisso

- **Caso de Uso: Cadastrar conta a pagar**

Nome do Caso de Uso: Cadastrar conta a pagar	**Funcionalidade:** Lista Contas a Pagar
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: O administrador cadastra todas as despesas que a academia consome. Estas despesas podem ser agrupadas por fornecedores ou data de vencimento.	
PRÉ-CONDIÇÃO	
1- Dados da conta a pagar disponíveis.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Verifica se a conta a pagar não está cadastrada;	
2- Se estiver cadastrada, alterar os dados modificados.	
3- Caso contrário, inserir os dados da conta a pagar;	
4- Armazena os dados na base de dados.	
PÓS-CONDIÇÃO	
1- Conta a pagar cadastrada.	

Tabela 15: Cadastrar conta a pagar

- **Caso de Uso: Cadastrar conta a receber**

Nome do Caso de Uso: Cadastrar conta a receber	**Funcionalidade:** Lista de Contas a Receber
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: O administrador cadastra as contas a receber dos clientes, permitindo que seja realizado um controle das despesas.	
PRÉ-CONDIÇÃO	
1- Dados das contas a receber disponíveis.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Verifica se a conta receber não está cadastrada;	
2- Se estiver cadastrada, alterar os dados modificados.	
3- Caso contrário, inserir os dados da conta a receber;	
4- Armazena os dados na base de dados;	
PÓS-CONDIÇÃO	
1- Conta a Receber cadastrada.	

Tabela 16: Cadastrar conta a receber

- **Caso de Uso: Cadastrar categoria**

Nome do Caso de Uso: Cadastrar categoria	**Funcionalidade:** Lista de Categorias
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Organiza as atividades, produtos e serviços que a academia oferece e também as despesas ou centros de custo da empresa. Alguns exemplos de categoria de atividade são natação, hidroginástica, musculação, ginástica, judô, capoeira, taxas diversas, exames e avaliações, lanchonete, entre outros (tudo que será interpretado como crédito para academia). Também pode-se cadastrar os centros de custa da academia, como: administração, folha de pagamento, manutenção do funcionamento das atividades, etc.	
PRÉ-CONDIÇÃO	
1- Dados da categoria disponíveis.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Verifica se a categoria não está cadastrada;	
2- Se estiver cadastrada, alterar os dados modificados.	
3- Caso contrário, inserir os dados da categoria;	
4- Armazena os dados na base de dados.	
PÓS-CONDIÇÃO	
1- Categoria cadastrada.	

Tabela 17: Cadastrar Categoria

- **Caso de Uso: Excluir categoria**

Nome do Caso de Uso: Excluir categoria	**Funcionalidade:** Lista de Categorias
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Exclui uma categoria cadastrada.	
PRÉ-CONDIÇÃO	
1- Categoria cadastrada.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Informa a categoria;	
2- Se a categoria for encontrada, excluir;	
3- Confirma a exclusão.	
4- Caso contrário, finaliza.	
PÓS-CONDIÇÃO	
1- Categoria excluída.	

Tabela 18: Excluir categoria

- **Caso de Uso: Excluir professor**

Nome do Caso de Uso: Excluir professor	**Funcionalidade:** Ficha de Professores
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Exclui um professor cadastrado.	
PRÉ-CONDIÇÃO	
1- Professor cadastrado.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Informa o professor;	
2- Se o professor for encontrado, excluir;	
3- Confirma a exclusão.	
4- Caso contrário, finaliza.	
PÓS-CONDIÇÃO	
1- Professor excluído.	

Tabela 19: Excluir professor

- **Caso de Uso: Excluir atividade**

Nome do Caso de Uso: Excluir atividade	**Funcionalidade:** Lista de Atividades
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Exclui uma atividade cadastrada.	
PRÉ-CONDIÇÃO	
1- Atividade cadastrada.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Informa a atividade;	
2- Se a atividade for encontrada, excluir;	
3- Confirma a exclusão.	
4- Caso contrário, finaliza.	
PÓS-CONDIÇÃO	
1- Atividade excluída.	

Tabela 20: Excluir atividade

- **Caso de Uso: Excluir local**

Nome do Caso de Uso: Excluir local	**Funcionalidade:** Lista de Locais
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Exclui um local cadastrado.	
PRÉ-CONDIÇÃO	
1- Local cadastrado.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Informa o local;	
2- Se o local for encontrado, excluir;	
3- Confirma a exclusão.	
4- Caso contrário, finaliza.	
PÓS-CONDIÇÃO	
1- Local excluído.	

Tabela 21: Excluir local

- **Caso de Uso: Excluir produto**

Nome do Caso de Uso: Excluir produto	**Funcionalidade:** Lista de Produtos
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Exclui um produto cadastrado.	
PRÉ-CONDIÇÃO	
1- Produto cadastrado.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Informa o produto;	
2- Se o produto for encontrado, excluir;	
3- Confirma a exclusão.	
4- Caso contrário, finaliza.	
PÓS-CONDIÇÃO	
1- Produto excluído.	

Tabela 22: Excluir produto

- **Caso de Uso: Excluir impedimento**

Nome do Caso de Uso: Excluir impedimento	**Funcionalidade:** Lista de Impedimentos
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Exclui um impedimento cadastrado.	
PRÉ-CONDIÇÃO	
1- Impedimento cadastrado.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Informa um impedimento;	
2- Se o impedimento for encontrado, excluir;	
3- Confirma a exclusão.	
3- Caso contrário, finaliza.	
PÓS-CONDIÇÃO	
1- Impedimento excluído.	

Tabela 23: Excluir impedimento

- **Caso de Uso: Excluir banco**

Nome do Caso de Uso: Excluir banco	**Funcionalidade:** Lista de Bancos
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Exclui um banco cadastrado.	
PRÉ-CONDIÇÃO	
1- Banco cadastrado.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Informa o banco;	
2- Se o banco for encontrado, excluir;	
3- Confirmar a exclusão.	
4- Caso contrário, finaliza.	
PÓS-CONDIÇÃO	
1- Banco excluído.	

Tabela 24: Excluir banco

- **Caso de Uso: Excluir fornecedor**

Nome do Caso de Uso: Excluir fornecedor	**Funcionalidade:** Lista de Fornecedores
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Exclui um fornecedor cadastrado.	
PRÉ-CONDIÇÃO	
1- Fornecedor cadastrado.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Informa o fornecedor;	
2- Se o fornecedor for encontrado, excluir;	
3- Confirmar a exclusão.	
3- Caso contrário, finaliza.	
PÓS-CONDIÇÃO	
1- Fornecedor excluído.	

Tabela 25: Excluir fornecedor

- **Caso de Uso: Excluir usuario**

Nome do Caso de Uso: Excluir usuario	**Funcionalidade:** Ficha de Usuarios
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Exclui um usuário cadastrado.	
PRÉ-CONDIÇÃO	
1- Usuário cadastrado.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Informa o usuário;	
2- Se o usuário for encontrado, excluir;	
3- Confirma a exclusão.	
3- Caso contrário, finaliza.	
PÓS-CONDIÇÃO	
1- Usuário excluído.	

Tabela 26: Excluir usuario

- **Caso de Uso: Excluir compromisso**

Nome do Caso de Uso: Excluir compromisso	**Funcionalidade:** Lista de compromissos
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Exclui um compromisso cadastrado.	
PRÉ-CONDIÇÃO	
1- Compromisso cadastrado.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Informa o compromisso;	
2- Se o compromisso for encontrado, exclui;	
3- Confirma a exclusão.	
4- Caso contrário, finaliza.	
PÓS-CONDIÇÃO	
1- Fornecedor excluído.	

Tabela 27: Excluir compromisso

- **Caso de Uso: Excluir conta a pagar**

Nome do Caso de Uso: Excluir conta a pagar	**Funcionalidade:** Lista de Contas a Pagar
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Exclui uma conta a pagar cadastrada.	
PRÉ-CONDIÇÃO	
1- Conta a Pagar cadastrada.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Informa a conta a pagar cadastrada;	
2- Se a conta a pagar for encontrada, excluir;	
3- Confirmar exclusão.	
4- Caso contrário, finaliza.	
PÓS-CONDIÇÃO	
1- Conta a pagar excluída.	

Tabela 28: Excluir conta a pagar

- **Caso de Uso: Excluir conta a receber**

Nome do Caso de Uso: Excluir conta a receber	**Funcionalidade:** Lista de Contas a Receber
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Exclui uma conta a receber cadastrada.	
PRÉ-CONDIÇÃO	
1- Conta a receber cadastrada.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Informa a conta a receber cadastrada;	
2- Se a conta a receber for encontrada, excluir.	
3- Confirmar exclusão.	
4- Caso contrário, finaliza.	
PÓS-CONDIÇÃO	
1- Conta a Receber excluída.	

Tabela 29: Excluir conta a receber

- **Caso de Uso: Excluir cliente**

Nome do Caso de Uso: Excluir cliente	**Funcionalidade:** Ficha de Clientes
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Exclui um cliente cadastrado.	
PRÉ-CONDIÇÃO	
1- Cliente cadastrado.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Informa o cliente;	
2- Se o cliente for encontrado, excluir;	
3- Confirmar exclusão.	
4- Caso contrário, finaliza.	
PÓS-CONDIÇÃO	
1- Cliente excluído.	

Tabela 30: Excluir cliente

- **Caso de Uso: Consultar clientes**

Nome do Caso de Uso: Consultar clientes	**Funcionalidade:** Ficha de Clientes
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Consulta todos os clientes cadastrados.	
PRÉ-CONDIÇÃO	
1- Clientes cadastrados e adicionados na base de dados.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Consulta os clientes.	
PÓS-CONDIÇÃO	
1- Clientes exibidos.	

Tabela 31: Consultar clientes

- **Caso de Uso: Consultar categorias**

Nome do Caso de Uso: Consultar categoria	**Funcionalidade:** Lista de Categorias
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Consulta todas as categorias.	
PRÉ-CONDIÇÃO	
1- Categorias cadastradas e armazenadas na base de dados.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Consulta as categorias.	
PÓS-CONDIÇÃO	
1- Categorias exibidas.	

Tabela 32: Consultar categorias

- **Caso de Uso: Consultar professores**

Nome do Caso de Uso: Consultar professores	**Funcionalidade:** Ficha de Professores
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Consulta todos os professores cadastrados.	
PRÉ-CONDIÇÃO	
1- Professores cadastrados e adicionados na base de dados.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Consulta os professores.	
PÓS-CONDIÇÃO	
1- Professores exibidos.	

Tabela 33: Consultar professores

- **Caso de Uso: Consultar atividades**

Nome do Caso de Uso: Consultar atividades	**Funcionalidade:** Lista de Atividades
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Consulta todas as atividades cadastradas.	
PRÉ-CONDIÇÃO	
1- Atividades cadastradas e adicionadas na base de dados.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Consulta as atividades.	
PÓS-CONDIÇÃO	
1- Atividades exibidas.	

Tabela 34: Consultar atividades

- **Caso de Uso: Consultar locais**

Nome do Caso de Uso: Consultar locais	**Funcionalidade:** Lista de Locais
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Consulta todos os locais cadastrados.	
PRÉ-CONDIÇÃO	
1- Locais cadastrados e adicionados na base de dados.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Consulta todos os locais.	
PÓS-CONDIÇÃO	
1- Locais exibidos.	

Tabela 35: Consultar locais

- **Caso de Uso: Consultar produtos**

Nome do Caso de Uso: Consultar produtos	**Funcionalidade:** Lista de Produtos
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Consulta todos os produtos.	
PRÉ-CONDIÇÃO	
1- Produtos cadastrados e adicionados na base de dados.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Consulta todos os produtos;	
PÓS-CONDIÇÃO	
1- Produtos exibidos.	

Tabela 36: Consultar produtos

- **Caso de Uso: Consultar impedimentos**

Nome do Caso de Uso: Consultar impedimentos	**Funcionalidade:** Lista de Impedimentos
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Consulta todos os impedimentos cadastrados.	
PRÉ-CONDIÇÃO	
1- Impedimentos cadastrados e adicionados na base de dados.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Consulta todos os impedimentos.	
PÓS-CONDIÇÃO	
1- Impedimentos exibidos.	

Tabela 37: Consultar impedimentos

- **Caso de Uso: Consultar bancos**

Nome do Caso de Uso: Consultar bancos	**Funcionalidade:** Lista de Bancos
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Consulta todos os bancos cadastrados.	
PRÉ-CONDIÇÃO	
1- Bancos cadastrados e adicionados na base de dados.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Consulta os bancos.	
PÓS-CONDIÇÃO	
1- Bancos exibidos.	

Tabela 38: Consultar bancos

- **Caso de Uso: Consultar fornecedores**

Nome do Caso de Uso: Consultar fornecedores	**Funcionalidade:** Ficha de Fornecedores
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Consulta todos os fornecedores cadastrados.	
PRÉ-CONDIÇÃO	
1- Fornecedores cadastrados e adicionados na base de dados.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Consulta os fornecedores.	
PÓS-CONDIÇÃO	
1- Fornecedores exibidos.	

Tabela 39: Consultar fornecedores

- **Caso de Uso: Consultar usuarios**

Nome do Caso de Uso: Consultar usuarios	**Funcionalidade:** Ficha de Usuarios
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Consulta todos os usuários cadastrados.	
PRÉ-CONDIÇÃO	
1- Usuários cadastrados e adicionados na base de dados.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Consulta os usuários.	
PÓS-CONDIÇÃO	
1- Usuários exibidos.	

Tabela 40: Consultar usuarios

- **Caso de Uso: Consultar compromissos**

Nome do Caso de Uso: Consultar compromissos	**Funcionalidade:** Lista de compromissos.
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Consulta todos os compromissos cadastrados.	
PRÉ-CONDIÇÃO	
1- Compromissos cadastrados e adicionados na base de dados.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Consulta os compromissos.	
PÓS-CONDIÇÃO	
1- Fornecedores exibidos.	

Tabela 41: Consultar compromissos

- **Caso de Uso: Consultar contas a pagar**

Nome do Caso de Uso: Consultar contas a pagar	**Funcionalidade:** Lista de Contas a Pagar
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Consulta todas as contas a pagar cadastradas.	
PRÉ-CONDIÇÃO	
1- Contas a Pagar cadastradas e adicionadas na base de dados.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Consulta as contas a pagar.	
PÓS-CONDIÇÃO	
1- Contas a pagar exibidas.	

Tabela 42: Consultar contas a pagar

- **Caso de Uso: Consultar contas a receber**

Nome do Caso de Uso: Consultar contas a receber	**Funcionalidade:** Lista de Contas a Receber
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Consulta todas as contas a receber cadastradas.	
PRÉ-CONDIÇÃO	
1- Contas a receber cadastradas e adicionadas na base de dados.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Consulta as contas a receber.	
PÓS-CONDIÇÃO	
1- Contas a Receber exibidas.	

Tabela 43: Consultar contas a receber

- **Caso de Uso: Consultar contas corrente**

Nome do Caso de Uso: Consultar contas correntes	**Funcionalidade:** Lista de Contas Correntes
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Consulta todas as contas correntes cadastradas.	
PRÉ-CONDIÇÃO	
1- Contas correntes cadastradas e adicionadas na base de dados.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Consulta as contas correntes.	
PÓS-CONDIÇÃO	
1- Contas Correntes exibidas.	

Tabela 44: Consultar contas correntes

- **Caso de Uso: Consulta compromissos agendados**

Nome do Caso de Uso: Consultar compromissos agendados	**Funcionalidade:** Lista de Compromissos
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Consulta todos os compromissos agendados.	
PRÉ-CONDIÇÃO	
1- Compromissos agendados e adicionados na base de dados.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Consulta os compromissos agendados.	
PÓS-CONDIÇÃO	
1- Compromissos agendados.	

Tabela 45: Consultar compromissos agendados

- **Caso de Uso: Matricular cliente**

Nome do Caso de Uso: Matricular cliente	**Funcionalidade:** Ficha de Clientes
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Matricula um cliente em alguma atividade da academia.	
PRÉ-CONDIÇÃO	
1- Cliente cadastrado;	
2- Atividade cadastrada;	
3- Vagas na atividade disponíveis;	
4- Verifica a situação do cliente.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Localiza o cliente;	
2- Se o cliente estiver cadastrado, matricula;	
3- Caso contrário, finaliza.	
PÓS-CONDIÇÃO	
1- Cliente matriculado.	

Tabela 46: Matricular cliente

- **Caso de Uso: Verificar situacao do cliente**

Nome do Caso de Uso: Verificar situacao do cliente	**Funcionalidade:** Ficha de Clientes
Atores Envolvidos: Administrador	
Tipo: Secundário	
Descrição: Verifica se o cliente tem algum motivo que o impeça de continuar freqüentando a academia normalmente.	
PRÉ-CONDIÇÃO	
1- Cliente cadastrado.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Localiza o cliente;	
2- Verifica a situação do cliente.	
PÓS-CONDIÇÃO	
1- Situação apresentada.	

Tabela 47: Verificar situação cliente

- **Caso de Uso: Cancelar matricula do cliente**

Nome do Caso de Uso: Cancelar matricula do cliente	**Funcionalidade:** Ficha de Clientes
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Cancela a matrícula de um cliente em alguma atividade física.	
PRÉ-CONDIÇÃO	
1- Cliente matriculado na atividade.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Localiza o cliente;	
2- Cancela a matrícula;	
3- Confirma o cancelamento.	
PÓS-CONDIÇÃO	
1- Matrícula cancelada.	

Tabela 48: Cancelar matricula

- **Caso de Uso: Mostrar historico do cliente**

Nome do Caso de Uso: Mostrar histórico do cliente	**Funcionalidade:** Ficha de Clientes
Atores Envolvidos: Administrador	
Tipo: Secundário	
Descrição: Apresenta a situação de cada cliente dentro da academia, tanto no módulo referente a matrícula quanto na parte financeira, podendo saber o que cada cliente pagou ou deverá pagar.	
PRÉ-CONDIÇÃO	
1- Cliente cadastrado.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Informa o cliente.	
PÓS-CONDIÇÃO	
1- Apresenta histórico do cliente.	

Tabela 49: Mostrar historico do cliente

- **Caso de Uso: Vender produto**

Nome do Caso de Uso: Vender produto	**Funcionalidade:** Lista de Produtos
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: Realiza uma operação de venda simples a um cliente da academia.	
PRÉ-CONDIÇÃO	
1- Cliente cadastrado.	
2- Produto cadastrado.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Informa o cliente;	
2- Informa o produto.	
PÓS-CONDIÇÃO	
1- Venda efetuada.	

Tabela 50: Vender produto

- **Caso de Uso: Informar recebimento**

Nome do Caso de Uso: Informar recebimento	**Funcionalidade:** Lista de Contas a Receber
Atores Envolvidos: Administrador	
Tipo: Secundário	
Descrição: Informa o recebimento de uma conta.	
PRÉ-CONDIÇÃO	
1- Conta adicionada na base de dados.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Informa a conta recebida;	
2- Informa o recebimento da conta.	
3- Caso contrário, finaliza.	
PÓS-CONDIÇÃO	
1- Conta recebida.	

Tabela 51: Informar recebimento

- **Caso de Uso: Informar pagamento**

Nome do Caso de Uso: Informar pagamento	**Funcionalidade:** Lista de Contas a Pagar
Atores Envolvidos: Administrador	
Tipo: Secundário	
Descrição: Informa o pagamento de uma conta paga.	
PRÉ-CONDIÇÃO	
1- Conta adicionada na base de dados.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Informa a conta paga;	
2- Informa o pagamento da conta.	
3- Caso contrário, finaliza.	
PÓS-CONDIÇÃO	
1- Conta paga.	

Tabela 52: Informar pagamento

- **Caso de Uso: Agendar compromisso**

Nome do Caso de Uso: Agendar compromisso	**Funcionalidade:** Lista de Compromissos
Atores Envolvidos: Administrador	
Tipo: Secundário	
Descrição: Controla a data e horário de compromissos dos clientes da academia.	
PRÉ-CONDIÇÃO	
1- Cliente cadastrado.	
2- Compromisso cadastrado.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Informa o cliente;	
2- Informa o compromisso.	
3- Caso contrário, fim.	
PÓS-CONDIÇÃO	
1- Compromisso agendado.	

Tabela 53: Agendar compromisso

- **Caso de Uso: Cancelar compromisso agendado**

Nome do Caso de Uso: Cancelar compromisso agendado	**Funcionalidade:** Lista de Compromissos
Atores Envolvidos: Administrador	
Tipo: Secundário	
Descrição: Cancela um compromisso marcado na agenda de compromisso.	
PRÉ-CONDIÇÃO	
1- Compromisso agendado;	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Informa o compromisso na Agenda de Compromissos;	
2- Cancelar o compromisso;	
3- Confirma o cancelamento.	
PÓS-CONDIÇÃO	
1- Compromisso agendado.	

Tabela 54: Cancelar compromisso

- **Caso de Uso: Controlar aulas dadas**

Nome do Caso de Uso: Controlar aulas dadas	**Funcionalidade:** Lista de Professores
Atores Envolvidos: Administrador	
Tipo: Secundário	
Descrição: Controla aulas que cada professor dará no mês corrente tanto por atividade ou por categoria, sendo assim o administrador saberá o número de aulas dadas por cada professor no mês, podendo ser calculado o pagamento do professor.	
PRÉ-CONDIÇÃO	
1- Dados da aula disponíveis;	
2- Professor cadastrado.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Informa o professor;	
2- Informa os dados.	
3- Caso contrário, fim.	
PÓS-CONDIÇÃO	
1- Controle realizado. 2- Calcula pagamento.	

Tabela 55: Controlar aulas dadas

- **Caso de Uso: Emitir relatorio da agenda de compromissos**

Nome do Caso de Uso: Emitir relatorio da agenda de compromissos	**Funcionalidade:** Relatorio
Atores Envolvidos: Administrador	
Tipo: Terciário	
Descrição: Imprimi uma listagem com todos os compromissos agendados agrupados por data e por horário.	
PRÉ-CONDIÇÃO	
1- Agenda de compromissos selecionada;	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Imprimir relatório da agenda de compromissos.	
PÓS-CONDIÇÃO	
1- Agenda de compromissos impressa.	

Tabela 56: Emitir relatorio da agenda de compromissos

- **Caso de Uso: Emitir relatorio dos aniversariantes**

Nome do Caso de Uso: Emitir relatorio dos aniversariantes	**Funcionalidade:** Relatorios
Atores Envolvidos: Administrador	
Tipo: Terciário	
Descrição: Imprimi uma listagem com todos os clientes aniversariantes referentes ao mês desejado.	
PRÉ-CONDIÇÃO	
1- Mês selecionado.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Imprimir relatório dos clientes aniversariantes.	
PÓS-CONDIÇÃO	
1- Relatório dos clientes aniversariantes impresso.	

Tabela 57: Emitir relatorio dos aniversariantes

- **Caso de Uso: Emitir relatorio das contas a pagar**

Nome do Caso de Uso: Emitir relatorio das contas a pagar	**Funcionalidade:** Relatorios
Atores Envolvidos: Administrador	
Tipo: Terciário	
Descrição: Imprimi uma listagem com todas as contas a pagar ou já pagas pela academia.	
PRÉ-CONDIÇÃO	
1- Contas a Pagar cadastradas.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Imprimir relatórios de Contas a Pagar.	
PÓS-CONDIÇÃO	
1- Relatório de Contas a Pagar impresso.	

Tabela 58: Emitir relatorio das contas a pagar

- **Caso de Uso: Emitir relatorio das contas a receber**

Nome do Caso de Uso: Emitir relatorio das contas a receber	**Funcionalidade:** Relatorios
Atores Envolvidos: Administrador	
Tipo: Terciário	
Descrição: Imprimi uma listagem com todas as contas a receber dos clientes ou já recebidas pela academia.	
PRÉ-CONDIÇÃO	
1- Contas a Receber cadastradas.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Imprimir relatório de Contas a Receber.	
PÓS-CONDIÇÃO	
1- Relatório de Contas a Receber impressa.	

Tabela 59: Emitir relatorio das contas a receber

- **Caso de Uso: Emitir relatorio das atividades**

Nome do Caso de Uso: Emitir relatorio das atividades	**Funcionalidade:** Relatorios
Atores Envolvidos: Administrador	
Tipo: Terciário	
Descrição: Imprimi uma listagem com todas as atividades que a academia oferece.	
PRÉ-CONDIÇÃO	
1- Atividades cadastradas.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Imprimir o relatório de Atividades.	
PÓS-CONDIÇÃO	
1- Relatório das atividades impresso.	

Tabela 60: Emitir relatorio das atividades

- **Caso de Uso: Emitir relatorio dos clientes**

Nome do Caso de Uso: Emitir relatorio dos clientes	**Funcionalidade:** Relatorios
Atores Envolvidos: Administrador	
Tipo: Terciário	
Descrição: O administrador solicita a impressão do relatório de todos os clientes do cadastro, podendo ter opções como imprimir todos os clientes do cadastro, por ordem alfabética, clientes que estejam com exames vencidos, com matrículas vencidas e não renovadas, clientes que tem acesso impedido por algum motivo, matriculados referente a atividade selecionada, matriculados referente a categoria selecionada e clientes inativos.	
PRÉ-CONDIÇÃO	
1- Clientes cadastrados.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Imprimir o relatorio de Clientes.	
PÓS-CONDIÇÃO	
1- Relatório de clientes impresso.	

Tabela 61: Emitir relatorio dos clientes

- **Caso de Uso: Emitir relatorio da folha de pagamento**

Nome do Caso de Uso: Emitir relatorio da folha de pagamento	**Funcionalidade:** Relatorios
Atores Envolvidos: Administrador	
Tipo: Terciário	
Descrição: Imprimi um relatório da folha de pagamento dos professores de determinado mês.	
PRÉ-CONDIÇÃO	
1- Professores cadastrados.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Imprimir o relatorio da folha de pagamento.	
PÓS-CONDIÇÃO	
1- Relatório da folha de pagamento impresso.	

Tabela 62: Emitir relatorio da folha de pagamento

- **Caso de Uso: Emitir relatorio do extrato de conta corrente**

Nome do Caso de Uso: Emitir relatorio do extrato de conta corrente.	**Funcionalidade:** Relatorios
Atores Envolvidos: Administrador	
Tipo: Terciário	
Descrição: Imprimi um relatório do extrato de uma conta corrente de acordo com o período informado.	
PRÉ-CONDIÇÃO	
1- Conta corrente cadastrada no relatório,.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Imprimir o relatorio do extrato de conta corrente;	
PÓS-CONDIÇÃO	
1- Relatório da conta corrente impresso.	

Tabela 63: Emitir relatorio do extrato de conta corrente

- **Caso de Uso: Visualizar grafico**

Nome do Caso de Uso: Visualizar grafico	**Funcionalidade:** Graficos
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: O administrador poderá visualizar um gráfico das despesas por categoria (analisando a despesa de cada categoria), das matrículas por categoria (analisando qual categoria mais procurada) ou dos códigos de impedimento (verificando quais os maiores impedimentos).	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Informa a opção.	
PÓS-CONDIÇÃO	
1- Gráfico visualizado.	

Tabela 64: Visualizar grafico

- **Caso de Uso: Transferir fundos**

Nome do Caso de Uso: Transferir fundos	**Funcionalidade:** Lista de Contas Correntes
Atores Envolvidos: Administrador	
Tipo: Terciário	
Descrição: O administrador poderá fazer transferência entre contas correntes.	
PRÉ-CONDIÇÃO	
1- Contas correntes cadastradas.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Transferir fundos entre duas contas correntes.	
PÓS-CONDIÇÃO	
1- Transferido	

Tabela 65: Transferir fundos

- **Caso de Uso: Movimentar conta corrente**

Nome do Caso de Uso: Movimentar conta corrente	**Funcionalidade:** Lista de Contas Correntes
Atores Envolvidos: Administrador	
Tipo: Terciário	
Descrição: O administrador poderá realizar um lançamento em conta corrente.	
PRÉ-CONDIÇÃO	
1- Conta corrente cadastrada.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Realizar o lançamento.	
PÓS-CONDIÇÃO	
1- Processar.	

Tabela 66: Movimentar conta corrente

- **Caso de Uso: Consultar contas correntes**

Nome do Caso de Uso: Consultar contas correntes	**Funcionalidade:** Lista de Contas Correntes
Atores Envolvidos: Administrador	
Tipo: Secundário	
Descrição: Consulta todas as contas correntes cadastradas.	
PRÉ-CONDIÇÃO	
1- Contas correntes cadastradas e adicionadas na base de dados.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Consulta as contas correntes.	
PÓS-CONDIÇÃO	
1- Contas Correntes exibidas.	

Tabela 67: Consultar contas correntes

- **Caso de Uso: Impedir acesso do cliente**

Nome do Caso de Uso: Impedir acesso do cliente	**Funcionalidade:** Ficha de Clientes
Atores Envolvidos: Administrador	
Tipo: Primário	
Descrição: O administrador impede que o cliente tenha acesso a academia, cadastrando no formulário de situação do cliente um impedimento.	
PRÉ-CONDIÇÃO	
1- Cliente cadastrado;	
2- Impedimento cadastrado.	
SEQUÊNCIA TIPICA DE EVENTOS	
1- Informa o nome do cliente;	
2- Informa a descrição do impedimento;	
3- Armazena os dados na base de dados.	
PÓS-CONDIÇÃO	
1- Acesso impedido.	

Tabela 68: Impedir acesso do cliente

6.2 Modelagem dos diagramas

Esta sessão apresenta os resultados obtidos através da modelagem do funcionamento do sistema administrativo usando a ferramenta *Rational Rose 2000*.

6.2.1 Diagrama de caso de uso

As figuras a seguir apresentam os Diagramas de Caso de Uso descrevendo a funcionalidade do ato Administrador interagindo com os casos de uso.

Esta Figura apresenta o Diagrama de Caso de Uso da funcionalidade Ficha de Clientes, onde os casos de uso estão descritos nas tabelas 5, 30, 31, 46, 47, 48 e 49.

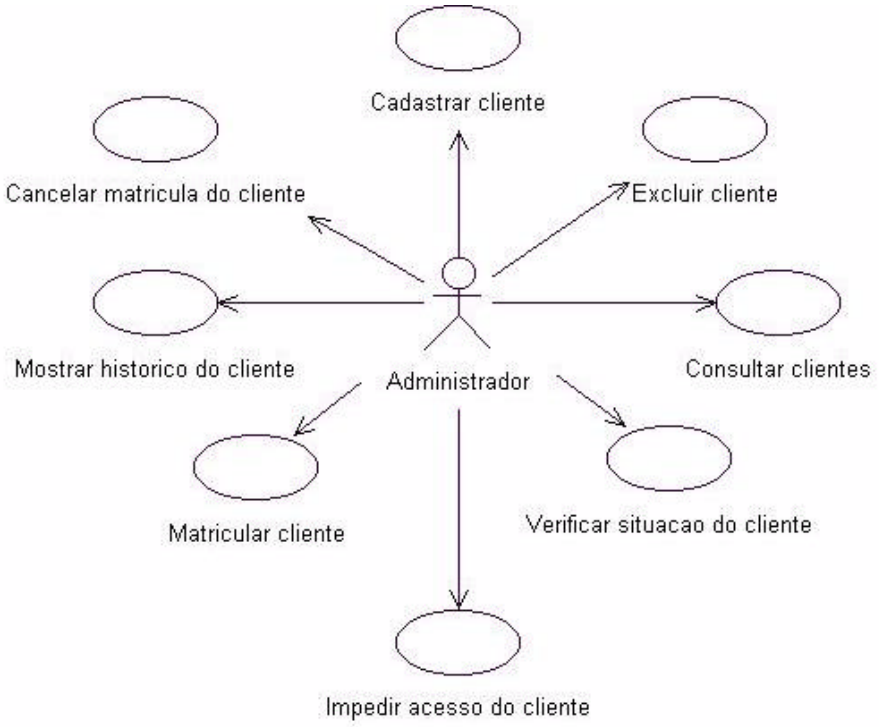


Figura 4: Diagrama de Caso de Uso da funcionalidade Ficha de Clientes

Esta Figura apresenta o Diagrama de Caso de Uso da funcionalidade Ficha de Professores, onde os casos de uso estão descritos nas tabelas 6, 19, 33 e 55.

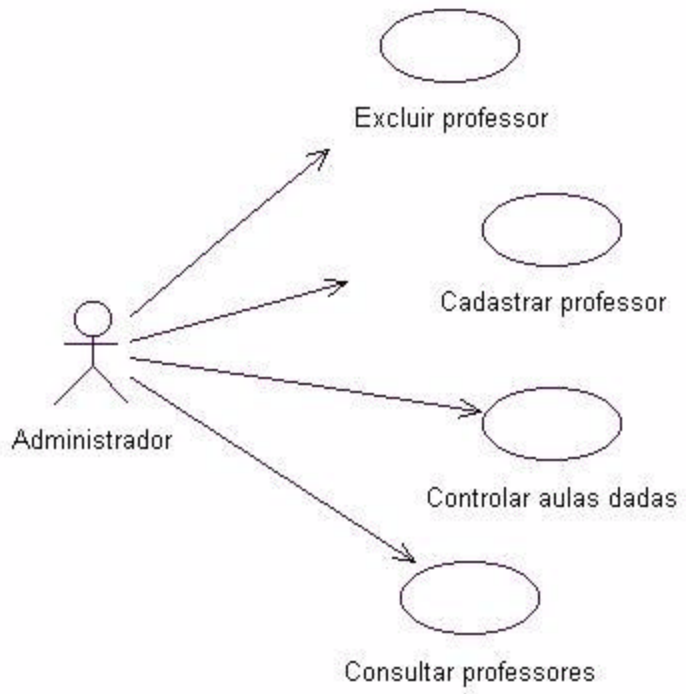


Figura 5: Diagrama de Caso de Uso da funcionalidade Ficha de Professores

Esta Figura apresenta o Diagrama de Caso de Uso da funcionalidade Lista de Categorias, onde os casos de uso estão descritos nas tabelas 17, 18 e 32.

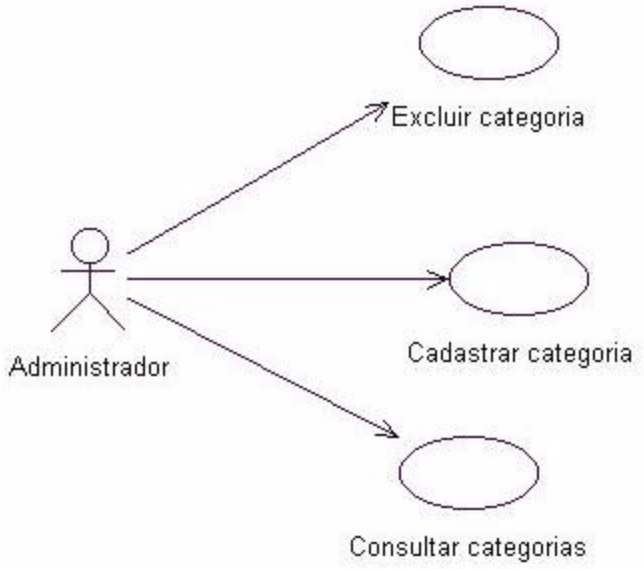


Figura 6: Diagrama de Caso de Uso da funcionalidade Lista de Categorias

Esta figura apresenta o Diagrama de Caso de Uso da funcionalidade Lista de Atividades, onde os casos de uso estão descritos nas tabelas 8, 20 e 34.

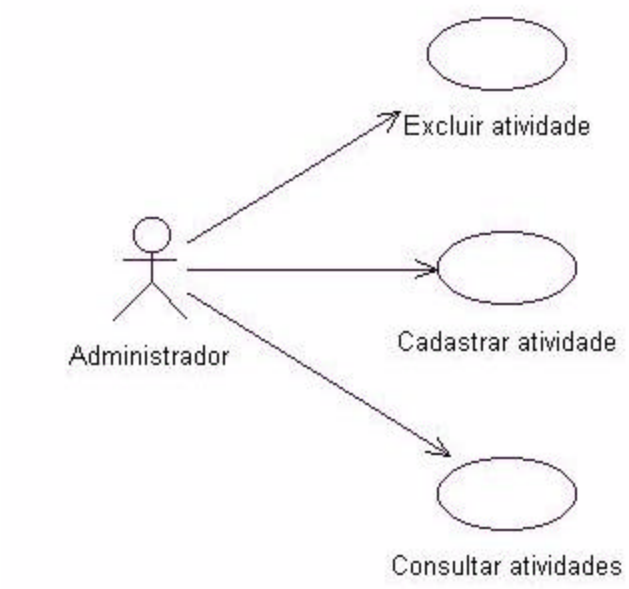


Figura 7: Diagrama de Caso de Uso da funcionalidade Lista de Atividades

Esta Figura apresenta o Diagrama de Caso de Uso da funcionalidade Lista de Impedimentos, onde os casos de uso estão descritos nas tabelas 10, 23 e 37.

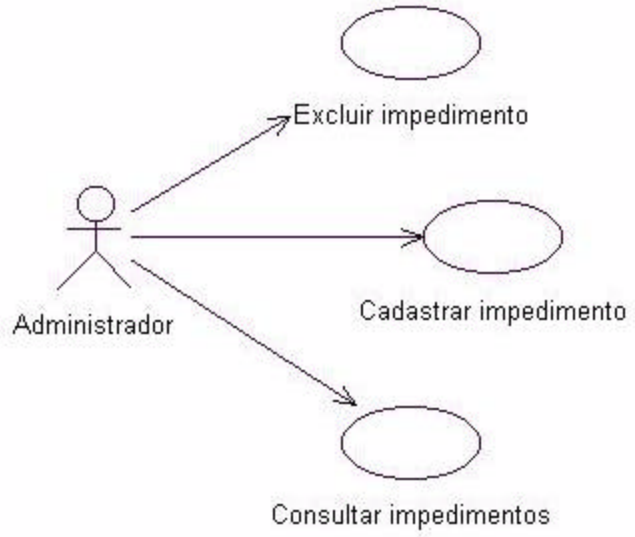


Figura 8: Diagrama de Caso de Uso da funcionalidade Lista de Impedimentos

Esta Figura apresenta o Diagrama de Caso de Uso da funcionalidade Lista de Bancos, onde os casos de uso estão descritos nas tabelas 11, 24 e 32.

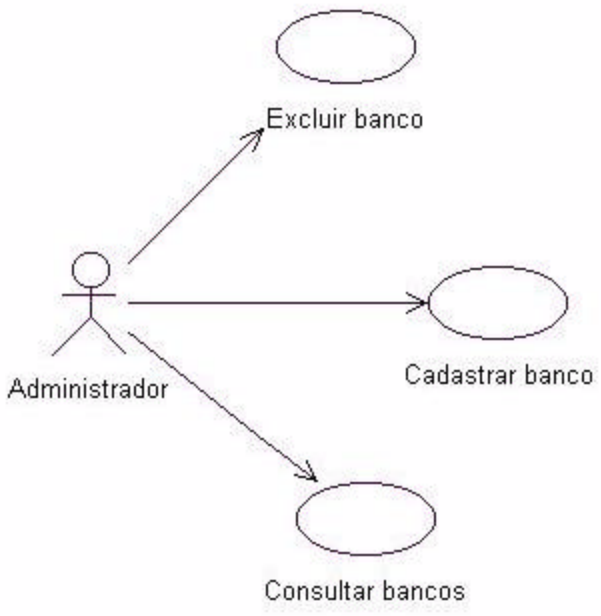


Figura 9: Diagrama de Caso de Uso da funcionalidade Lista de Bancos

Esta Figura apresenta o Diagrama de Caso de Uso da funcionalidade Ficha de Usuarios, onde os casos de uso estão descritos nas tabelas 13, 26 e 40.

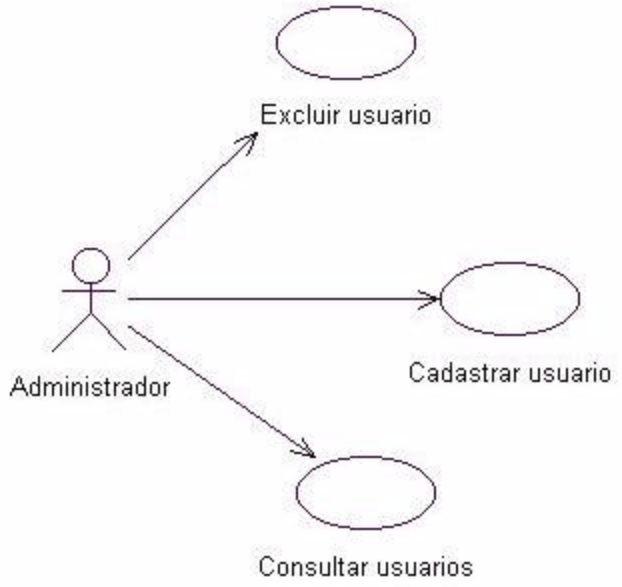


Figura 10: Diagrama de Caso de Uso da funcionalidade Ficha de Usuários

Esta Figura apresenta o Diagrama de Caso de Uso da funcionalidade Lista de Compromissos, onde os casos de uso estão descritos nas tabelas 14, 27, 41, 45, 53 e 54.

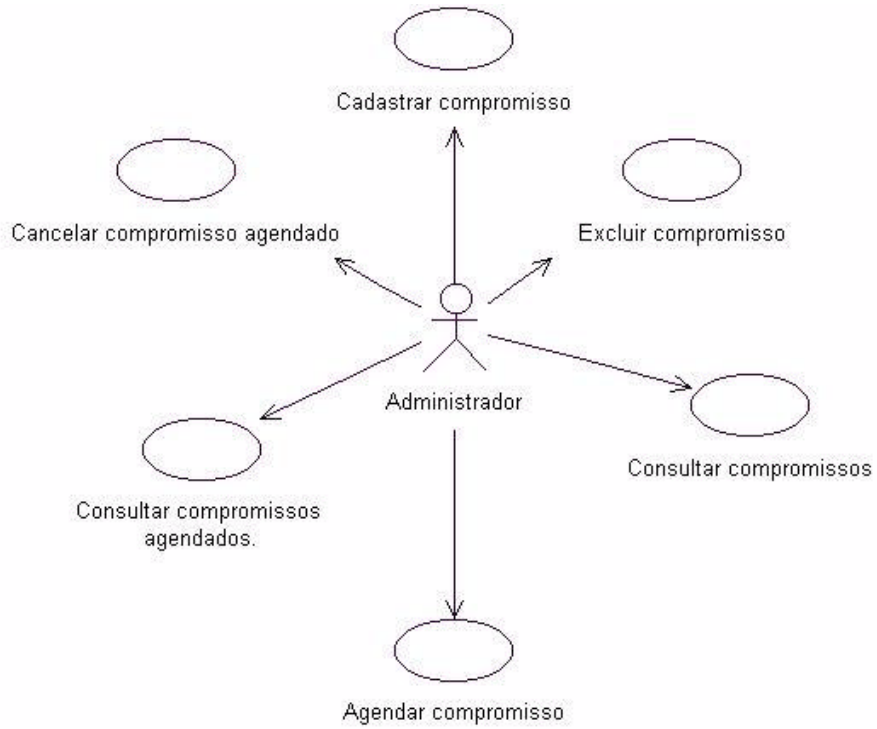


Figura 11: Diagrama de Caso de Uso da funcionalidade Lista de Compromissos

Esta Figura apresenta o Diagrama de Caso de Uso da funcionalidade Ficha de Fornecedores, onde os casos de uso estão descritos nas tabelas 12, 25 e 39.

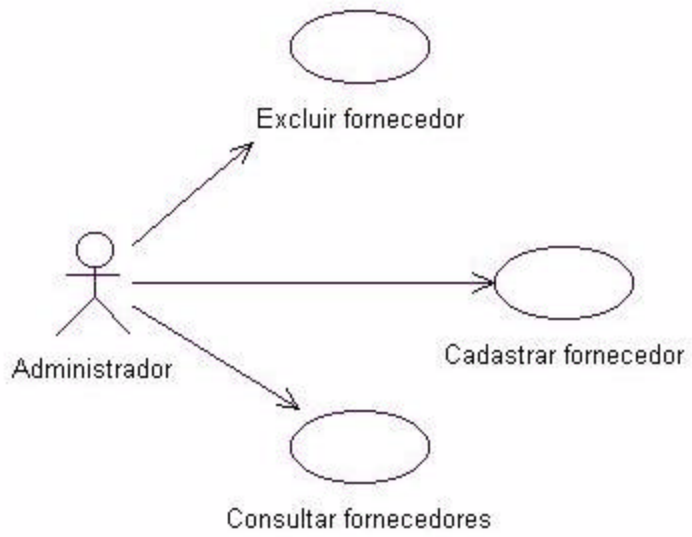


Figura 12: Diagrama de Caso de Uso da funcionalidade Ficha de Fornecedores

Esta Figura apresenta o Diagrama de Caso de Uso da funcionalidade Lista de Produtos, onde os casos de uso estão descritos nas tabelas 9, 22, 36 e 50.

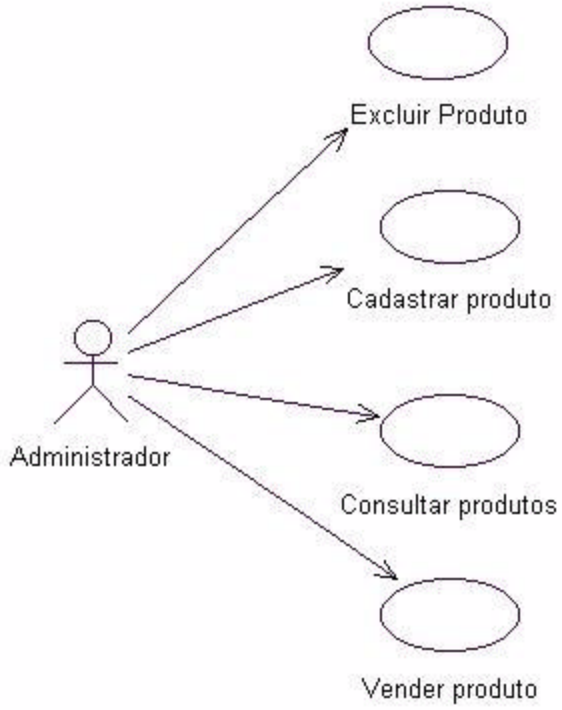


Figura 13: Diagrama de Caso de Uso da funcionalidade Lista de Produtos

Esta Figura apresenta o Diagrama de Caso de Uso da funcionalidade Lista de Locais, onde os casos de uso estão descritos nas tabelas 7, 21 e 35.

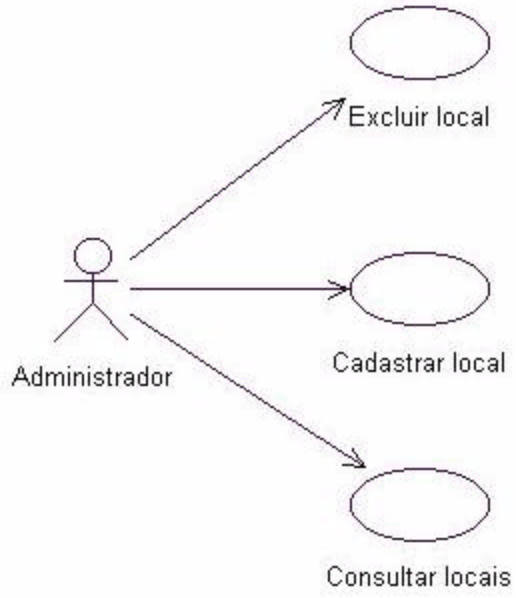


Figura 14: Diagrama de Caso de Uso da funcionalidade Lista de Locais

Esta Figura apresenta o Diagrama de Caso de Uso da funcionalidade Lista de Contas a Pagar, onde os casos de uso estão descritos nas tabelas 15, 28 e 42.

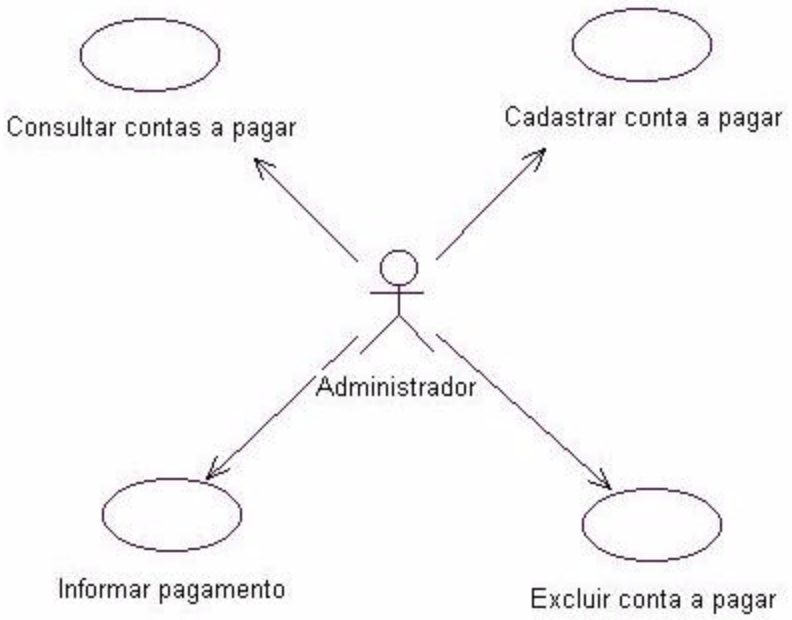


Figura 15: Diagrama de Caso de Uso da funcionalidade Lista de Contas a Pagar

Esta Figura apresenta o Diagrama de Caso de Uso da funcionalidade Lista de Contas a Receber, onde os casos de uso estão descritos nas tabelas 16, 28 e 43.

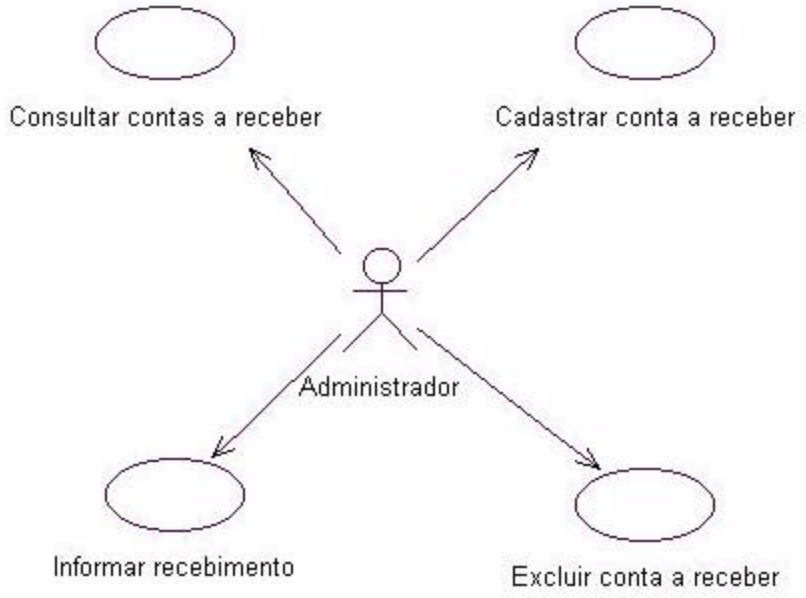


Figura 16: Diagrama de Caso de Uso da funcionalidade Lista de Contas a Receber

Esta Figura apresenta o Diagrama de Caso de Uso da funcionalidade Relatórios, onde os casos de uso estão descritos nas tabelas 56, 57, 58, 59, 60, 61, 62 e 63.

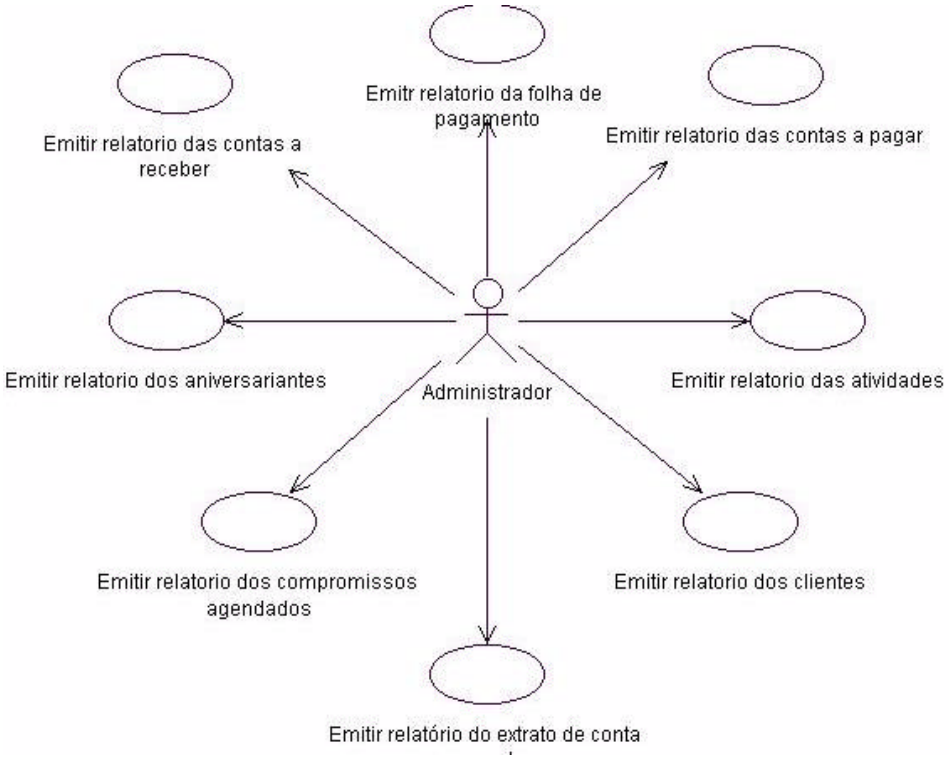


Figura 17: Diagrama de Caso de Uso da funcionalidade Relatórios

Esta Figura apresenta o Diagrama de Caso de Uso da funcionalidade Grafico, onde os casos de uso estão descritos na tabela 64.

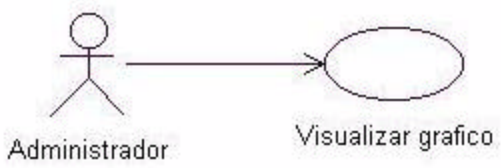


Figura 18: Diagrama de Caso de Uso da funcionalidade Grafico

Esta Figura apresenta o Diagrama de Caso de Uso da funcionalidade Lista de Contas Corrente, onde os casos de uso estão descritos na tabela 69.

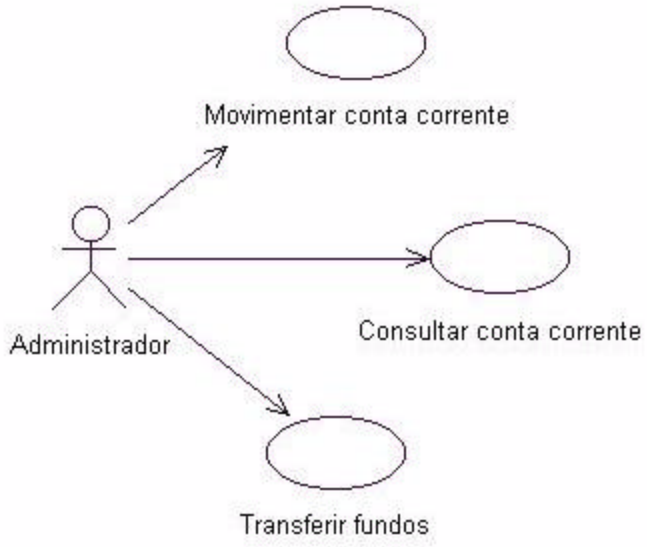


Figura 19: Diagrama de Caso de Uso da funcionalidade Lista de Contas Correntes

5.2.2 Diagrama de Seqüência

As Figuras a seguir apresentam os Diagramas de Seqüência desenvolvidos a fim de visualizar o cenário na ordem em que ele acontece em determinado tempo.

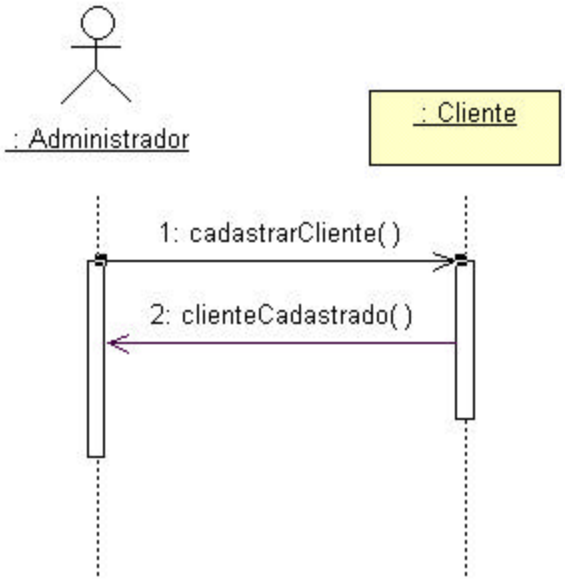


Figura 20: Diagrama de Seqüência Cadastrar cliente

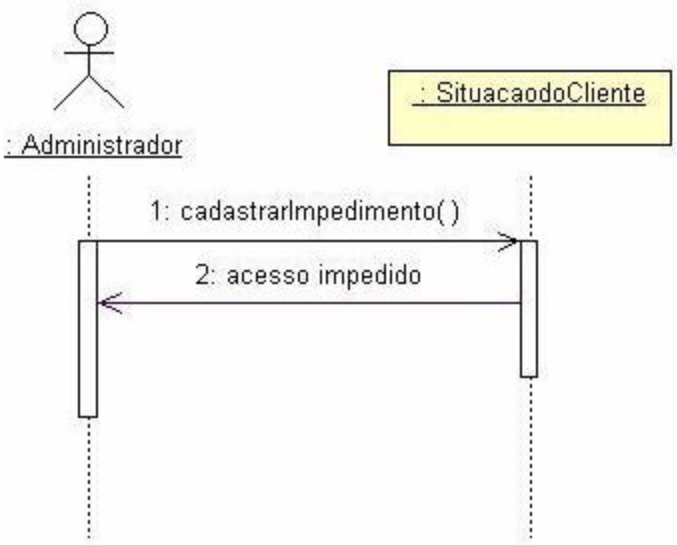


Figura 21: Diagrama de Seqüência Impedir acesso do cliente

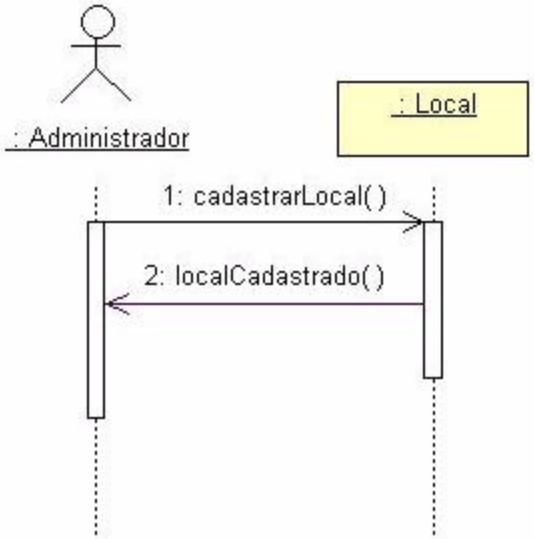


Figura 22: Diagrama de Seqüência Cadastrar local

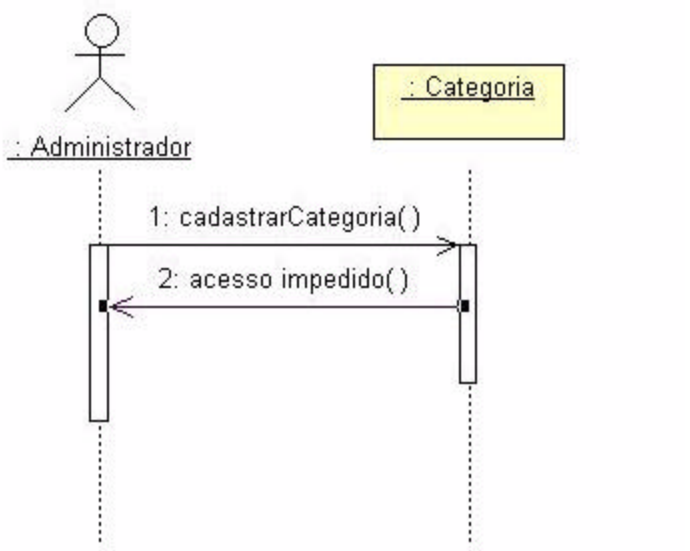


Figura 23: Diagrama de Seqüência Cadastrar categoria

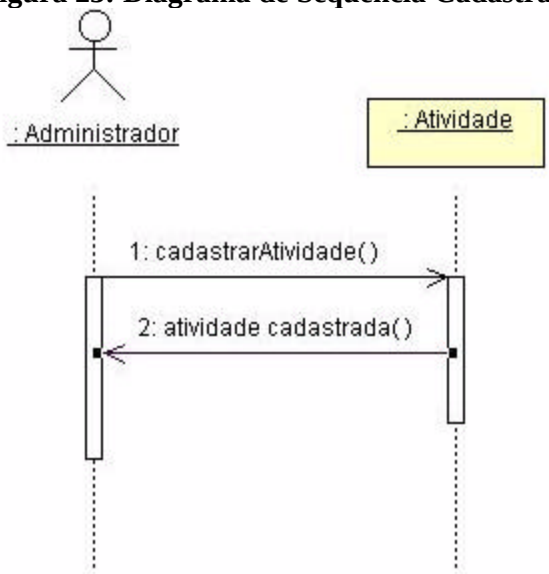


Figura 24: Diagrama de Seqüência Cadastrar atividade

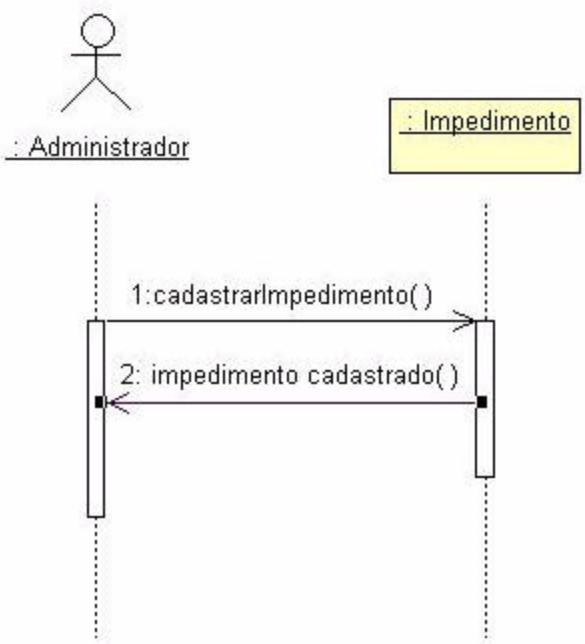


Figura 25: Diagrama de Seqüência Cadastrar impedimento

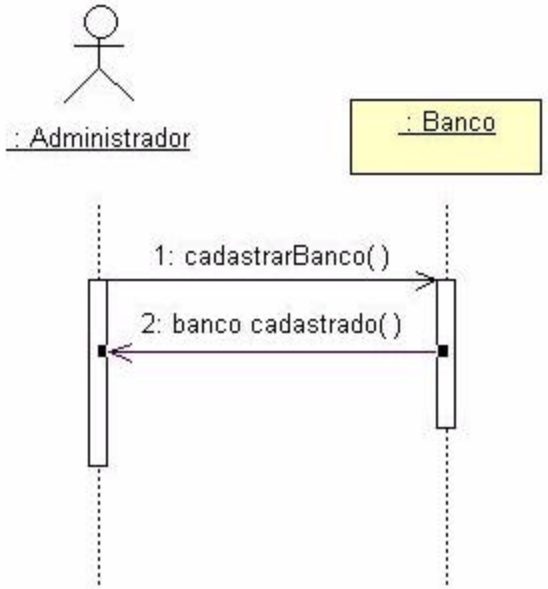


Figura 26: Diagrama de Seqüência Cadastrar banco

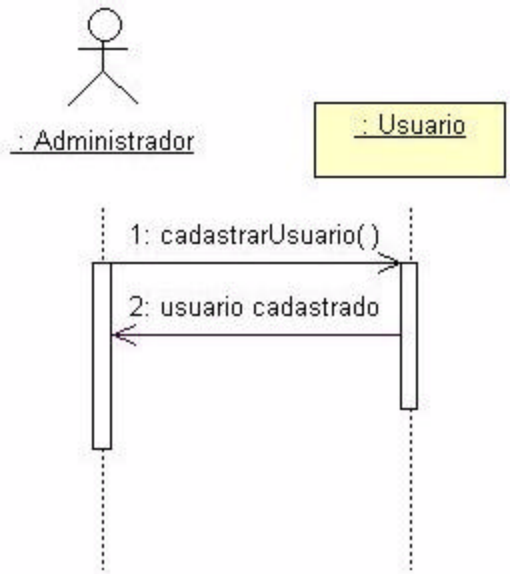


Figura 27: Diagrama de Seqüência Cadastrar usuário

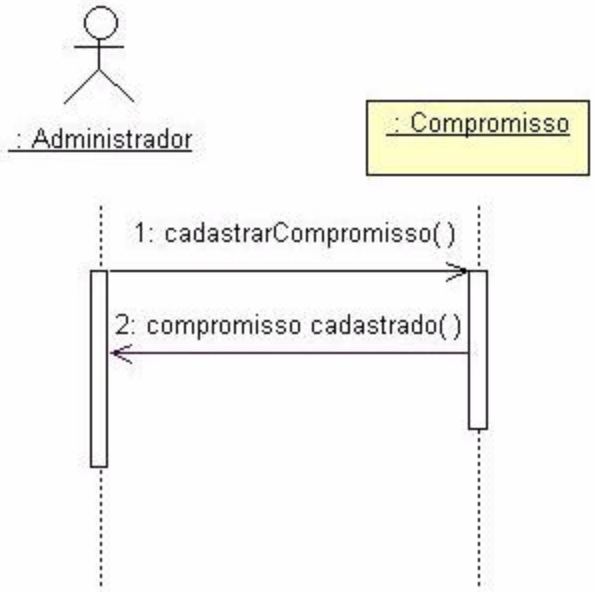


Figura 28: Diagrama de Seqüência Cadastrar compromisso

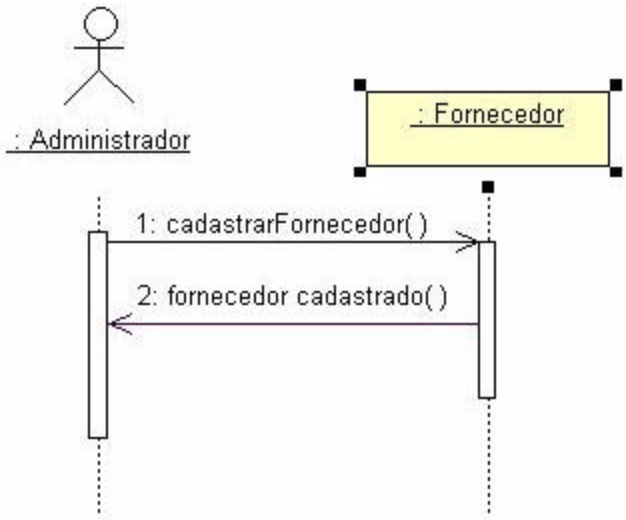


Figura 29: Diagrama de Seqüência Cadastrar fornecedor

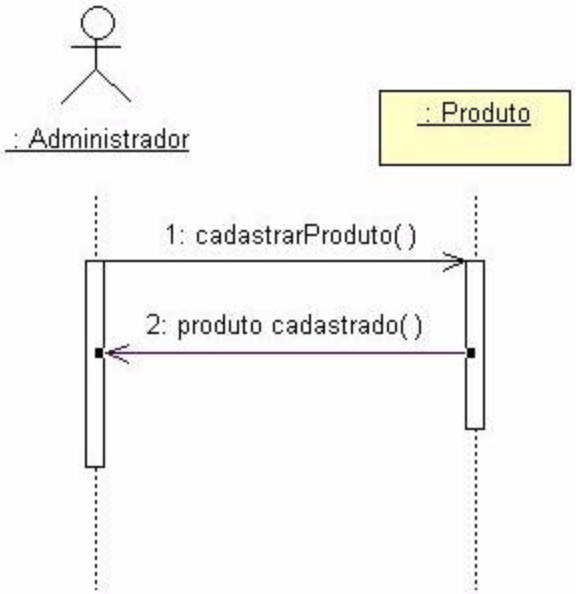


Figura 30: Diagrama de Seqüência Cadastrar produto

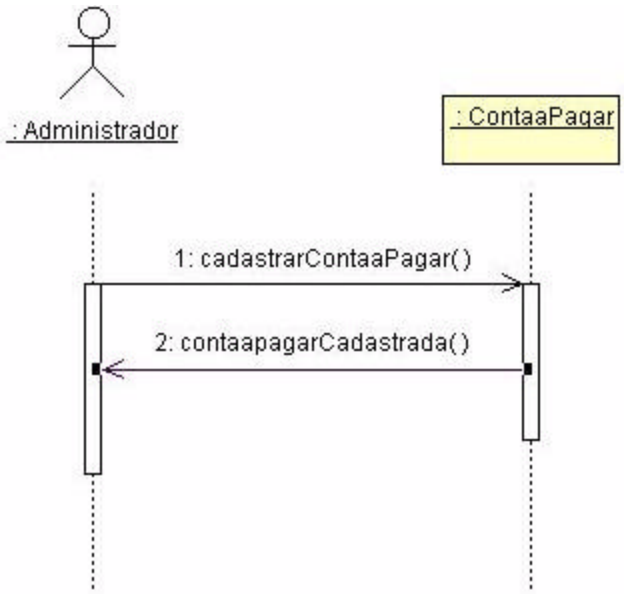


Figura 31: Diagrama de Seqüência Cadastrar conta a pagar

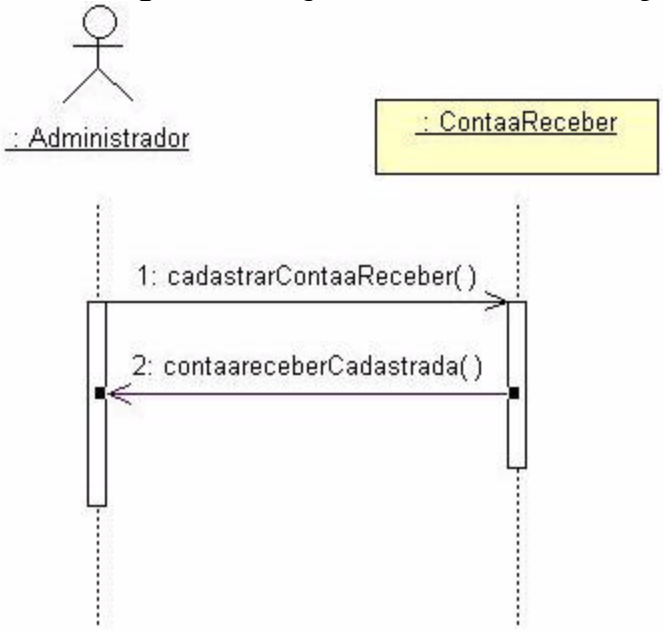


Figura 32: Diagrama de Seqüência Cadastrar conta a receber

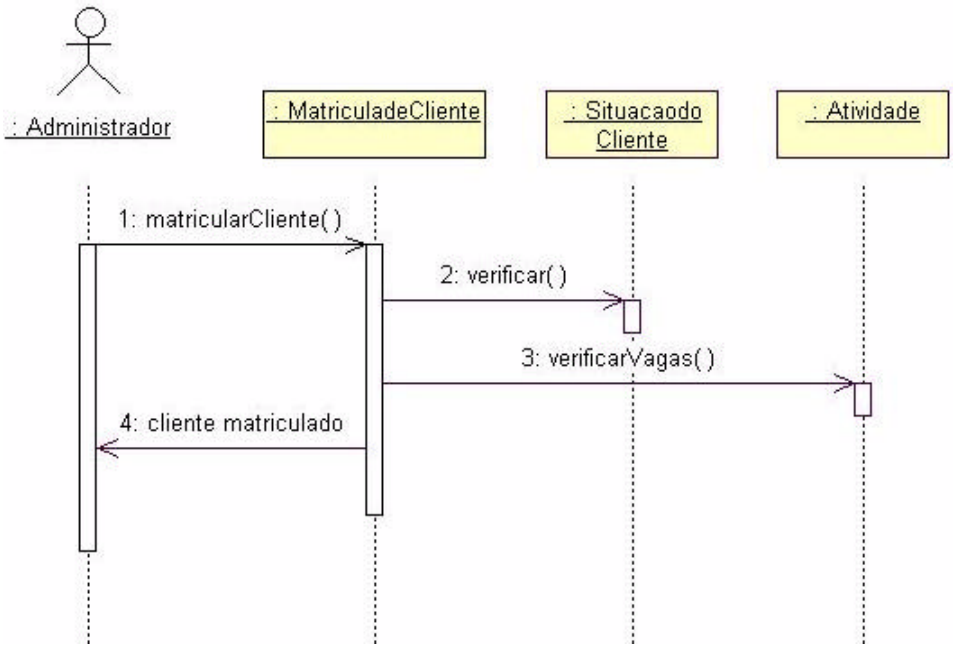


Figura 33: Diagrama de Seqüência Matricular cliente

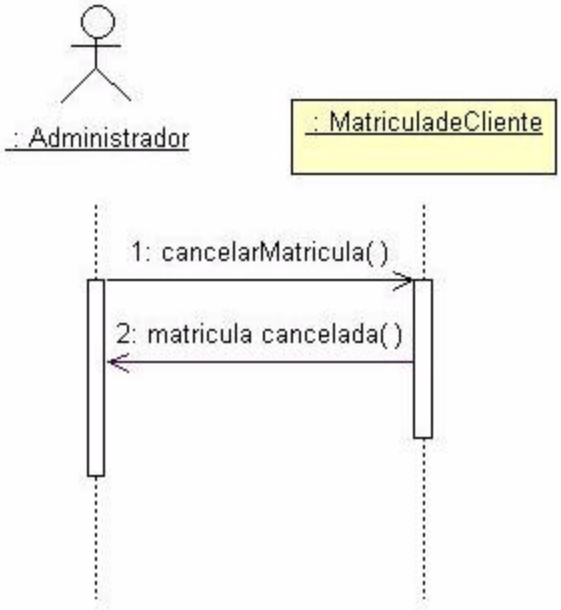


Figura 34: Diagrama de Seqüência Cancelar matricula do cliente

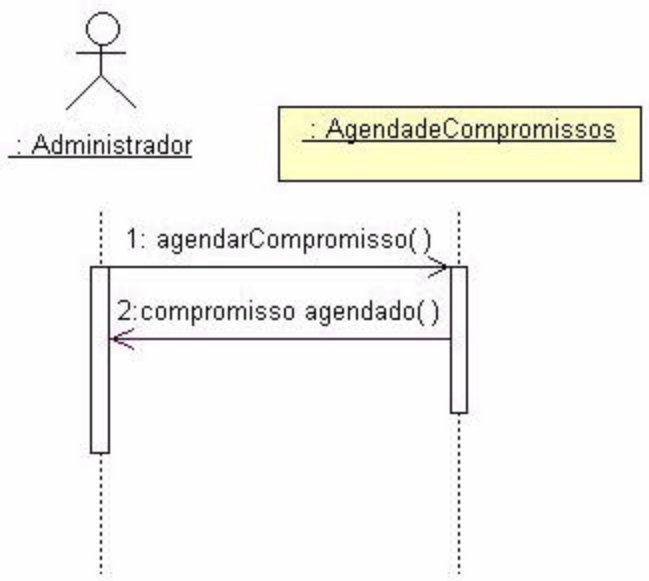


Figura 35: Diagrama de Seqüência Agendar compromisso

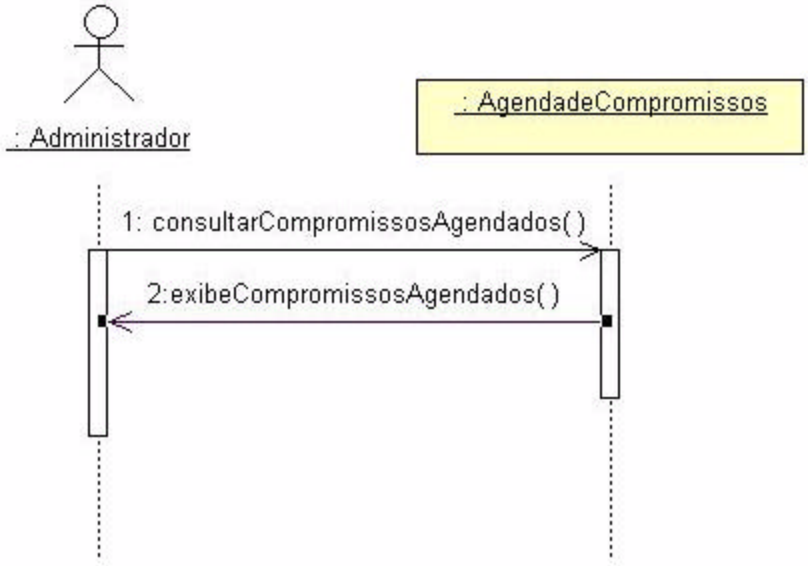


Figura 36: Diagrama de Seqüência Consultar compromissos agendados

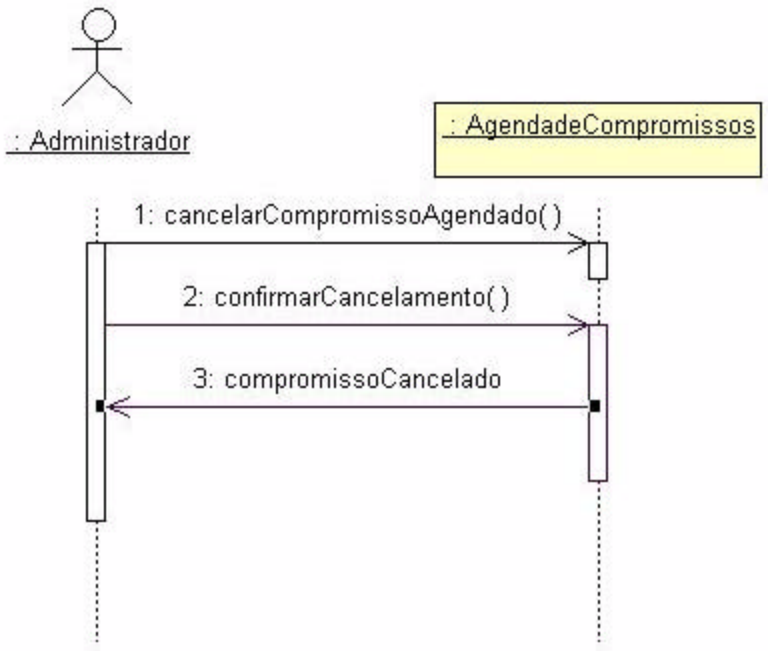


Figura 37: Diagrama de Seqüência Cancelar compromisso agendado

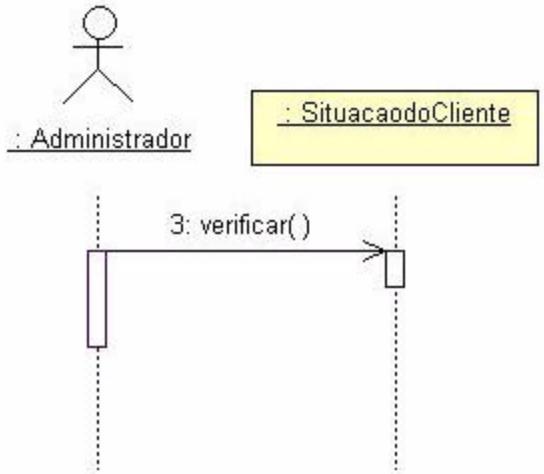


Figura 38: Diagrama de Seqüência Verificar situação do cliente

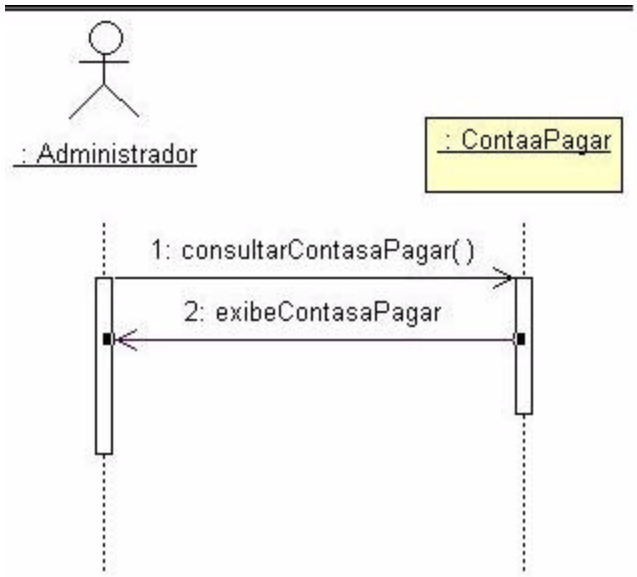


Figura 39: Diagrama de Seqüência Consultar contas a pagar

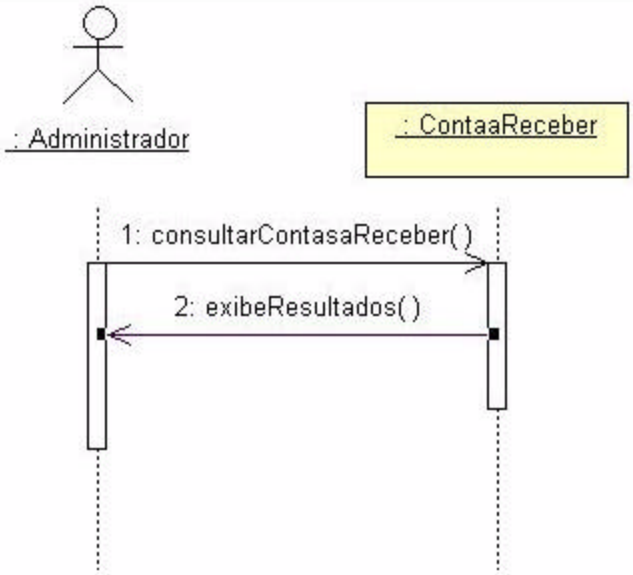


Figura 40: Diagrama de Seqüência Consultar contas a receber

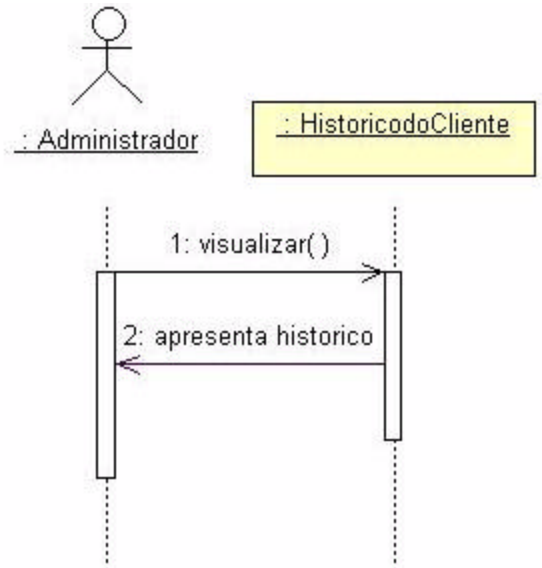


Figura 41: Diagrama de Seqüência Mostrar histórico do cliente

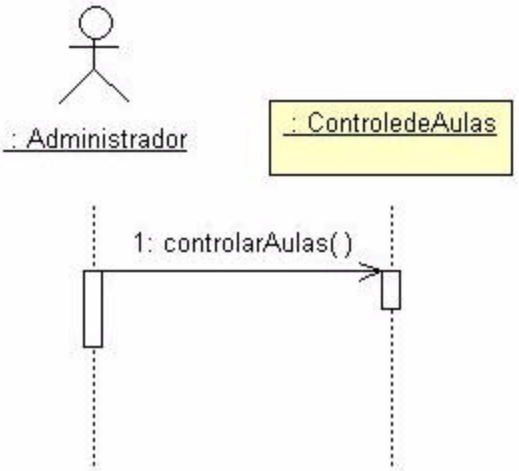


Figura 42: Diagrama de Seqüência Controlar aulas dadas

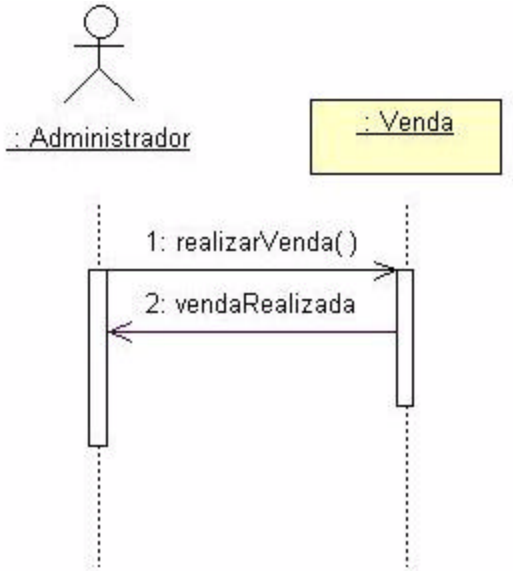


Figura 43: Diagrama de Seqüência Vender produto

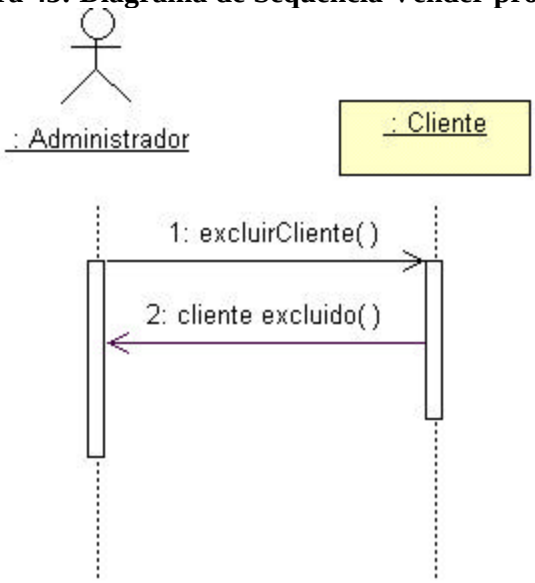


Figura 44: Diagrama de Seqüência Excluir cliente

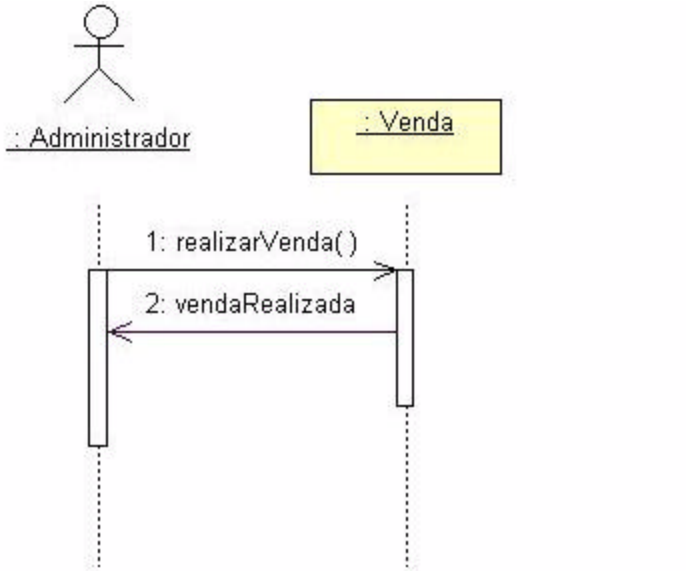


Figura 45: Diagrama de Seqüência Excluir categoria

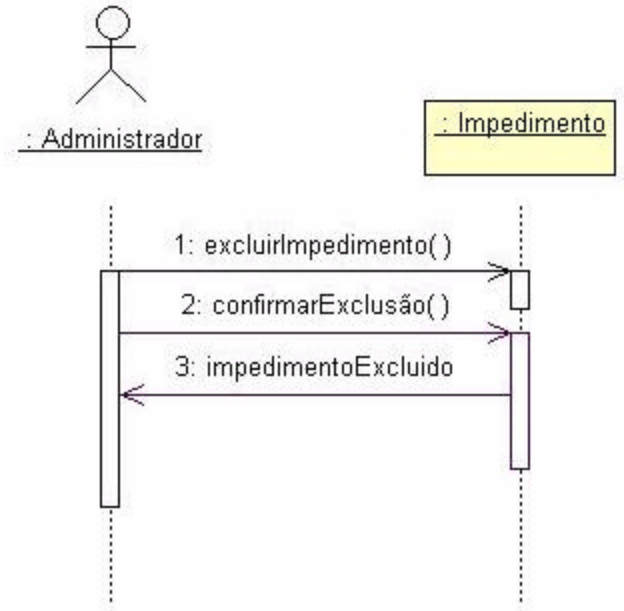


Figura 46: Diagrama de Seqüência Excluir impedimento

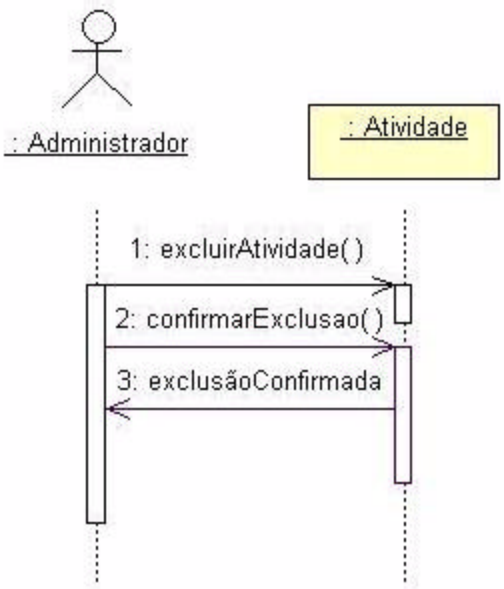


Figura 47: Diagrama de Seqüência Excluir atividade

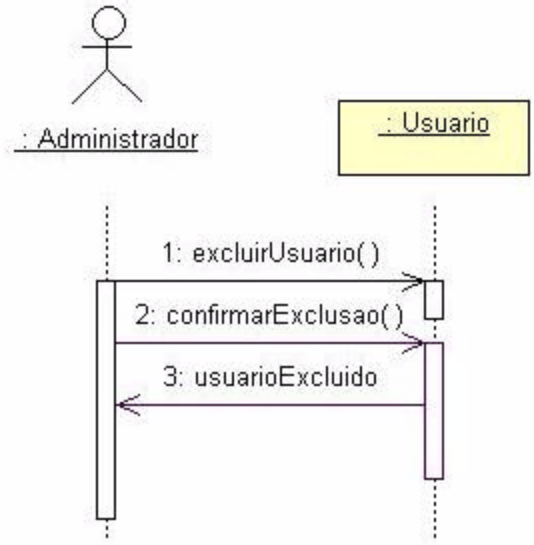


Figura 48: Diagrama de Seqüência Excluir usuário

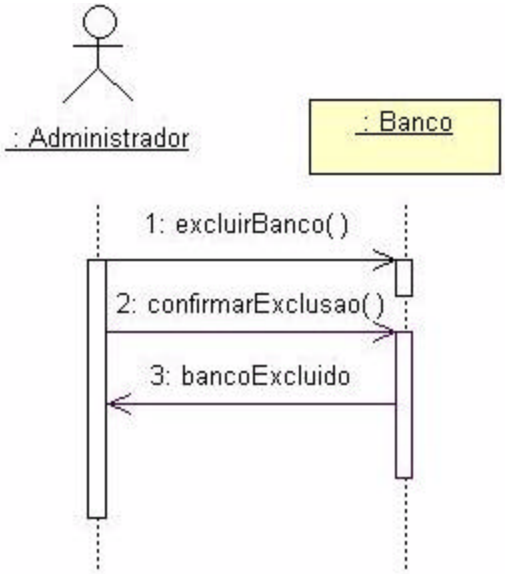


Figura 49: Diagrama de Seqüência Excluir banco

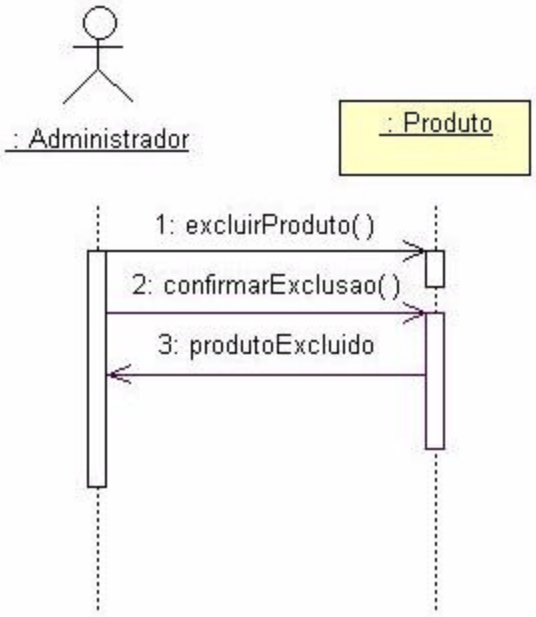


Figura 50: Diagrama de Seqüência Excluir produto

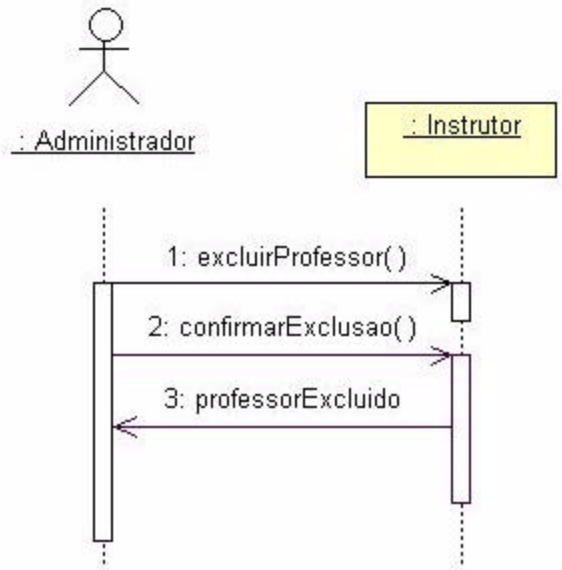


Figura 51: Diagrama de Seqüência Excluir professor

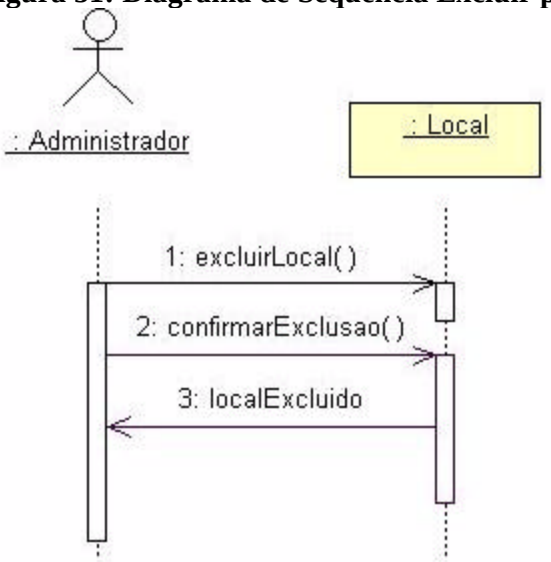


Figura 52: Diagrama de Seqüência Excluir local

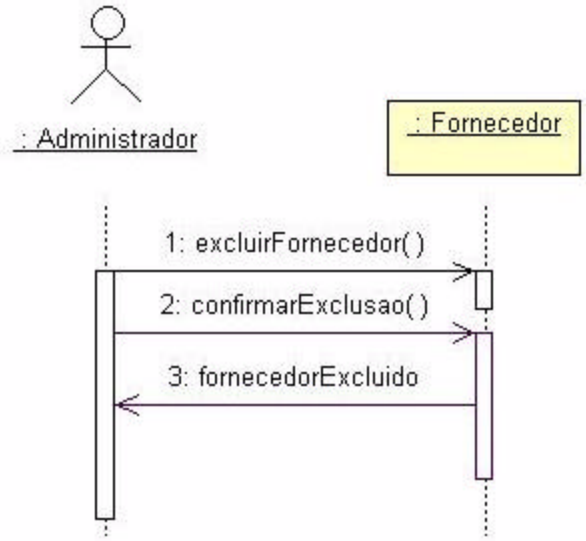


Figura 53: Diagrama de Seqüência Excluir fornecedor

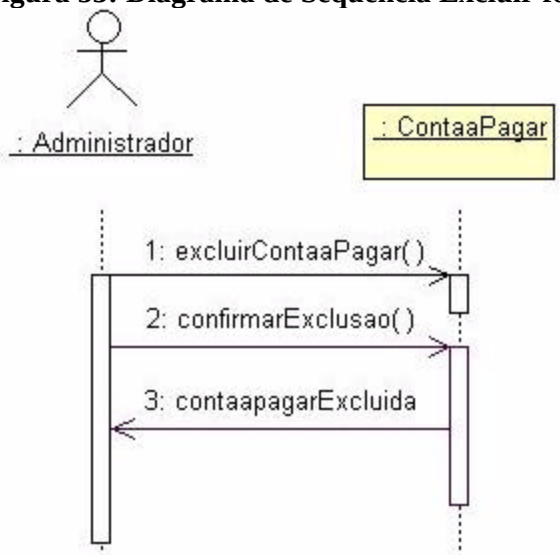


Figura 54: Diagrama de Seqüência Excluir conta a pagar

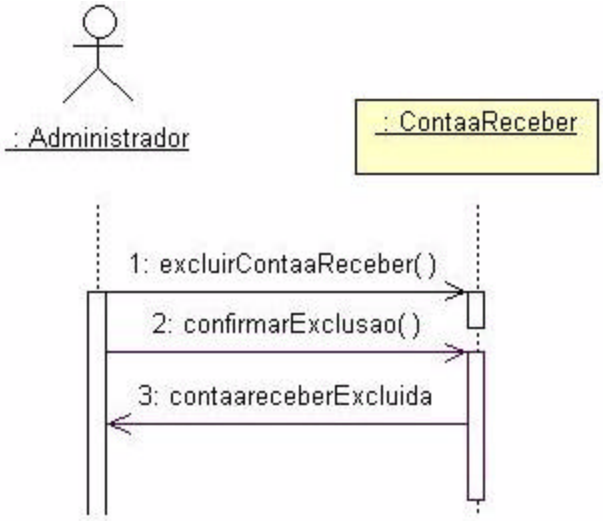


Figura 55: Diagrama de Seqüência Excluir conta a receber

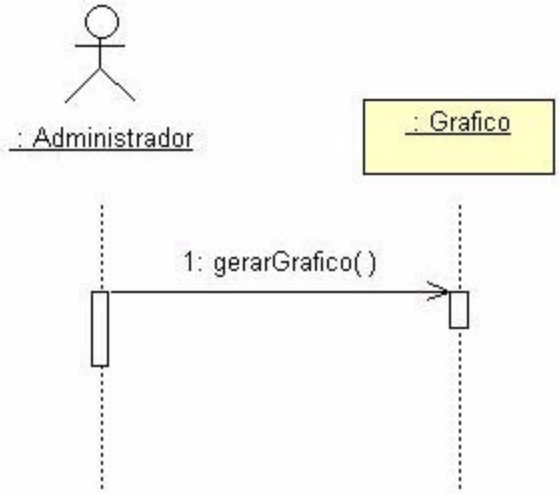


Figura 56: Diagrama de Seqüência Visualizar grafico

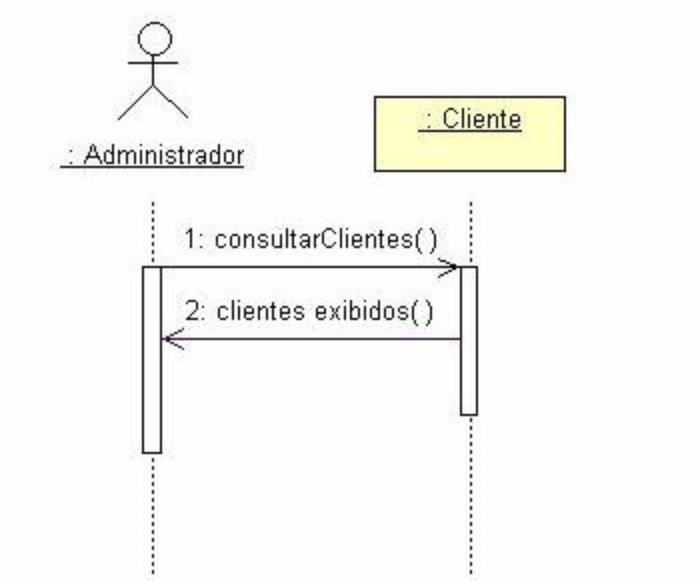


Figura 57: Diagrama de Seqüência Consultar clientes

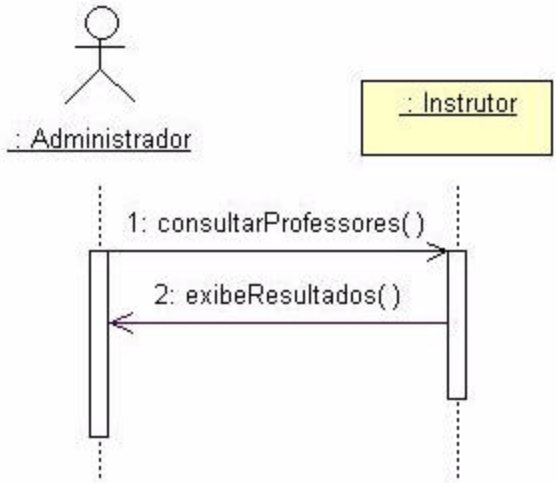


Figura 58: Diagrama de Seqüência Consultar professores

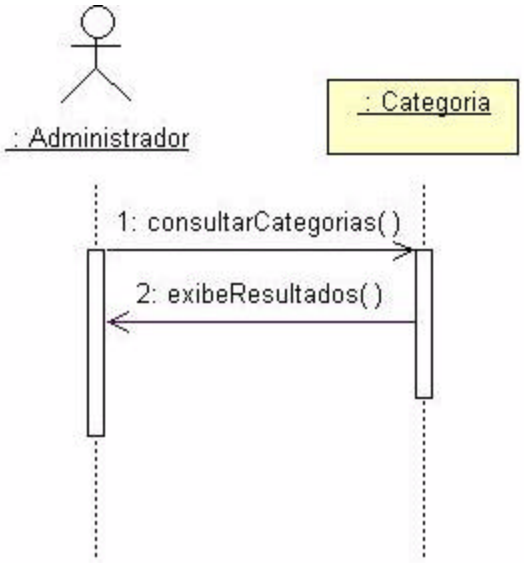


Figura 59: Diagrama de Seqüência Consultar categorias

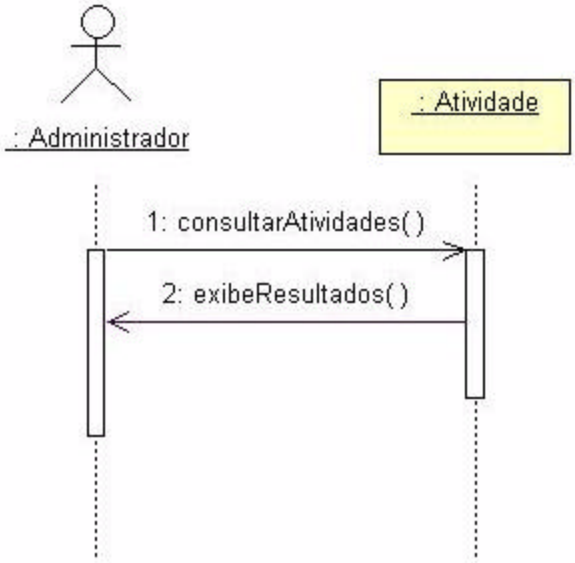


Figura 60: Diagrama de Seqüência Consultar atividades

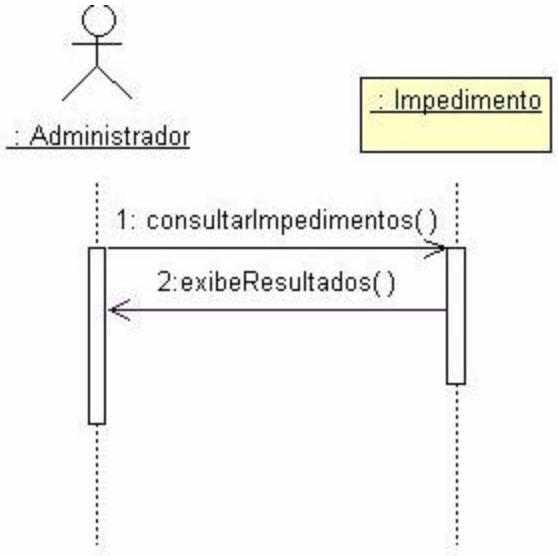


Figura 61: Diagrama de Seqüência Consultar impedimentos

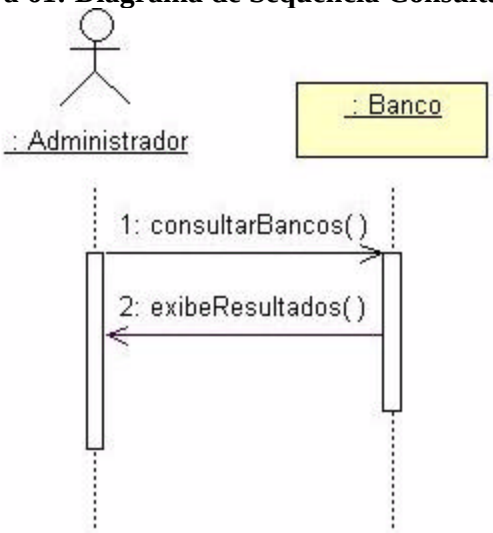


Figura 62: Diagrama de Seqüência Consultar bancos

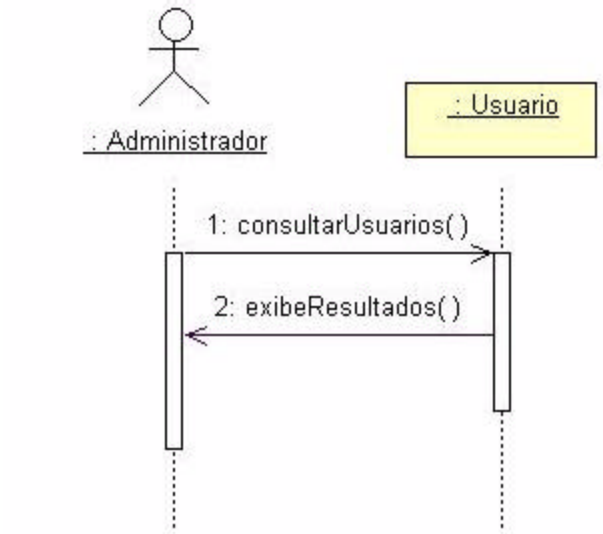


Figura 63: Diagrama de Seqüência Consultar usuários

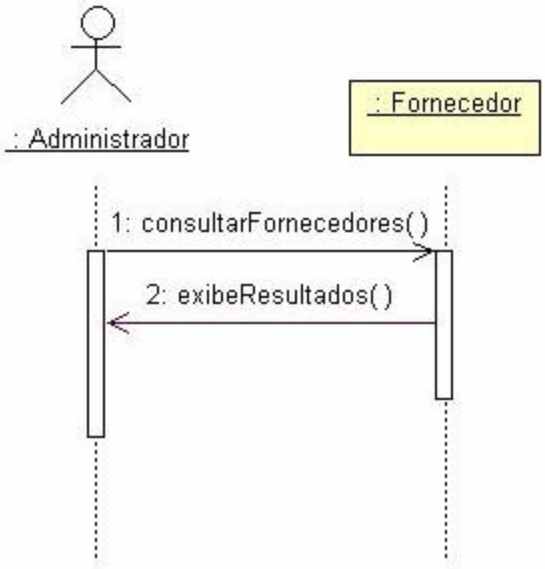


Figura 64: Diagrama de Seqüência Consultar fornecedores

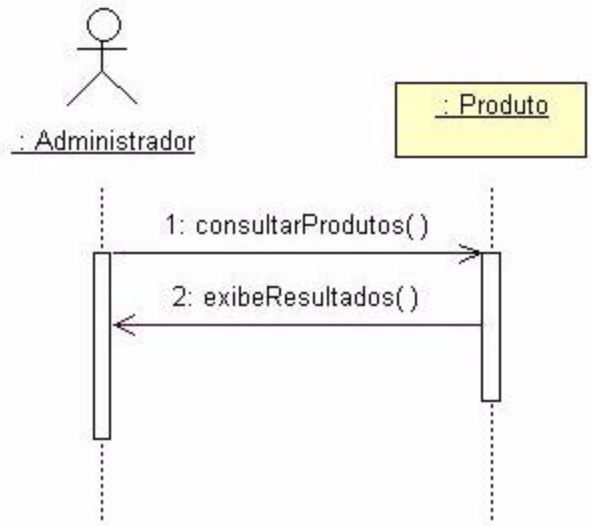


Figura 65: Diagrama de Seqüência Consultar produtos

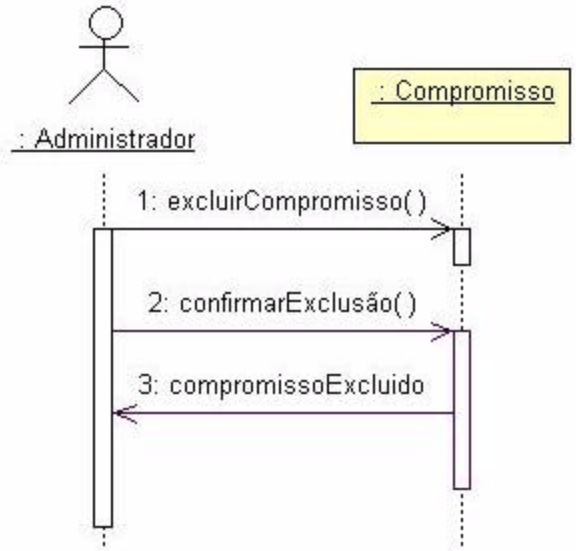


Figura 66: Diagrama de Seqüência Excluir compromisso

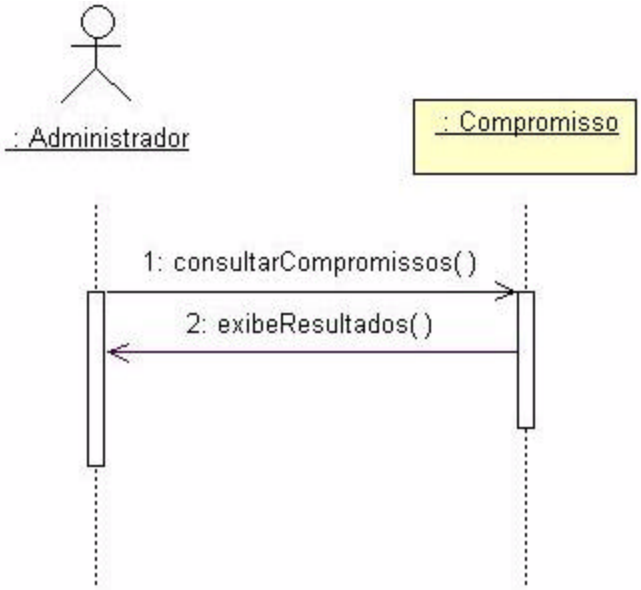


Figura 67: Diagrama de Seqüência Consultar compromissos

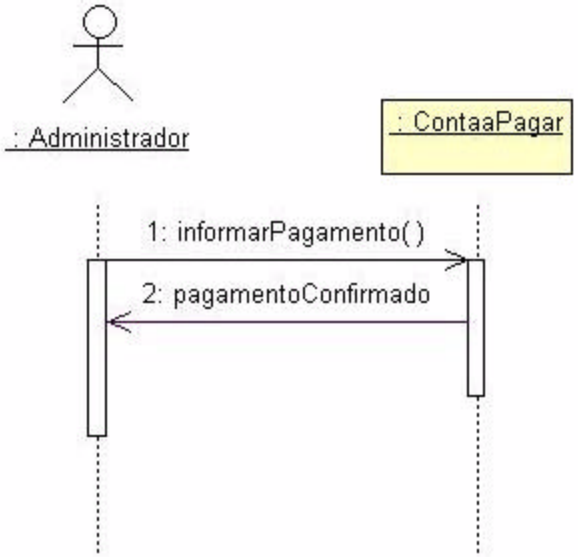


Figura 68: Diagrama de Seqüência Informar pagamento

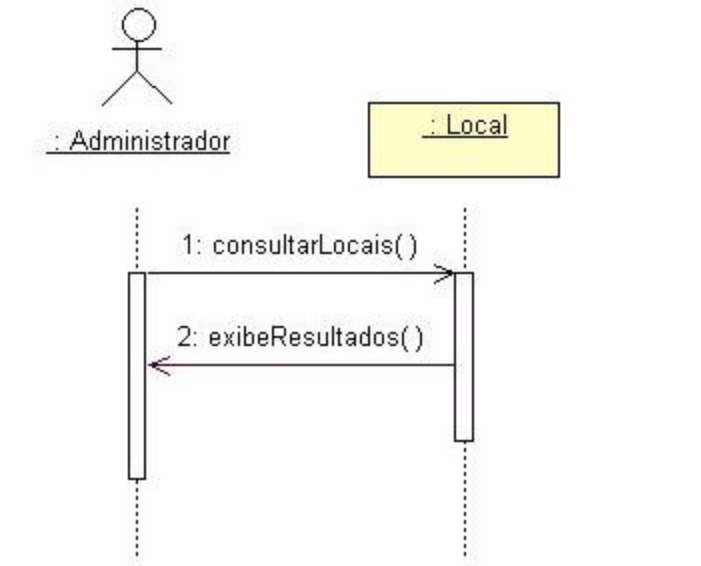


Figura 69: Diagrama de Seqüência Consultar locais

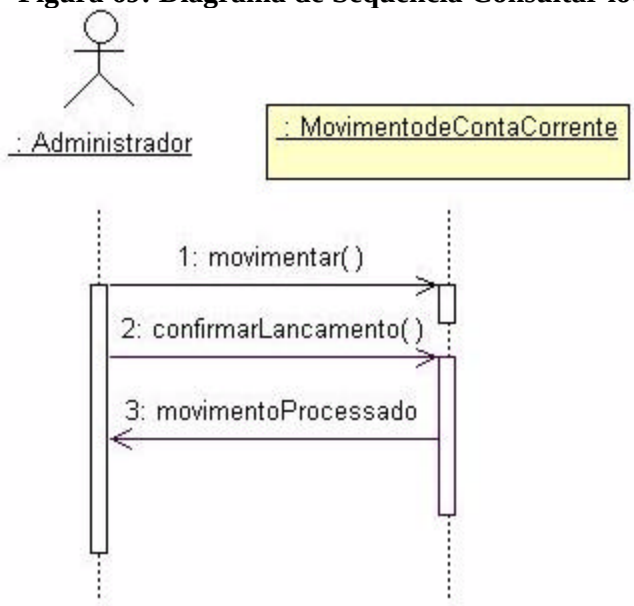


Figura 70: Diagrama de Seqüência Movimentar conta corrente

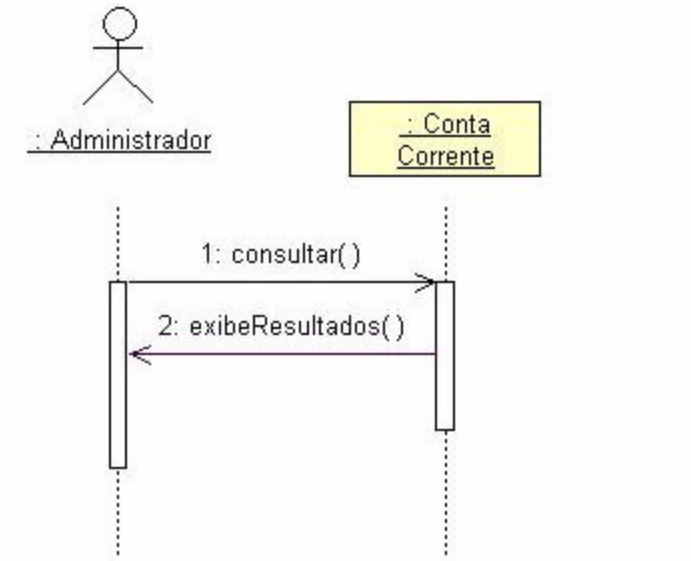


Figura 71: Diagrama de Seqüência Consultar contas correntes

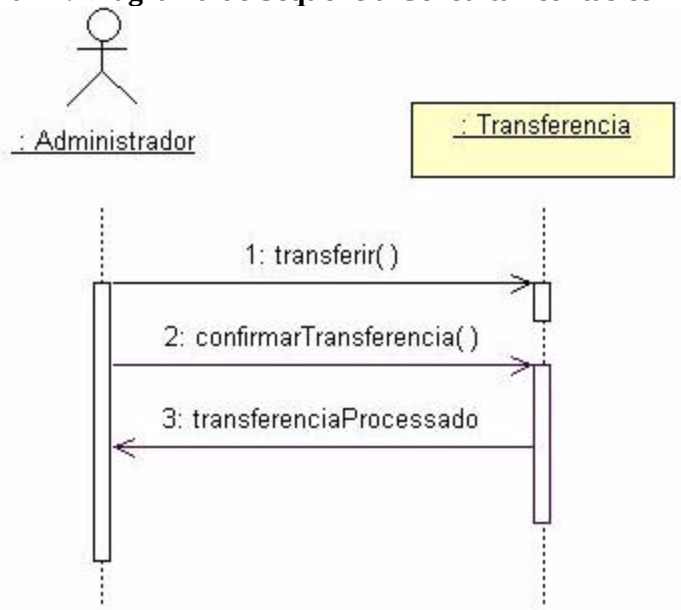


Figura 72: Diagrama de Seqüência Transferir fundos

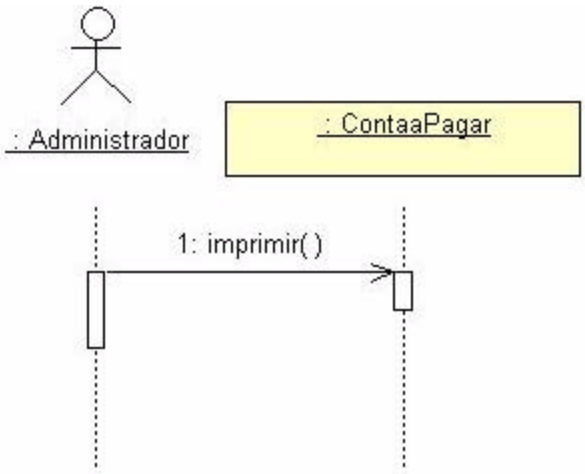


Figura 73: Diagrama de Seqüência Emitir relatorio da folha de pagamento

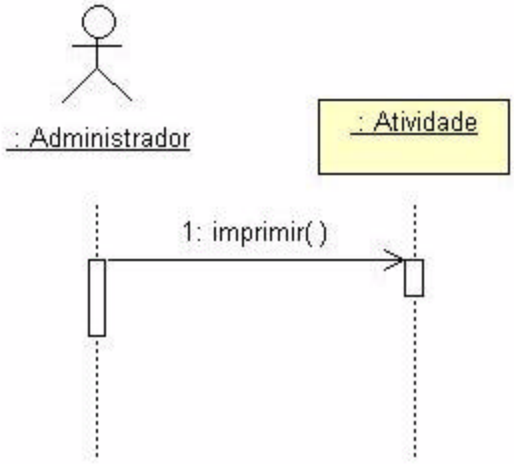


Figura 74: Diagrama de Seqüência Emitir relatorio das atividades

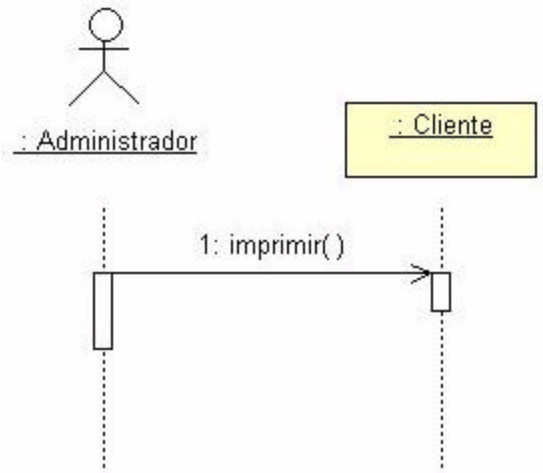


Figura 75: Diagrama de Seqüência Emitir relatorio dos clientes

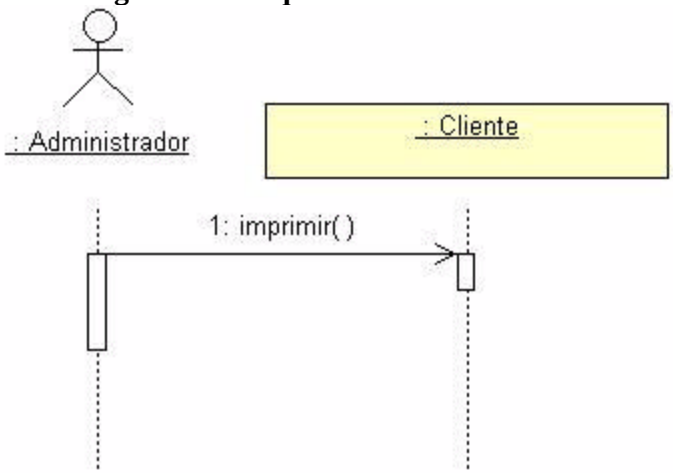


Figura 76: Diagrama de Seqüência Emitir relatorio dos aniversariantes

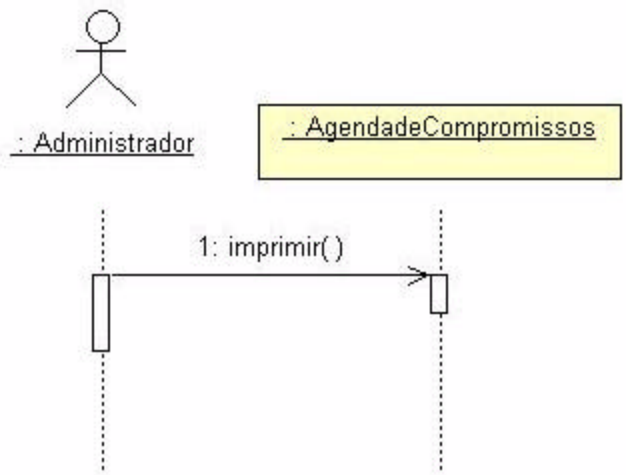


Figura 77: Diagrama de Seqüência Emitir relatorio dos compromissos agendados

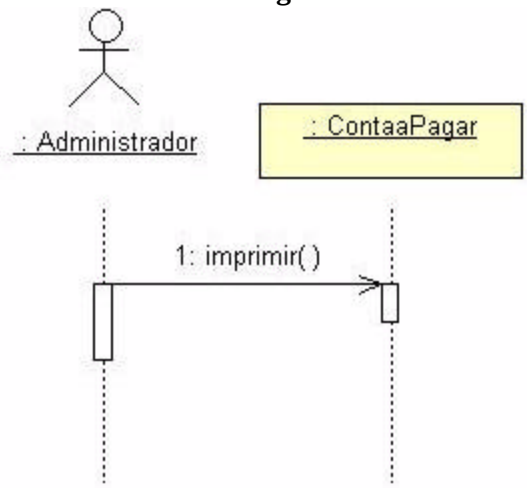


Figura 78: Diagrama de Seqüência Emitir relatorio das contas a pagar

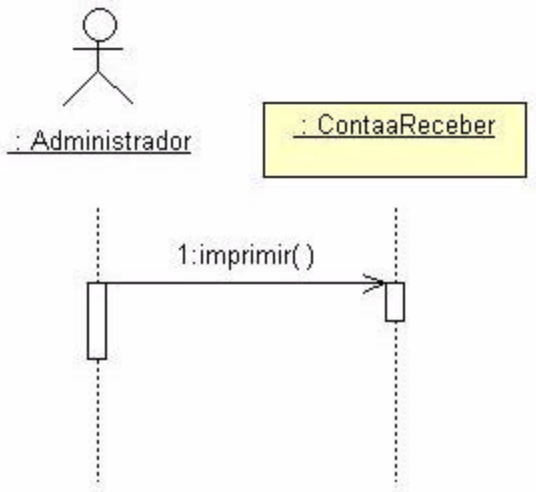


Figura 79: Diagrama de Seqüência Emitir relatorio das contas a receber

5.2.3 Diagrama de Classe

A Figura 80 apresenta o Diagrama de Classe. O código gerado, os atributos e operações de cada classe estão em anexo.

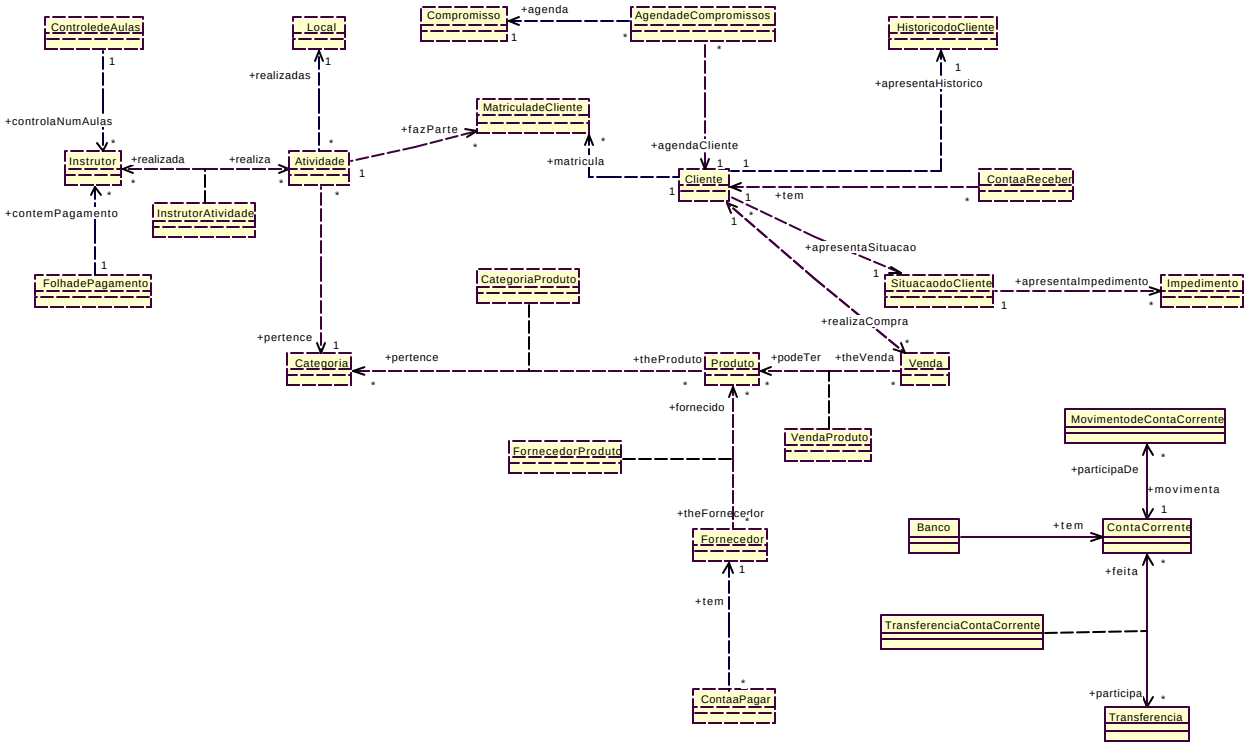


Figura 80: Diagrama de Classe

A ferramenta *Rational Rose 2000* permite ao usuário gerar o código do programa em várias linguagens como Java, Oracle 8, C++ e Visual Basic, mais uma vantagem, além da modelagem do funcionamento, proporcionada pela ferramenta. Neste projeto gerou-se o código em Java, sendo alguns exemplos apresentados no Anexo A. O código das demais classes também estará disponível na interface.

7 CONCLUSÃO

A documentação é fundamental para o sucesso do processo de desenvolvimento de um sistema, ajudando tanto o analista quanto o desenvolvedor. Porém muitos acham que esta é uma tarefa desnecessária, motivo de vários sistemas não serem usados adequadamente ou serem substituídos por outros, por causa da dificuldade em manter o *software* existente.

Para realização deste trabalho foi feito um estudo sobre a UML e utilização da ferramenta *Rational Rose 2000,* o que proporcionou o desenvolvimento da documentação.

Os diversos diagramas criados servem para uma melhor interpretação deste desenvolvimento, propiciando uma visão geral do funcionamento. Portanto, nesta primeira etapa de análise no desenvolvimento da modelagem, torna-se a base para uma posterior implementação do *software*.

Pelo fato de ser um sistema completo e de razoável complexidade, uma contribuição deste trabalho, além de mostrar as vantagens da documentação, é a disponibilização de um documento digital descrevendo as suas etapas da modelagem passo a passo.

8 TRABALHOS FUTUROS

Uma possibilidade de continuação deste trabalho é a modelagem do restante dos requisitos necessários, que ainda não foram modelados, a fim de desenvolver um *software* bem-sucedido.

A documentação da modelagem mostrando o funcionamento do sistema da **Academia Kemper Forma** estará disponível para que, posteriormente, seja desenvolvido o sistema, no qual o desenvolvedor se beneficiará de todas as vantagens que a documentação *on-line* proporciona.

Foi observado o interesse por parte do administrador em desenvolver uma página para a academia. Para isto a interface já criada poderá ser complementada, por exemplo, para auxiliar os alunos da academia, mostrando as atividades que serão realizadas no dia, consulta de avaliações físicas, ficha de musculação, entre outros.

Este trabalho servirá de auxilio, como fonte de pesquisa, para o desenvolvimento de outros trabalhos a serem realizados por alunos de Engenharia de Software.

9 REFERÊNCIA BIBLLIOGRAFICA

[1] ANDRADE, EDNÉIA LEONOR PEREIRA DE. **Método de desenvolvimento de sistemas orientados a objetos utilizando UML.** Brasília: Embrapa – DIN, 1999. 90p.

[2] Casos de Uso. Disponível na internet. http://www.netjava.hpg.ig.com.br.htm.

[3] CE-2 Definição do Modelo para o Protótipo de Software de Identificação de usuários. Disponível na internet. http://www.comp.ita.Br.

[4] http://www.pr.gov.br/celepar/celepar/batebyte/edicoes/1999/bb88

[5] LARMAN, CRAIG. **Utilizando UML E PADRÕES** – Uma introdução à análise e ao projeto orientado a objetos. ISBN: 8573076518. Publicado pela Bookman em 1999.

[6] PRESMANN, ROGER S. **Engenharia de Software**. Tradução por José Carlos Barbosa. São Paulo; Makron Books, 1995.

[7] RESENDE, ANTÔNIO MARIA PEREIRA. Notas de aula, 2000.

[8] YOURDON, EDWARD. **Análise Estruturada Moderna**. ISBN: 8570016158. Publicado pela Editora Campus 1990.

ANEXO A

CLASSE: AGENDADECOMPROMISSOS

```
/**
 * Na Agenda de Compromissos são cadastrados todos os compromissos
 * agendados em ordem cronológica.
 */
public class AgendadeCompromissos
{
   /**
    * Descreve o local onde será o compromisso.
    */
   private Local descricaoLocal;

   /**
    * Descrição do compromisso marcado.
    */
   private Compromisso descricaoCompromisso;

   /**
    * Horário marcado para o compromisso.
    */
   private time horarioCompromisso;

   /**
    * Data do compromisso.
    */
```

```
private date dataCompromisso;

/**
 * Nome do cliente.
 */
private Cliente nomeCliente;

/**
 * Categoria na qual o compromisso agendado faz parte.
 */
private Categoria descricaoCategoria;
private int codigoAgendamento;
public Cliente agendaCliente;
public Compromisso agenda;

/**
 */
public AgendadeCompromissos()
{
}

/**
 * O administrador agenda um compromisso na AgendadeCompromissos,
 * podendo controlar a data e horário de compromissos dos clientes da
 * academia. Exemplos de compromissos podem ser avaliação física, avaliação
 * nutricional, torneio  interno, palestra sobre atividade física, etc.
 * @return Compromisso
 */
```

```
public Compromisso agendarCompromisso()
{
 return null;
}

/**
 * Consulta todos os compromissos agendados.
 * @return AgendadeCompromissos
 */
public AgendadeCompromissos consultarCompromissosAgendados()
{
 return null;
}

/**
 * Cancela um compromisso marcado na Agenda de Compromissos.
 */
public void cancelarCompromissoAgendado()
{

}

/**
 * O administrador confirma o cancelamento do compromisso.
 */
public void confirmarCancelamento()
{
```

```
    }

    /**
     * O administrador solicita a emissão do relatório dos compromissos
     * agendados.
     */
    public void emitirRelatoriodaAgenda()
    {
    }
}
/**
 *
 *
 *
 *
 *
 * AgendadeCompromissos.imprimir(){
 *
 *     }
 *
 *
 *
 *
 *
 *
 */
```

CLASSE: ATIVIDADE

```
/**
 * Classe das atividades cadastradas. O administrador cadastra todas as
 * atividades que a academia oferece a seus clientes. Por exemplo: Body Pump,
 * Body Combat, Power Local, Step, Alongamento, Swing, Swing/Local,
 * Step/Abdome, Combat/Attack, Super Local, Super Step, The Best of Body
 * Combat, Body Systems, Capoeira e  Forró.
 */
public class Atividade
{

   /**
    * Códiga da atividade.
    */
   private int codigoAtividade;

   /**
    * Categoria na qual a atividade pertence.
    */
   private Categoria categoriaAtividade;

   /**
    * Descrição da atividade.
    */
   private char descricaoAtividade;

   /**
```

```
 * Local onde a atividade será realizada.
 */
private local localAtividade;

/**
 * Nome do professor responsável pela atividade.
 */
private Instrutor professorAtividade;

/**
 * Horário de inícil da aula.
 */
private time horarioInicio;

/**
 * Idade recomendada para participar da atividade.
 */
private int faixaEtariaAtividade;

/**
 * Sexo do cliente que poderá participar da atividade (feminino, masculino,
 * outros).
 */
private char sexo;

/**
 * Quantidade de vagas que local comporta de clientes para participar da
 * atividade.
```

```
 */
private int vagasAtividade;

/**
 * Preço da atividade por mês.
 */
private value precoMes;

/**
 * Preço da atividade por aula.
 */
private value precoAula;

/**
 * Dias da semana que será realizada a atividade.
 */
private date diasAula;

/**
 * Horário em que a atividade finaliza.
 */
private time horarioTermino;
public Cliente participam[];

/**
 * Uma atividade pertence a uma categoria.
 */
public Categoria pertence;
```

```
public Instrutor realizada[];
public MatriculadeCliente fazParte[];
public Local realizadas;

/**
 */
public Atividade()
{
}

/**
 * Cadastra a atividade, informando seus dados que serão armazenados na base
 * de dados.
 * @return char
 */
public char cadastrarAtividade()
{
 return 0;
}

/**
 * Consulta todas as atividades cadastradas.
 * @return Atividade
 */
public Atividade consultarAtividades()
{
 return null;
}
```

```
/**
 * Exclui uma atividade cadastrada.
 */
public void excluirAtividade()
{
}

/**
 * Verifica se há vaga para o cliente ser matriculado na atividade.
 */
public void verificarVagas()
{
}

/**
 * O administrador confirma a exclusão da atividade.
 */
public void confirmarExclusao()
{
}

/**
 * Emitir um relatório de todas as atividades cadastradas, podendo saber os
 * horários de cada atividade, qual o professor responsável e numero de vagas
 * por atividade relacionada.
 */
public void emitirRelatoriodeAtividades()
```

```
  {
  }
}
/**
 *
 *
 *
 *
 *
 *
 * Atividade.imprimir(){
 *
 *    }
 *
 *
 *
 *
 *
 *
 *
 */
```

CLASSE: BANCO

```
/**
 * O administrador cadastra todas os bancos em que o a academia possui uma
 * conta corrente.
 */
public class Banco
{

   /**
    * Código do banco.
    */
   private int codigoBanco;

   /**
    * Descrição do Banco.
    */
   private char descricaoBanco;

   /**
    * Número da Agência do Banco
    */
   private char agencia;

   /**
    * Número da conta corrente.
    */
   private long contaCorrente;
```

```
/**
 * Saldo inicial do cliente.
 */
private value saldoInicial;

/**
 * Saldo atual do cliente.
 */
private value saldoAtual;

/**
 * Numero do banco
 */
private char numBanco;
public ContaCorrente tem;

/**
 */
public Banco()
{
}

/**
 * Cadastra o Banco, incluindo todos os dados que serão armazenados na base
 * de dados.
 */
public char cadastrarBanco()
```

```
  {
   return 0;
  }

  /**
   * Consulta todos os bancos cadastrados.
   * @return Banco
   */
  public Banco consultarBancos()
  {
   return null;
  }

  /**
   * Excluir um banco cadastrado.
   */
  public void excluirBanco()
  {
  }

  /**
   * O administrador confirma a exclusão do banco.
   */
  public void confirmarExclusao()
  {
  }
}
```

CLASSE: CATEGORIA

```
/**
 * Classe das categorias cadastradas. O administrador cadastra  as atividades,
 * produtos e serviços que a academia oferece e também as despesas ou centros
 * de custo da empresa. Alguns exemplos de categoria de atividade são natação,
 * hidroginástica, musculação, ginástica, judô, catoeira, taxas diversas, exames e
 * avaliações, lanchonete, entre outros (tudo que será interpretado como crédito
 * para academia). Os centros de custo da academia podem ser administração,
 * folha de pagamento e manutenção do funcionamento das atividades.
 */
public class Categoria
{

   /**
    * Código da categoria.
    */
   private int codigoCategoria;

   /**
    * Descrição da categoria.
    */
   private char descricaoCategoria;
   public Atividade contem[];
   public Produto temProdutos;
   public Produto theProduto;

   /**
```

```
 */
public Categoria()
{
}

/**
 * Cadastra a nova categoria, informando os dados que serão armazenados na
 * base de dados.
 * @return char
 */
public char cadastrarCategoria()
{
 return 0;
}

/**
 * Consulta todas as categorias cadastradas.
 * @return Categoria
 */
public Categoria consultarCategorias()
{
 return null;
}

/**
 * Exclui uma categoria cadastrada.
 */
public void excluirCategoria()
```

```
    {
    }

    /**
     * O administrador confirma a exclusão da categoria.
     */
    public void confirmarExclusao()
    {
    }
}
```

CATEGORIAPRODUTO

```
/**
 * Esta é uma classe intermediária, pois há uma associação de muitos-pra-muitos
 * entre as classes Produto e Categoria.
 */
public class CategoriaProduto
{

   /**
    * Código da categoria do produto.
    */
   private Categoria codigoCategoria;

   /**
    * Código do produto.
    */
   private Produto codigoProduto;

   /**
    */
   public CategoriaProduto()
   {
   }
}
```

CLASSE: CLIENTE

```
/**
 * Classe dos clientes cadastrados, ou seja, todas as pessoas que praticam alguma
 * atividade ou consomem algum tipo de produto ou serviço da academia.
 */
public class Cliente
{

   /**
    * Código do cliente.
    */
   private int codigoCliente;

   /**
    * Nome do cliente
    */
   private char nomeCliente;

   /**
    * Endereço do cliente.
    */
   private char enderecoCliente;

   /**
    * Cidade onde situa o cliente.
    */
   private char cidadeCliente;
```

```
/**
 * Estado da cidade do cliente.
 */
private char estadoCliente;

/**
 * CEP da cidade do cliente.
 */
private long CEPCliente;

/**
 * Número do telefone de contato do cliente.
 */
private num foneCliente;

/**
 * Número do faz do cliente.
 */
private num faxCliente;

/**
 * Sexo do cliente.
 */
private long sexo;

/**
 * Data de nascimento do cliente
```

```
 */
private date dataNascimentoCliente;

/**
 * Número de identidade do cliente.
 */
private long IDCliente;

/**
 * Número do CPF do cliente.
 */
private long CPFCliente;

/**
 * E-mail do cliente.
 */
private char e_mailCliente;

/**
 * Nome do responsável do cliente.
 */
private char responsavel;
public SituacaodoCliente apresentaSituacao;
public HistoricodoCliente tem;
public ContaaReceber podemTer[];
public Venda realizaCompra[];
public AgendadeCompromissos temCompromisso[];
public HistoricodoCliente apresentaHistorico;
```

```
public MatriculadeCliente matricula[];

/**
 */
public Cliente()
{
}

/**
 * O administrador cadastra o novo cliente, informando todos os dados do
 * cliente, onde serão armazenados no banco de dado.
 * @return char
 */
public char cadastrarCliente()
{
 return 0;
}

/**
 * Consulta todos os clientes cadastrados.
 */
public void consultarClientes()
{
}

/**
 * Exclui um cliente cadastrado.
 */
```

```
    public void excluirCliente()
    {
    }

    /**
     * O administrador confirma a exclusão do cliente.
     */
    public void confirmarExclusao()
    {
    }

    /**
     * Solicita a impressão dos aniversariantes.
     */
    public void emitirRelatorioAniversariantes()
    {
    }

    /**
     * O administrador pode solicitar uma impressão de cliente em ordem
     * alfabética, com exames vencidos, com matrículas vencidas, com acesso
     * impedido, matriculado por atividade, matriculado por categoria e dos
     * clientes inativos (que não estão matriculados em nenhuma atividade).
     */
    public void emitirRelatorioClientes()
    {
    }
}
```

```
/**
 *
 *
 *
 *
 * Cliente.imprimir(){
 *
 *     }
 *
 *
 *
 *
 */
```

CLASSE: COMPROMISSO

```
/**
 * O administrador cadastra todos os compromissos que podem ser marcados
 * referente a academia. Alguns exemplos de compromisso podem ser exame
 * médico, avaliação física, avaliação nutricional, torneio interno, palestra sobre
 * atividade física.
 */
public class Compromisso
{

  /**
   * Código do compromisso.
   */
  private int codigoCompromisso;

  /**
   * Descrição do compromisso.
   */
  private char descricaoCompromisso;

  /**
   * Categoria na qual o compromisso pertence.
   */
  private Categoria categoriaCompromisso;
  public AgendadeCompromissos theAgendadeCompromissos;

  /**
```

```
 */
public Compromisso()
{
}

/**
  * Cadastra o novo compromisso, informando os dados do compromisso que serão
 * armazenados na base de dados.
 * @return char
 */
public char cadastrarCompromisso()
{
 return 0;
}

/**
 * Exclui um compromisso cadastrado.
 */
public void excluirCompromisso()
{
}

/**
 * Consulta todos os compromissos cadastrados.
 * @return Compromisso
 */
public Compromisso consultarCompromissos()
```

```
    {
     return null;
    }

    /**
     * O administrador confirma a exclusão do compromisso.
     */
    public void confirmarExclusão()
    {
    }

    /**
     * O administrador solicita a impressão do relatório de todos os compromissos
     * agendados agupados por data e por ordem de horário.
     */
    public void emitirRelatorioCompromissos()
    {
    }
}
```

CLASSE: CONTAAPAGAR

```
/**
 * Classe de todas as contas a pagar cadastradas. O administrador cadastra todas
 * as despesas que a academia consome. Estas despesas podem ser agrupadas por
 * fornecedores ou data de vencimento.
 */
public class ContaaPagar
{

  /**
   * Código da conta a pagar.
   */
  private int codigoContaPagar;

  /**
   * Nome do fornecedor que será pago.
   */
  private Fornecedor nomeFornecedor;

  /**
   * Categoria que a conta a pagar pertence.
   */
  private Categoria categoriaFornecedor;

  /**
   * Valor da conta a pagar.
   */
```

```
private value valorContaaPagar;

/**
 * Data de vencimento da conta a pagar.
 */
private date dataVencimento;

/**
 * Se houver atraso, indica o o número de dias de atraso.
 */
private int diasAtraso;

/**
 * Se houver atraso, calcula o juros.
 */
private value valorJuros;
public Fornecedor tem;

/**
 */
public ContaaPagar()
{
}

/**
 * Cadastra a conta a pagar, informando os dados necessários que serão
 * armazenados na base de dados.
 * @return char
```

```
 */
public char cadastrarContaaPagar()
{
 return 0;
}

/**
 * Informa que uma conta cadastrada foi paga.
 */
public void informarPagamento()
{
}

/**
 * Consulta todas as contas a pagar cadastradas.
 * @return ContaaPagar
 */
public ContaaPagar consultarContasaPagar()
{
 return null;
}

/**
 * Exclui uma conta a pagar cadastrada.
 */
public void excluirContaaPagar()
{
}
```

```
    /**
     * O administrador confirma a exclusão da conta a pagar.
     */
    public void confirmarExclusao()
    {
    }

    /**
     * Imprime a Folha de Pagamento.
     */
    public void emitirRelatorio()
    {
    }
}
/**
 *
 *
 *
 *
 *
 * ContaaPagar.imprimir(){
 *
 *     }
 *
 *
 *
 *
```

```
*
*/
```

CLASSE: CONTAARECEBER

```
/**
 * Classe que contém todas as contas a receber cadastradas. O administrador
 * cadastra as contas a receber dos clientes, permitindo que seja realizado um
 * controle das despesas.
 */
public class ContaaReceber
{

   /**
    * Código da conta a receber.
    */
   private int codigoContaReceber;

   /**
    * Nome do cliente que pagará a conta.
    */
   private Cliente nomeCliente;

   /**
    * Valor da conta a receber.
    */
   private value valorContaaReceber;

   /**
    * Data de recebimento da conta.
    */
```

```
private date dataRecebimento;

/**
 * Número de dias de atraso.
 */
private int diasAtraso;

/**
 * Se houver atraso calcula o valor do juros.
 */
private value valorJuros;
public Cliente tem;

/**
 */
public ContaaReceber()
{
}

/**
 * Cadastra a conta a receber, informando os dados necessários que serão
 * armazenados na base de dados.
 * @return char
 */
public char cadastrarContaaReceber()
{
 return 0;
}
```

```
/**
 * Informa o recebimento de uma conta
 */
public void informarRecebimento()
{
}

/**
 * Exclui uma conta a receber cadastrada.
 */
public void excluirContaaReceber()
{
}

/**
 * Consulta todas as contas a receber.
 * @return ContaaReceber
 */
public ContaaReceber consultarContasaReceber()
{
 return null;
}

/**
 * O administrador confirma a exclusão da conta a receber.
 */
public void confirmarExclusao()
```

```
    {
    }

    /**
     * Solicita a impressão do relatório de contas a receber.
     */
    public void emitirRelatorio()
    {
    }
}
/**
 *
 *
 *
 *
 * ContaaReceber.imprimir(){
 *
 *     }
 *
 *
 *
 *
 *
 */
```

CLASSE: CONTACORRENTE

```
/**
 * Funciona como um controle bancário. É possível realizar lançamento,
 * transferências e fazer consultas.
 */
public class ContaCorrente
{

   /**
    * Descrição do banco.
    */
   private Banco descricaoBanco;

   /**
    * Numero do banco.
    */
   private char numBanco;

   /**
    * Número da agência.
    */
   private char agencia;

   /**
    * Número da conta corrente.
    */
   private char contaCorrente;
```

```
    private int codigoContaCorrente;
    public Banco pertence;
    public MovimentodeContaCorrente participaDe[];
    public Transferencia participa;

    /**
     */
    public ContaCorrente()
    {
    }

    /**
     * Consulta os últimos lançamentos das contas correntes cadastradas.
     */
    public void consultar()
    {
    }
}
/**
 *
 *
 *
 * ContaCorrente.processar(){
 *
 *
 *
 *
 *
```

CLASSE: CONTROLEDEAULAS

```
/**
 * Classe de controle do número de aulas de cada professor ou instrutor.
 */
public class ControledeAulas
{

  /**
   * Nome do professor.
   */
  private Instrutor nomeProfessor;

  /**
   * Número de aulas que o professor realizou.
   */
  private int numeroAulas;

  /**
   * Mes que será analisado.
   */
  private char mes;

  /**
   * Ano que será analisado.
   */
  private char ano;
  public Instrutor controlaNumAulas[];
```

```
  /**
   */
  public ControledeAulas()
  {

  }

  /**
   * Controlar as aulas que cada professor realizou.
   * @return ControledeAulas
   */
  public ControledeAulas controlarAulas()
  {
   return null;
  }
}
```

CLASSE: FOLHADEPAGAMENTO

```
/**
 * Contém todas as dívidas que a academia contraiu.
 */
public class FolhadePagamento
{

  /**
   * Código do pagamento da Folha de Pagamento.
   */
  private RelatorioFolhadePagamento codigoPagamento;

  /**
   * Código do professor que será pago.
   */
  private Instrutor codigoProfessor;
  public Instrutor contemPagamento[];

  /**
   */
  public FolhadePagamento()
  {
  }

  /**
   */
  public void emitirRelatorio()
```

```
    {
    }
}
```

CLASSE: FORNECEDOR

```
/**
 * Classe dos fornecedores cadastrados. O administrador cadastra todos os
 * fornecedores e prestadores de serviço da academia referente a produtos e
 * matérias fornecidas por eles.
 */
public class Fornecedor
{

   /**
    * Código do fornecedor.
    */
   private int codigoFornecedor;

   /**
    * Nome do fornecedor.
    */
   private char nomeFornecedor;

   /**
    * Endereço do fornecedor.
    */
   private char enderecoFornecedor;

   /**
    * Cidade onde situa o fornecedor.
    */
```

```
private char cidadeFornecedor;

/**
 * Estado da cidade do fornecedor.
 */
private char estadoFornecedor;

/**
 * Número do telefone de contato do fornecedor.
 */
private long foneForncedor;

/**
 * Número do fax do fornecedor.
 */
private long faxFornecedor;
public Produto fornecido[];
public ContaaPagar podemTer;
public Produto theProduto;

/**
 */
public Fornecedor()
{
}

/**
 * Consultar os fornecedores cadastrados.
```

```
 */
public Fornecedor consultarFornecedores()
{
 return null;
}

/**
 * Excluir um fornecedor cadastrado.
 */
public void excluirFornecedor()
{
}

/**
 * Cadastra o fornecedor, informando os dados necessários que serão
 * armazenados na base de dados.
 * @return char
 */
public char cadastrarFornecedor()
{
 return 0;
}

/**
 * O administrador confirma a exclusão do fornecedor.
 */
public void confirmarExclusao()
{
```

```
    }
}
```

CLASSE: FORNECEDORPRODUTO

```
/**
 * Classe dos fornecedores cadastrados. O administrador cadastra todos os
 * fornecedores e prestadores de serviço da academia referente a produtos e
 * matérias fornecidas por eles.
 */
public class Fornecedor
{

   /**
    * Código do fornecedor.
    */
   private int codigoFornecedor;

   /**
    * Nome do fornecedor.
    */
   private char nomeFornecedor;

   /**
    * Endereço do fornecedor.
    */
   private char enderecoFornecedor;

   /**
    * Cidade onde situa o fornecedor.
    */
```

```
private char cidadeFornecedor;

/**
 * Estado da cidade do fornecedor.
 */
private char estadoFornecedor;

/**
 * Número do telefone de contato do fornecedor.
 */
private long foneForncedor;

/**
 * Número do fax do fornecedor.
 */
private long faxFornecedor;
public Produto fornecido[];
public ContaaPagar podemTer;
public Produto theProduto;

/**
 */
public Fornecedor()
{
}

/**
 * Consultar os fornecedores cadastrados.
```

```
     * @return Fornecedor
     */
    public Fornecedor consultarFornecedores()
    {
     return null;
    }

    /**
     * Excluir um fornecedor cadastrado.
     */
    public void excluirFornecedor()
    {
    }

    /**
     * Cadastra o fornecedor, informando os dados necessários que serão armazenados na
     * base de dados.
     * @return char
     */
    public char cadastrarFornecedor()
    {
     return 0;
    }

    /**
     * O administrador confirma a exclusão do fornecedor.
     */
```

```
    public void confirmarExclusao()
    {
    }
}
```

CLASSE: FORNECEDORPRODUTO

```
/**
 * Esta é uma classe intermediária, pois há uma associação de muitos-pra-muitos
 * entre as classes Fornecedor e Produto.
 */
public class FornecedorProduto
{

   /**
    * Código do produto que o fornecedor fornece.
    */
   private Produto codigoProduto;

   /**
    * Código do forneceor que fornece o produto.
    */
   private Fornecedor codigoFornecedor;

   /**
    */
   public FornecedorProduto()
   {
   }
}
```

CLASSE: GRAFICO

```
/**
 * O administrador poderá visualizar um gráfico das despesas por categoria
 * (analisando a despesa de cada categoria), das matrículas por categoria
 * (analisando qual categoria mais procurada) ou dos códigos de impedimento
 * (verificando quais os maiores impedimentos).
 */
public class Grafico
{

   /**
    */
   public Grafico()
   {
   }

   /**
    * O administrador poderá visualizar um gráfico das despesas por categoria
    * (analisando a despesa de cada categoria), das matrículas por categoria
    * (analisando qual categoria mais procurada) ou dos códigos de impedimento
    * (verificando quais os maiores impedimentos).
    */
   public void gerarGrafico()
   }
}
```

CLASSE: HISTORICODOCLIENTE

```
/**
 * Classe dos históricos dos clientes. Mostra a situação de cada cliente dentro da
 * academia, tanto no módulo referente a matrículas, tanto na parte financeira de
 * cada um, podendo saber o que cada aluno pagou ou irá pagar dentro da
academia.
 */
public class HistoricodoCliente
{

  /**
   * Nome do cliente que será analisado o histórico.
   */
  private Cliente nomeCliente;

  /**
   * Descrição da atividade que o cliente está matriculado.
   */
  private Atividade descricaoAtividade;

  /**
   * Data da matrícula do cliente.
   */
  private date dataMatricula;

  /**
   * Valor da atividade que o cliente participa ou participou.
```

```
 */
private long valorAtividade;

/**
 * Data do pagamento.
 */
private date dataPagamento;

/**
 * Data do vencimento do pagamento.
 */
private date dataVencimento;
public Cliente apresentaHistorico;

/**
 */
public HistoricodoCliente()
{
}

/**
 * Visualiza a situação de cada aluno dentro da academia.
 * @return HistoricodoCliente
 */
public HistoricodoCliente visualizar()
{
 return null;
}
```

}

CLASSE: IMPEDIMENTO

```
/**
 * Classe de todos os códigos de impedimento que restringem o acesso de

 * clientes que tenham alguma pendência com a academia. Por exemplo:
 * matrícula vencidade, pendência de pagamento, exame médico vencido,
 * avaliação física vencida, avaliação nutriconal vencida, não matriculado, entre
 * outros.
 */
public class Impedimento
{

   /**
    * Código de impedimento.
    */
   private int codigoImpedimento;

   /**
    * Descrição do impedimento.
    */
   private char descricaoImpedimento;
   public SituacaodoCliente determina;

   /**
    */
   public Impedimento()
   {
```

```
}

/**
 * Cadastra o novo impedimento, informando seus dados que serão
 * armazenados na base de dados.
 * @return char
 */
public char cadastrarImpedimento()
{
 return 0;
}

/**
 * Consulta todos os impedimentos cadastrados.
 * @return Impedimento
 */
public Impedimento consultarImpedimentos()
{
 return null;
}

/**
 * Exclui um impedimento cadastrado.
 */
public void excluirImpedimento()
{

}
```

```
    /**
     * O administrador confirma a exclusão do impedimento.
     */
    public void confirmarExclusão()
    {

    }
}
```

CLASSE: INSTRUTOR

```
/**
 * Classe de todos os professores ou instrutores, ou seja, são cadastradas todas
 * as pessoas capacitadas a orientar pessoas que praticam alguma atividade na
 * academia ou mesmo em qualquer outro lugar.
 */
public class Instrutor
{

   /**
    * Código do professor.
    */
   private int codigoProfessor;

   /**
    * Nome do Professor.
    */
   private char nomeProfessor;

   /**
    * Endereço completo do professor.
    */
   private char enderecoProfessor;

   /**
    * Cidade onde mora o professor.
    */
```

```
private char cidadeProfessor;

/**
 * Estado da cidade do professor.
 */
private char estadoProfessor;

/**
 * CEP da cidade do professor.
 */
private char CEPProfessor;

/**
 * Telefone de contato do professor.
 */
private long foneProfessor;

/**
 * CPF do professor.
 */
private long CPFProfessor;

/**
 * Forma com que o professor  receberá, se é por aula ou mensalmente.
 */
private char formaRecebimento;

/**
```

```
    * Valor do pagamento do professor.
    */
   private short valorPagamento;
   public Atividade realiza;
   public ControledeAulas temControle;

   /**
    */
   public Instrutor()
   {

   }

   /**
    * Cadastra o professor, informando os dados que serão armazenados na base
de
    * dados.
    * @return char
    */
   public char cadastrarProfessor()
   {
    return 0;
   }

   /**
    * Consulta  todos os os professores cadastrados.
    * @return Instrutor
    */
```

```
    public Instrutor consultarProfessores()
    {
     return null;
    }

    /**
     * Exclui um professor cadastrado.
     */
    public void excluirProfessor()
    {
    }

    /**
     * O administrador confirma a exclusão do professor.
     */
    public void confirmarExclusao()
    {
    }
}
```

CLASSE: INSTRUTORATIVIDADE

```
/**
 * Esta é uma classe intermediária, pois há uma associação de muitos-pra-muitos
 * entre as classes Atividade e Instrutor.
 */
public class InstrutorAtividade
{

   /**
    * Código do professor.
    */
   private Instrutor codigoProfessor;

   /**
    * Código da atividade.
    */
   private Atividade codigoAtividade;

   /**
    */
   public InstrutorAtividade()
   {
   }
}
```

CLASSE: LOCAL

```
/**
 * Classe dos locais cadastrados.O administrador cadastra todas as dependências
 * da academia, por exemplo: piscina, sala de musculação, sala de ginástica.
 */
public class Local
{

   /**
    * Este código é criado para controle interno da academia.
    */
   private int codigoLocal;

   /**
    * Descrição do local.
    */
   private char descricaoLocal;

   /**
    */
   public Local()
   {
   }

   /**
    * Cadastra o novo local, informando os dados necessários que serão
```

```
 * armazenados na base de dados.
 * @return char
 */
public char cadastrarLocal()
{
 return 0;
}

/**
 * Consulta todos os locais cadastrados.
 * @return Local
 */
public Local consultarLocais()
{
 return null;
}

/**
 * Exclui um cliente cadastrado.
 */
public void excluirLocal()
{
}

/**
 * O administrador confirma a exclusão do local.
 */
public void confirmarExclusao()
```

```
    {
    }
}
```

CLASSE: MATRICULADECLIENTE

```
/**
 * Esta é uma classe intermediária, pois há uma relação de muitos-pra-muitos
 * entre as classe Atividade e Cliente.
 */
public class MatriculadeCliente
{

   /**
    * Código do cliente a ser matriculado.
    */
   private Cliente codigoCliente;

   /**
    * Informa o tipo de mensalidade. Se a modalidade de cobrança for do tipo
    * "Mensalidade" pede que seja informado também um dia de pagamento.
    * Todos os meses será gerada uma conta a receber em favor da academia,
    * tendo o matriculado como sacado. Caso a modalidade seja "Contrato" será
    * preciso informar o início da validade do contrato e sua duração (em meses).
    */
   private char tipoMensalidade;

   /**
    * Indica o código em qual atividade o cliente está sendo matriculado.
    */
   private Atividade codigoAtividade;
```

```
/**
 * Valor que o cliente pagará no decorrer da prática da atividade.
 */
private long valor;

/**
 * Date que foi realizada a matrícula do cliente na atividade.
 */
private date dataMatricula;
public Atividade matricula;
public Cliente matriculado;

/**
 */
public MatriculadeCliente()
{
}

/**
 * Matricula o cliente em alguma atividade.
 * @return Cliente
 */
public Cliente matricularCliente()
{
 return null;
}

/**
```

```
    * Cancela a matrícula de um cliente em alguma atividade, excluindo a
    * atividade da base de dados do cliente.
    */
   public void cancelarMatricula()
   {
   }
}
```

MOVIMENTODECONTACORRENTE

```
/**
 * Consulta os últimos lançamentos das contas correntes cadastradas. Exemplos
 * de lançamentos podem ser: pagamento de conta de luz, conta de telefone,
 * entre outros.
 */
public class MovimentodeContaCorrente
{

   /**
    * Descrição do Banco que sofrerá movimentação.
    */
   private Banco descricaoBanco;

   /**
    * Data do movimento.
    */
   private date dataMovimento;

   /**
     * Um histórico como, conta de luz, conta de telefone, conta de água, entre
outros.
    */
   private char historico;

   /**
    * Valor do débito
```

```
 */
private long valorDebito;

/**
 * Valor do crédito
 */
private long valorCredito;

/**
 * Saldo
 */
private long saldo;
public ContaCorrente movimenta;

/**
 */
public MovimentodeContaCorrente()
{
}

/**
 */
public void movimentar()
{
}

/**
 * O administrador confirma o lançamento que será processado.
```

```
     */
    public void confirmaLancamento()
    {
    }
}
```

CLASSE: PRODUTO

```
/**
 * O administrador cadastra todos os produtos e serviços que a academia oferece
 * aos seus clientes, por exemplo: roupa de ginástica, óculos para natação,
 * tornozeleira, salgados, luvas, etc;
 */
public class Produto
{

   /**
    * Código do produto
    */
   private int codigoProduto;

   /**
    * Categoria que pertence o produto.
    */
   private Categoria categoriaProduto;

   /**
    * Descrição do produto.
    */
   private char descricaoProduto;

   /**
    * Preço do produto.
    */
```

```
   private char precoProduto;

   /**
    * Nome do fornecedor que fornece o produto.
    */
   private Fornecedor fornecedorProduto = initval;
   public Categoria pertence[];
   public Fornecedor fornecido;
   public Venda serVendido[];
   public Categoria theCategoria;
   public Fornecedor theFornecedor;
   public Venda theVenda;

   /**
    */
   public Produto()
   {
   }

   /**
    * Cadastra o novo produto, informando os dados necessários que serão
armazenados
    * na base de dados.
    * @return char
    */
   public char cadastrarProduto()
   {
    return 0;
```

```
  }

  /**
   * Consulta todos os produtos cadastrados.
   * @return Produto
   */
  public Produto consultarProdutos()
  {
   return null;
  }

  /**
   * Exclui um produto cadastrado.
   */
  public void excluirProduto()
  {
  }

  /**
   * O administrador confirma a exclusão do produto.
   */
  public void confirmarExclusao()
  {
  }
}
```

CLASSE: SITUACAODOCLIENTE

```
/**
 * Esta classe armazena mostra se o acesso do cliente à academia está liberado
 * ou impedido, ou seja, verifica se há algum código de  impedimento cadastrado
 * ou não.
 */
public class SituacaodoCliente
{

   /**
    * Código do cliente.
    */
   private int codigoCliente;

   /**
    * Nome do cliente.
    */
   private Cliente nomeCliente;

   /**
    * Descrição do código de impedimento.
    */
   private Impedimento descricaoImpedimento;

   /**
    * Data da matrícula do cliente.
    */
```

```
private char dataMatricula;

/**
 * Data do Exame Médico do cliente.
 */
private char dataExameMedico;

/**
 * Data da Avaliação Física
 */
private char dataAvaliacaoFisica;

/**
 * Data da Avaliação Nutricional.
 */
private char dataAvaliacaoNutricional;
public Cliente mostraSituacao[];
public Impedimento apresentaImpedimento[];

/**
 */
public SituacaodoCliente()
{
}

/**
 * Verifica a situação do cliente, ou seja, se há algum motivo que o impeça de
 * participar das atividades da academia.
```

```
     * @return SituacaodoCliente
     */
    public SituacaodoCliente verificar()
    {
     return null;
    }

    /**
     * Cadastra na situação do cliente um impedimento.
     * @return char
     */
    public char cadastrarImpedimento()
    {
     return 0;
    }
}
```

CLASSE: TRANSFERENCIACONTACORRENTE

```
/**
 * Esta é uma classe intermediária, pois há uma associação de muitos-pra-muitos
 * entre as classes Conta Corrente e Transferencia.
 */
public class TransferenciaContaCorrente
{
   private ContaCorrente codigoContaCorrente;
   private Transferencia codigoTransferencia;
   /**
    */
   public TransferenciaContaCorrente()
   {
   }
}
```

CLASSE: USUARIO

```
/**
 * Classe dos usuários cadastrados.
 */
public class Usuario
{

   /**
    * Código do usuário.
    */
   private int codigoUsuario;

   /**
    * Nome do usuário.
    */
   private char nomeUsuario;

   /**
    * Senha de acesso ao sistema do usuário.
    */
   private char senhaUsuario;

   /**
    * Opções que serão permitidas acesso do usuário.
    */
   private char opcoesAcesso;
```

```
  /**
   */
  public Usuario()
  {
  }

  /**
   * Cadastra o novo usuário, informando os dados necessário que serão armazenados
   * na base de dados.
   * @return char
   */
  public char cadastrarUsuario()
  {
   return 0;
  }

  /**
   * Consulta todos os usuários cadastrados.
   * @return Usuario
   */
  public Usuario consultarUsuarios()
  {
   return null;
  }

  /**
   * Exclui um usuário cadastrado.
```

```
     */
    public void excluirUsuario()
    {
    }

    /**
     * O administrador confirma a exclusão do usuário.
     */
    public void confirmarExclusao()
    {
    }
}
```

CLASSE: VENDA

```
/**
 * Classe das vendas realizadas.
 */
public class Venda
{

   /**
    * Especifica um código para a venda para o controle interno da academia.
    */
   private int codigoVenda;

   /**
    * Descrição do produto vendido.
    */
   private Produto descricaoProduto;

   /**
    * Nome do cliente.,
    */
   private Cliente nomeCliente;

   /**
    * Valor total da compra
    */
   private long valorCompra;
```

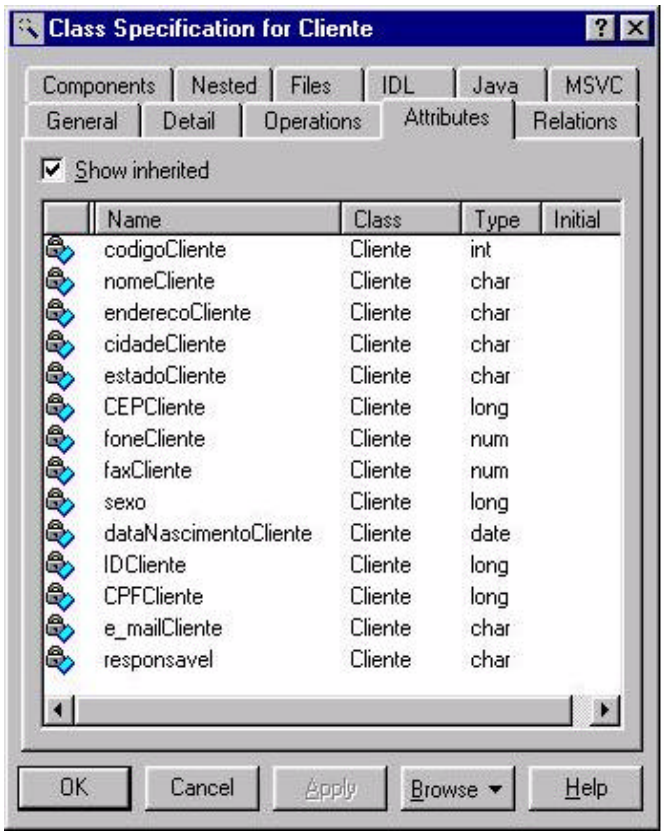
```
  /**
   * Quantidade do produto que o cliente adquiriu.
   */
  private int quantidadeProduto;

  /**
   * Quantidade do produto que está sendo adquirido.
   */
  private int quantidade;
  public Cliente realizada;
  public Produto podeTer[];
  public Produto theProduto;

  /**
   */
  public Venda()
  {
  }

  /**
   * Realiza uma venda simples a um cliente.
   */
  public void realizarVenda()
  {
  }
}
```

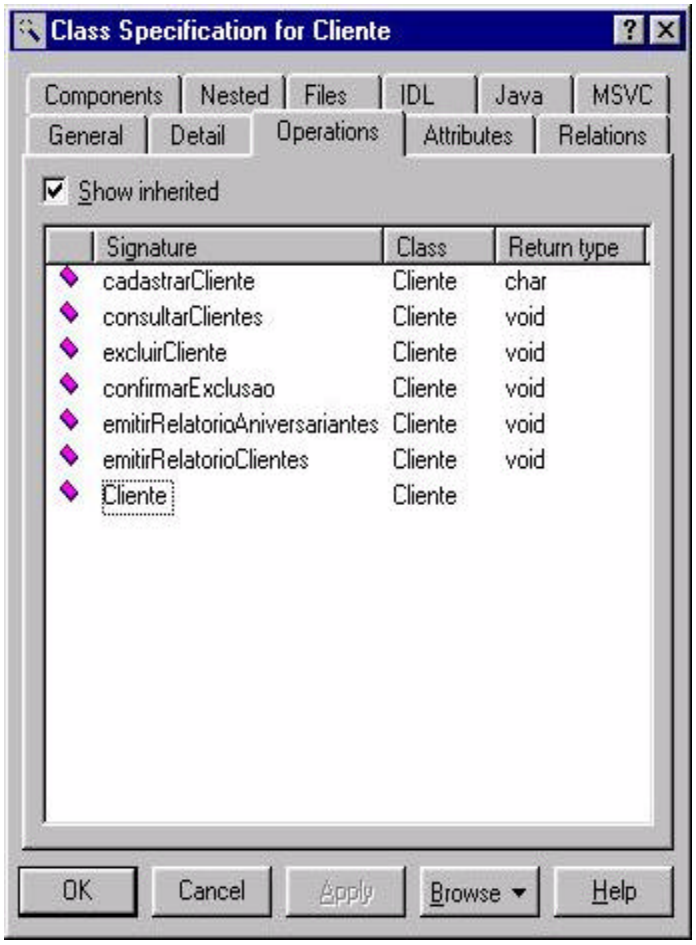
CLASSE: VENDAPRODUTO

```
/**
 * Esta é uma classe intermediária, pois há uma relação de muitos-pra-muitos
 * entre as classes Venda e Produto.
 */
public class VendaProduto
{

   /**
    * Código do produto.
    */
   private Produto codigoProduto;

   /**
    * Codigo da venda.
    */
   private Venda codigoVenda;

   /**
    */
   public VendaProduto()
   {
   }
```

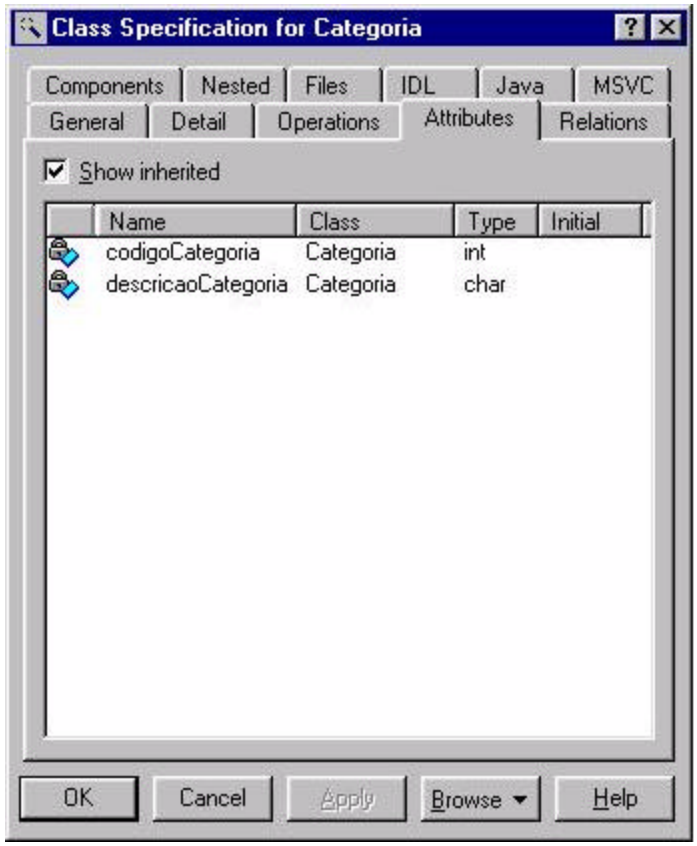
ANEXO B

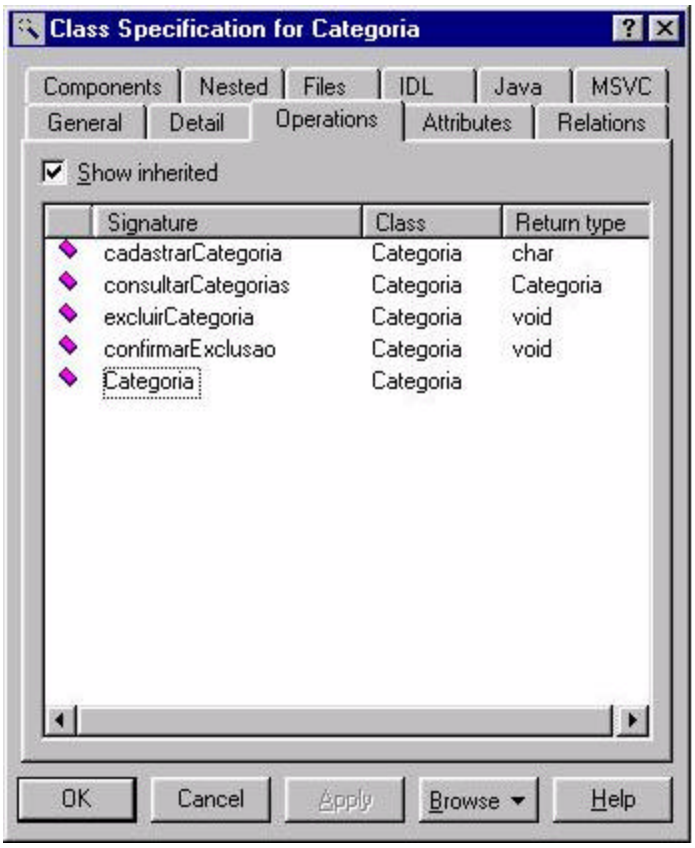
- **CLASSE : CLIENTE**



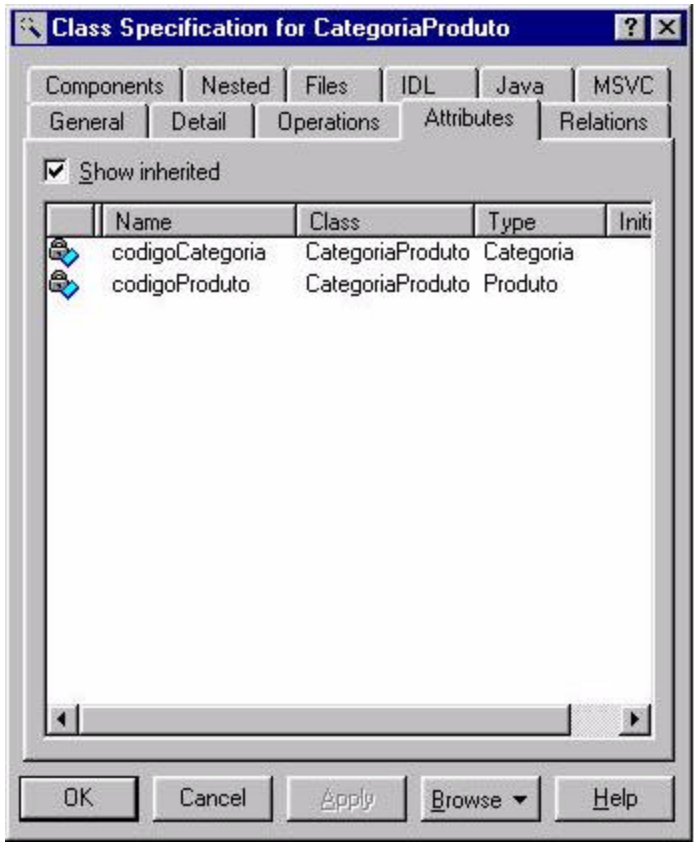


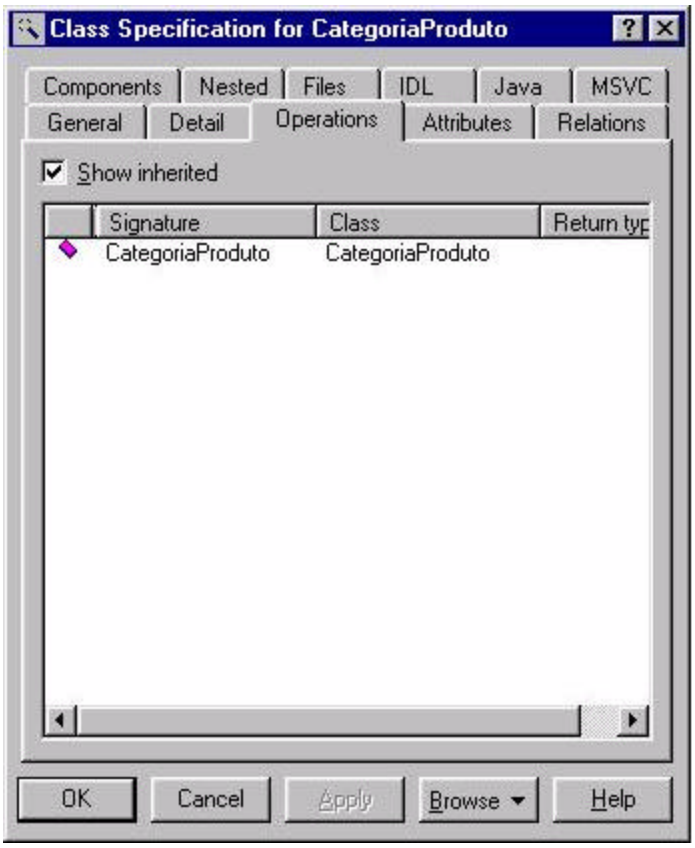
- **CLASSE: CATEGORIA**


Class Specification for Categoria
Components
Nested
Files
IDL
Java
MSVC
General
Detail
Operations
Attributes
Relations
Show inherited
Name
Class
Type
Initial
codigoCategoria
Categoria
int
descricaoCategoria
Categoria
char
OK
Cancel
Apply
Browse
Help

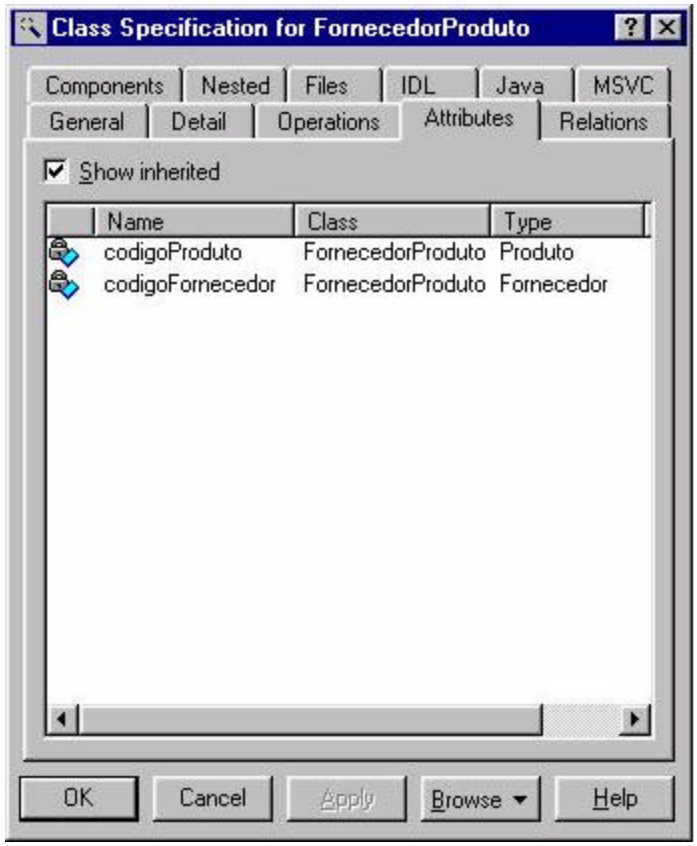


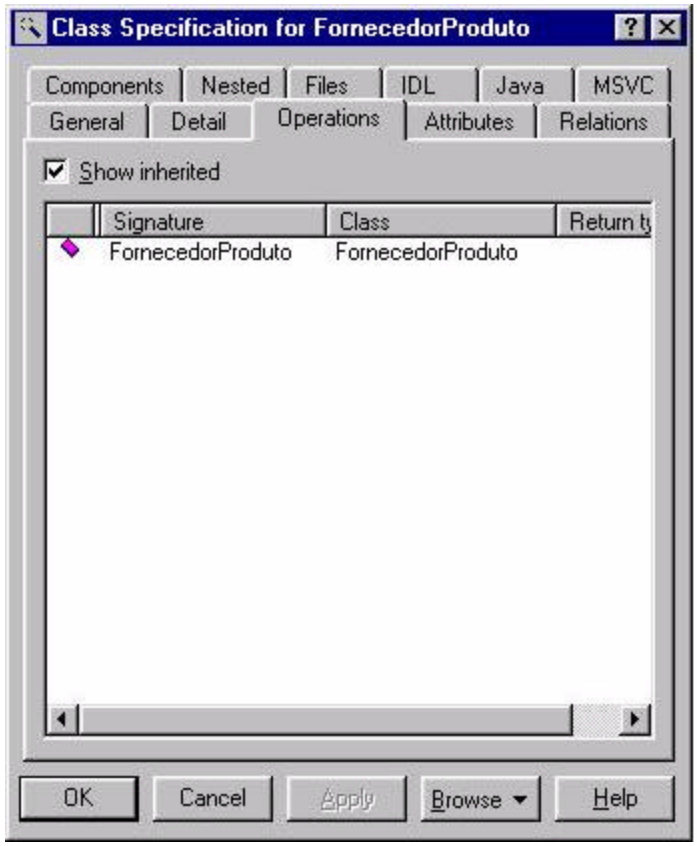
- **CLASSE: CATEGORIAPRODUTO**


Class Specification for CategoriaProduto
Components
Nested
Files
IDL
Java
MSVC
General
Detail
Operations
Attributes
Relations
Show inherited
Name
Class
Type
Initi
codigoCategoria
CategoriaProduto
Categoria
codigoProduto
CategoriaProduto
Produto
OK
Cancel
Apply
Browse
Help

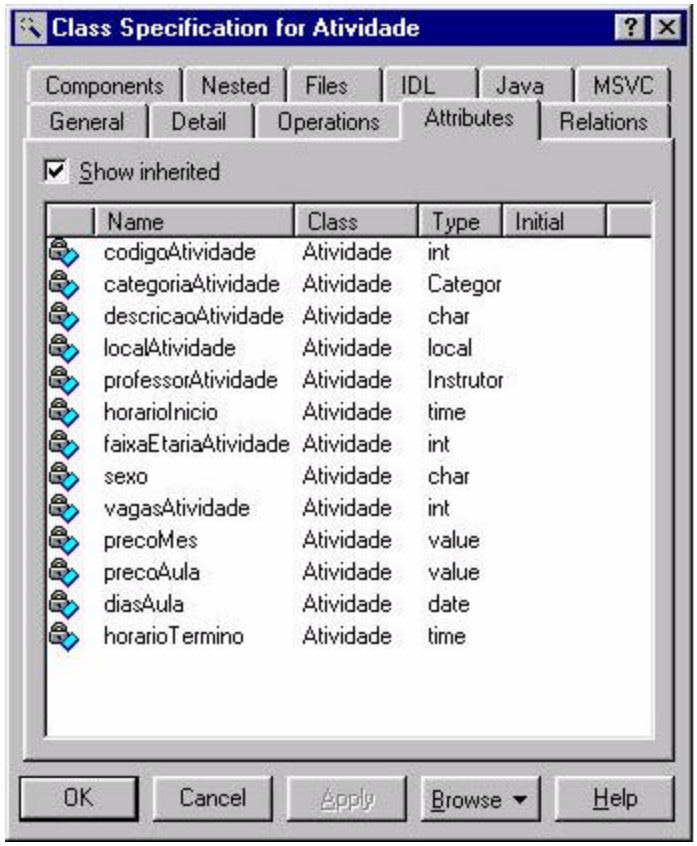


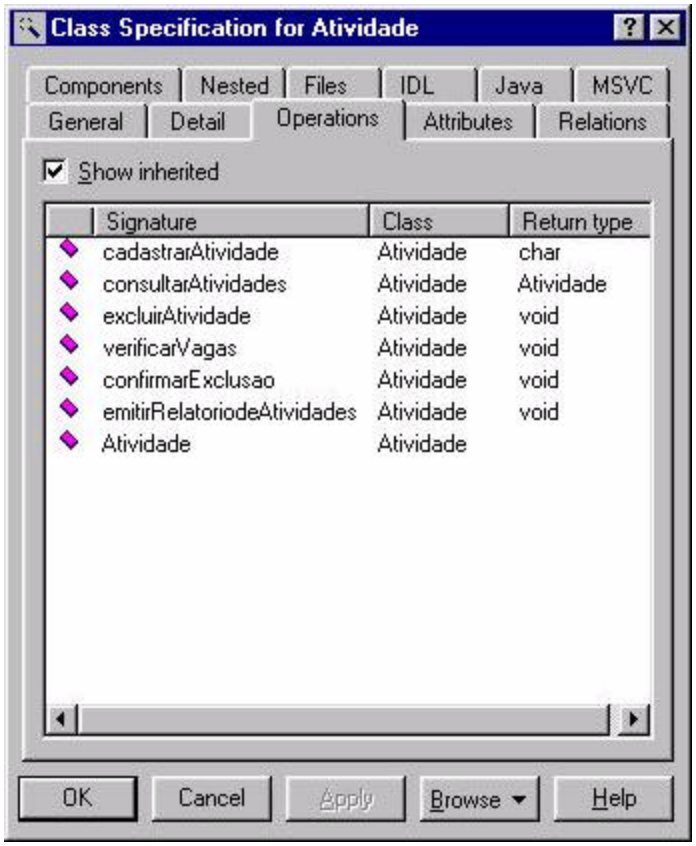
- **CLASSE: FORNECEDORPRODUTO**


Class Specification for FornecedorProduto
Components
Nested
Files
IDL
Java
MSVC
General
Detail
Operations
Attributes
Relations
Show inherited
Name
Class
Type
codigoProduto
FornecedorProduto
Produto
codigoFornecedor
FornecedorProduto
Fornecedor
OK
Cancel
Apply
Browse
Help

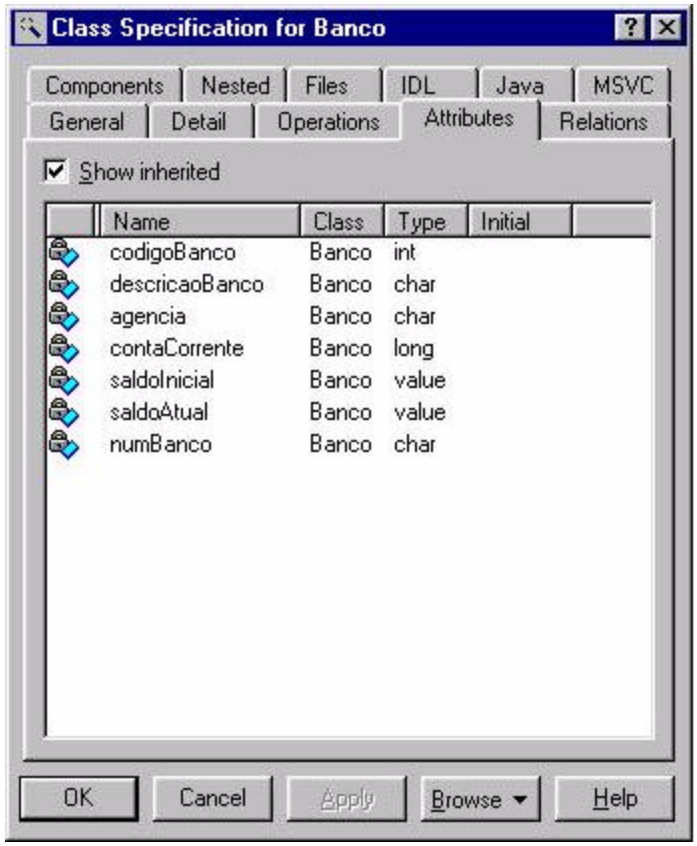


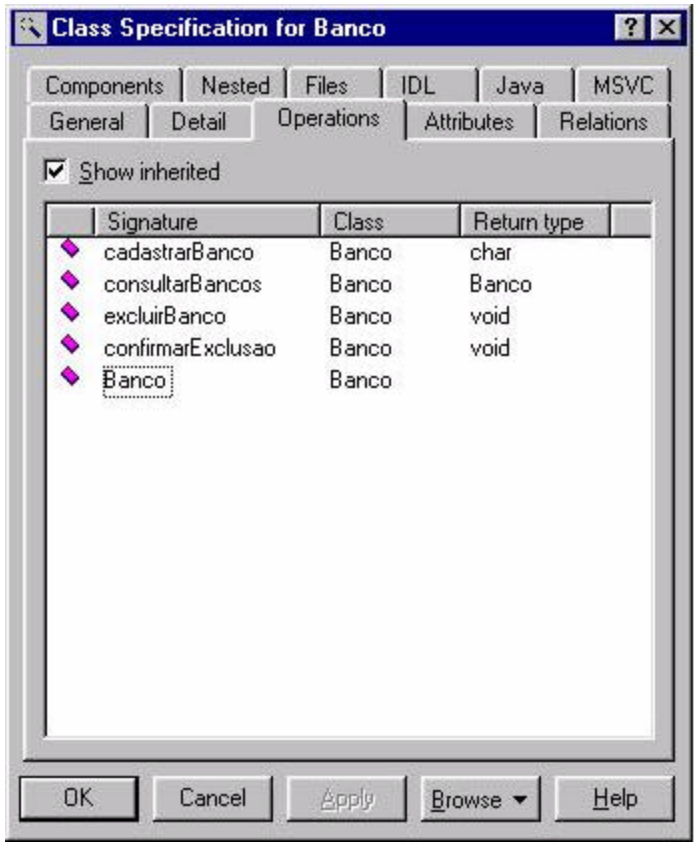
- **CLASSE: ATIVIDADE**


Class Specification for Atividade
Components
Nested
Files
IDL
Java
MSVC
General
Detail
Operations
Attributes
Relations
Show inherited
Name
Class
Type
Initial
codigoAtividade Atividade int
categoriaAtividade Atividade Categor
descricaoAtividade Atividade char
localAtividade Atividade local
professorAtividade Atividade Instrutor
horarioInicio Atividade time
faixaEtariaAtividade Atividade int
sexo Atividade char
vagasAtividade Atividade int
precoMes Atividade value
precoAula Atividade value
diasAula Atividade date
horarioTermino Atividade time
OK
Cancel
Apply
Browse
Help

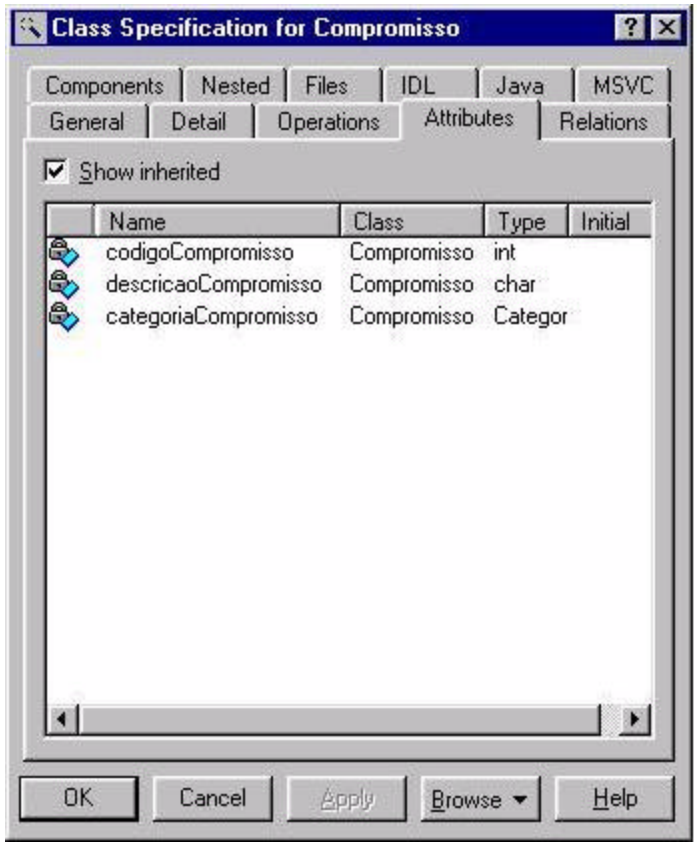


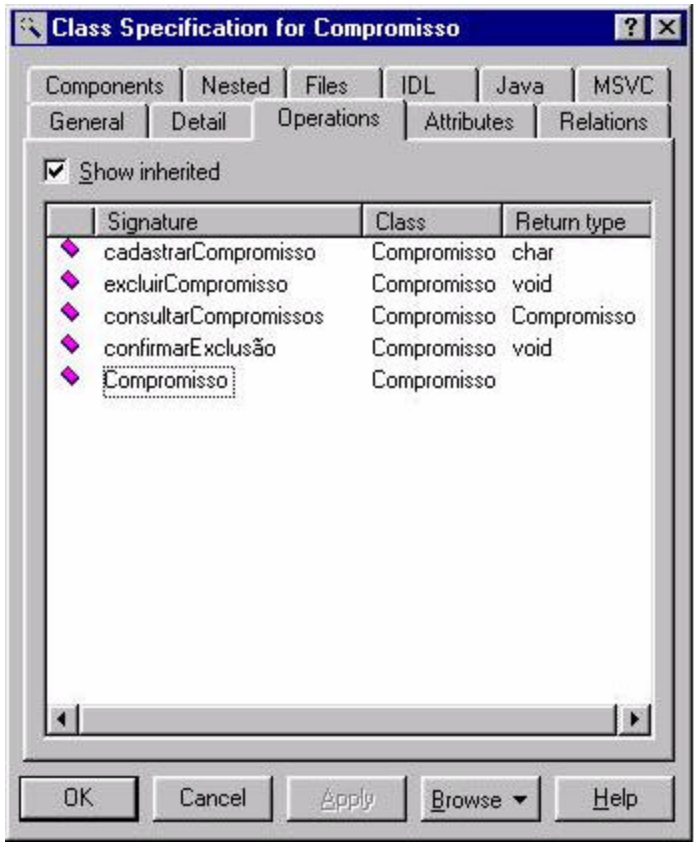
- **CLASSE: BANCO**


Class Specification for Banco
Components
Nested
Files
IDL
Java
MSVC
General
Detail
Operations
Attributes
Relations
Show inherited
Name
Class
Type
Initial
codigoBanco Banco int
descricaoBanco Banco char
agencia Banco char
contaCorrente Banco long
saldoInicial Banco value
saldoAtual Banco value
numBanco Banco char
OK
Cancel
Apply
Browse
Help

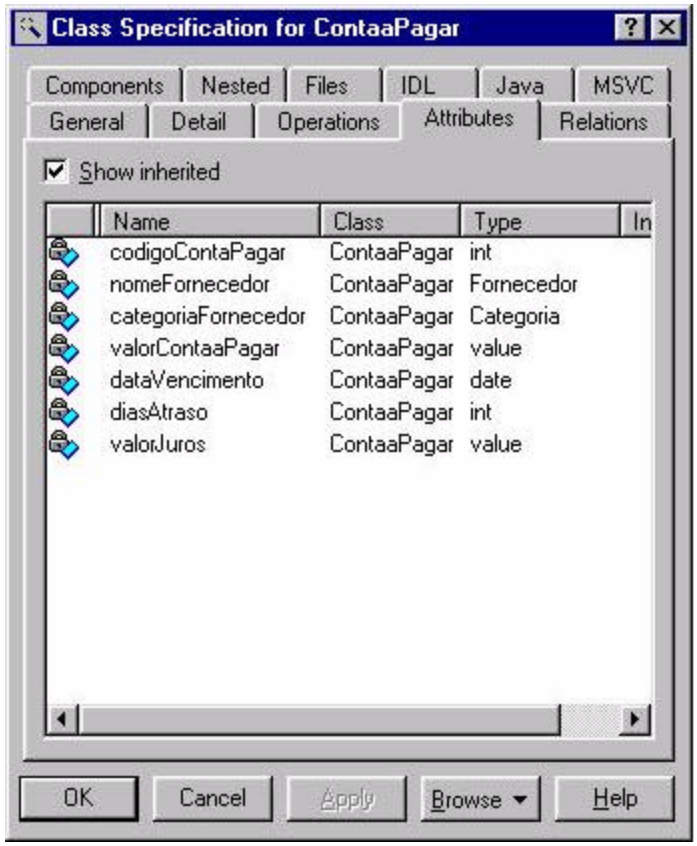


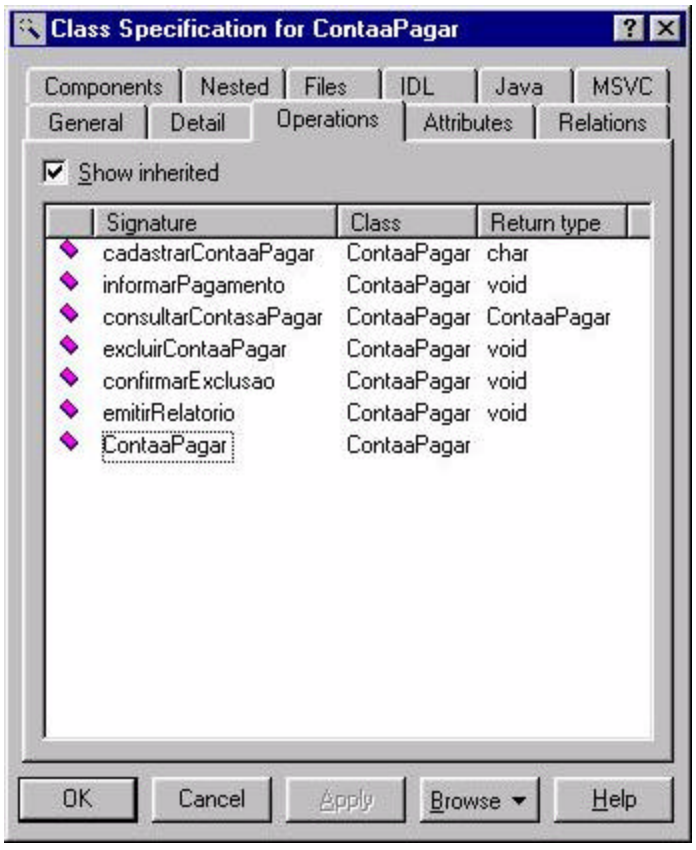
- **CLASSE: COMPROMISSO**


Class Specification for Compromisso
Components
Nested
Files
IDL
Java
MSVC
General
Detail
Operations
Attributes
Relations
Show inherited
Name
Class
Type
Initial
codigoCompromisso
Compromisso
int
descricaoCompromisso
Compromisso
char
categoriaCompromisso
Compromisso
Categor
OK
Cancel
Apply
Browse
Help

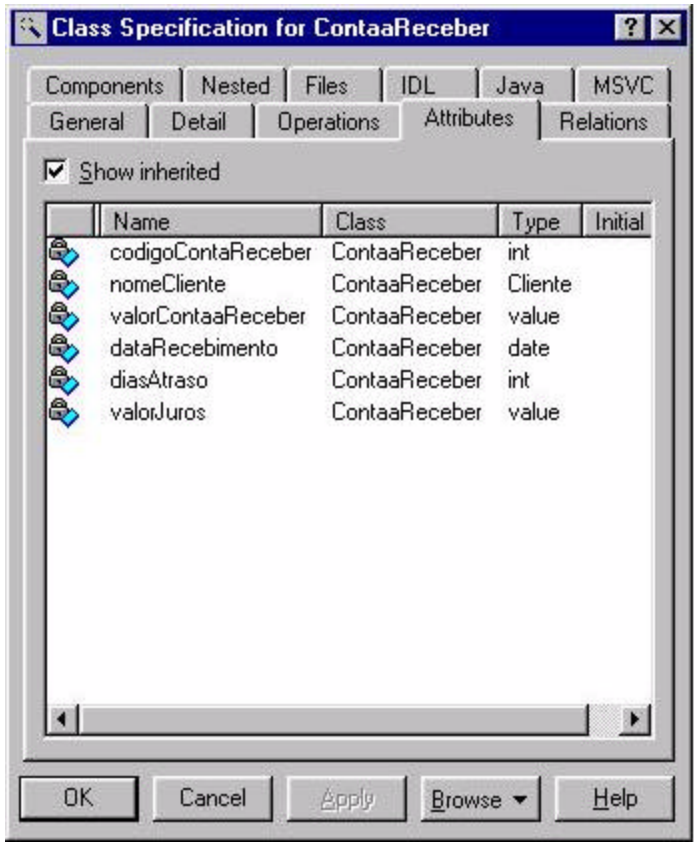


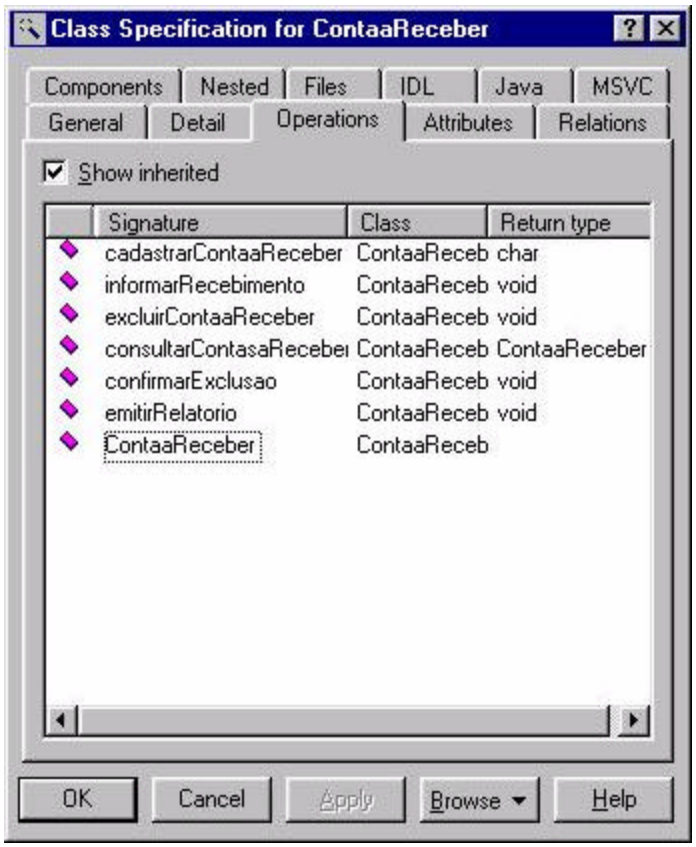
- **CLASSE: CONTAAPAGAR**


Class Specification for ContaaPagar
Components
Nested
Files
IDL
Java
MSVC
General
Detail
Operations
Attributes
Relations
Show inherited
Name
Class
Type
In
codigoContaPagar ContaaPagar int
nomeFornecedor ContaaPagar Fornecedor
categoriaFornecedor ContaaPagar Categoria
valorContaaPagar ContaaPagar value
dataVencimento ContaaPagar date
diasAtraso ContaaPagar int
valorJuros ContaaPagar value
OK
Cancel
Apply
Browse
Help

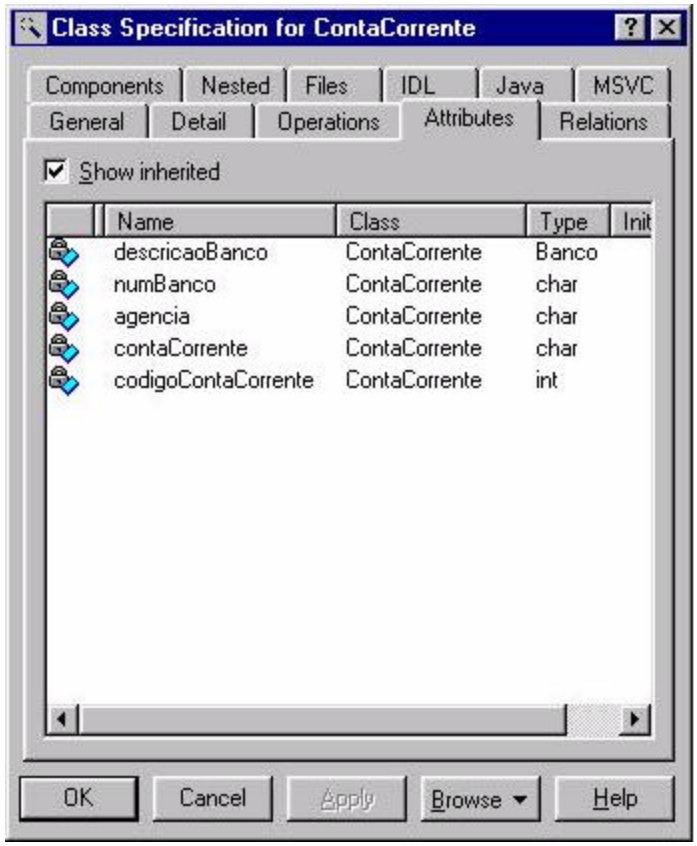


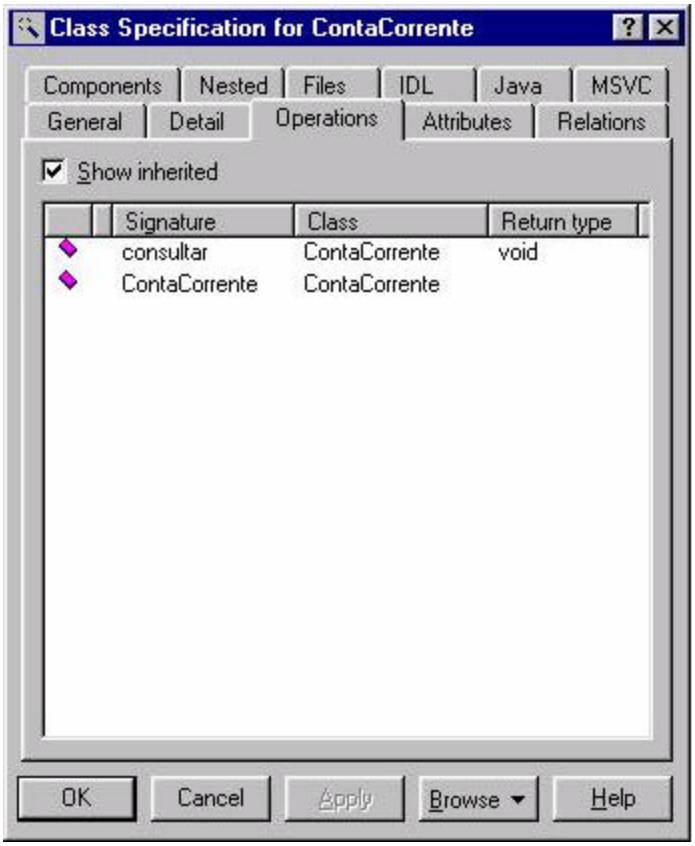
- **CLASSE: CONTAARECEBER**


Class Specification for ContaaReceber
Components
Nested
Files
IDL
Java
MSVC
General
Detail
Operations
Attributes
Relations
Show inherited
Name
Class
Type
Initial
codigoContaReceber
ContaaReceber
int
nomeCliente
ContaaReceber
Cliente
valorContaaReceber
ContaaReceber
value
dataRecebimento
ContaaReceber
date
diasAtraso
ContaaReceber
int
valorJuros
ContaaReceber
value
OK
Cancel
Apply
Browse
Help

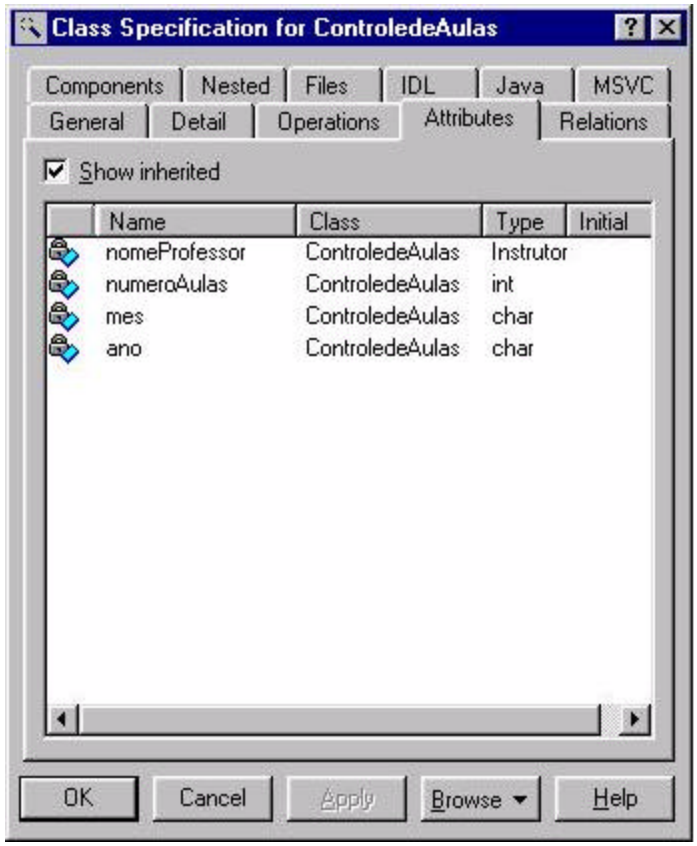


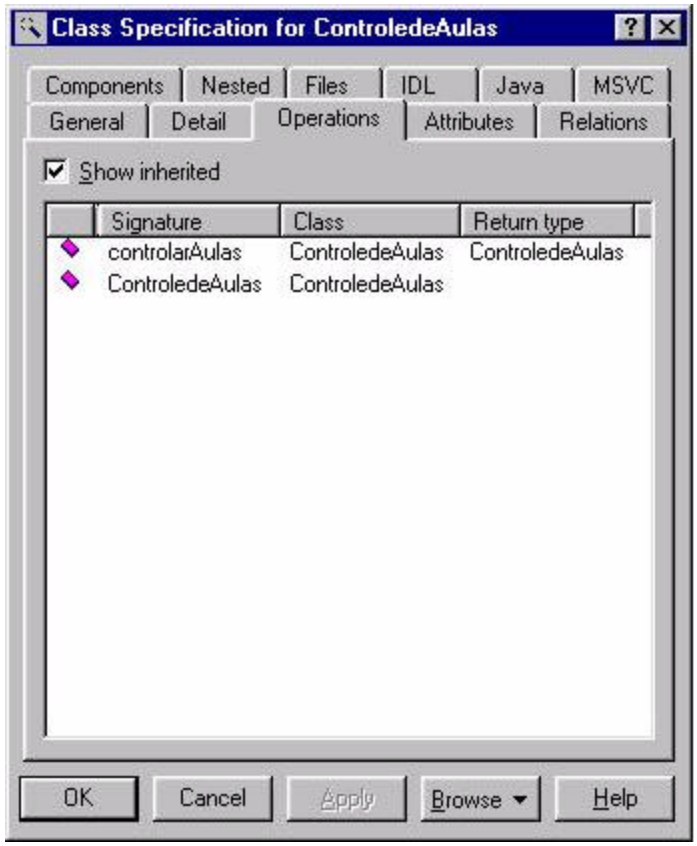
- **CLASSE: CONTACORRENTE**


Class Specification for ContaCorrente
Components
Nested
Files
IDL
Java
MSVC
General
Detail
Operations
Attributes
Relations
Show inherited
Name
Class
Type
Init
descricaoBanco
ContaCorrente
Banco
numBanco
ContaCorrente
char
agencia
ContaCorrente
char
contaCorrente
ContaCorrente
char
codigoContaCorrente
ContaCorrente
int
OK
Cancel
Apply
Browse
Help

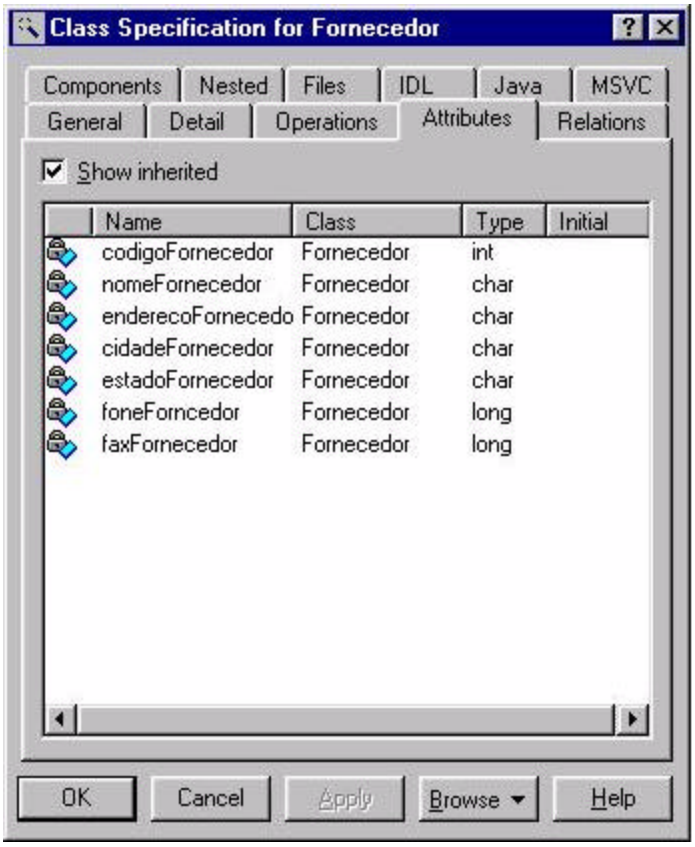


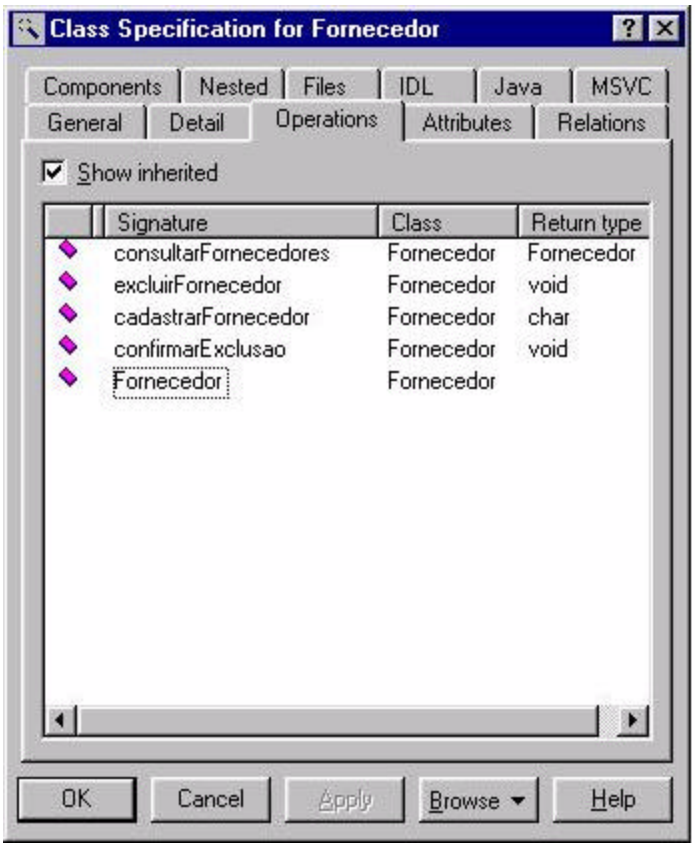
- **CLASSE: CONTROLEDEAULAS**


Class Specification for ControledeAulas
Components
Nested
Files
IDL
Java
MSVC
General
Detail
Operations
Attributes
Relations
Show inherited
Name
Class
Type
Initial
nomeProfessor
ControledeAulas
Instrutor
numeroAulas
ControledeAulas
int
mes
ControledeAulas
char
ano
ControledeAulas
char
OK
Cancel
Apply
Browse
Help

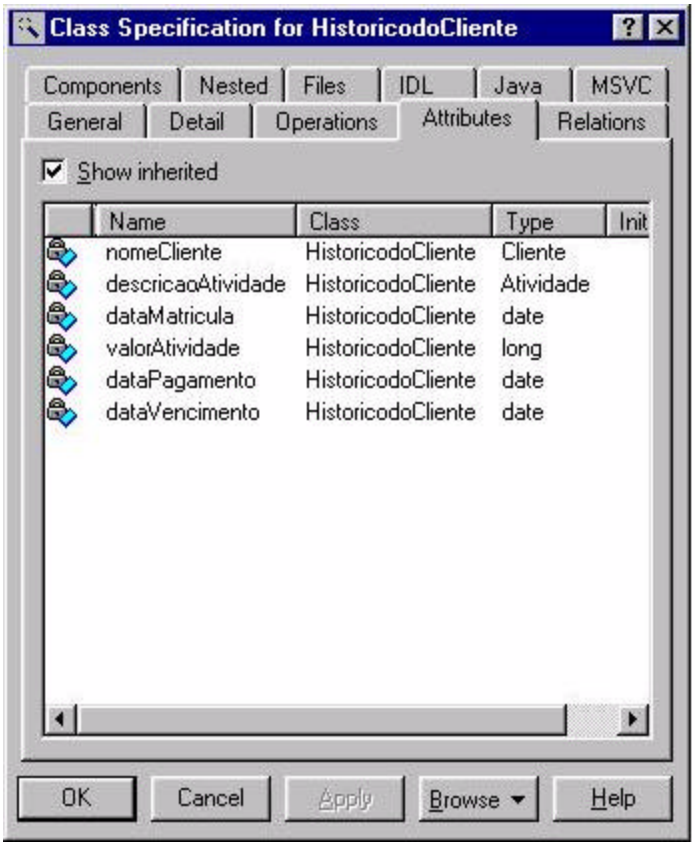

Class Specification for ControledeAulas
Components
Nested
Files
IDL
Java
MSVC
General
Detail
Operations
Attributes
Relations
Show inherited
Signature
Class
Return type
controlarAulas
ControledeAulas
ControledeAulas
ControledeAulas
ControledeAulas
OK
Cancel
Apply
Browse
Help

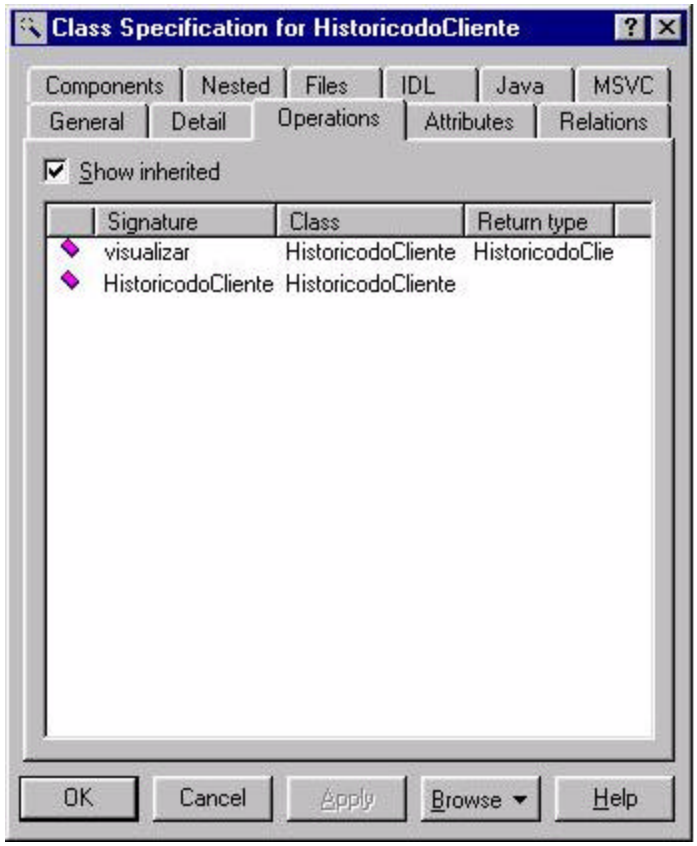
- **CLASSE: FORNECEDOR**



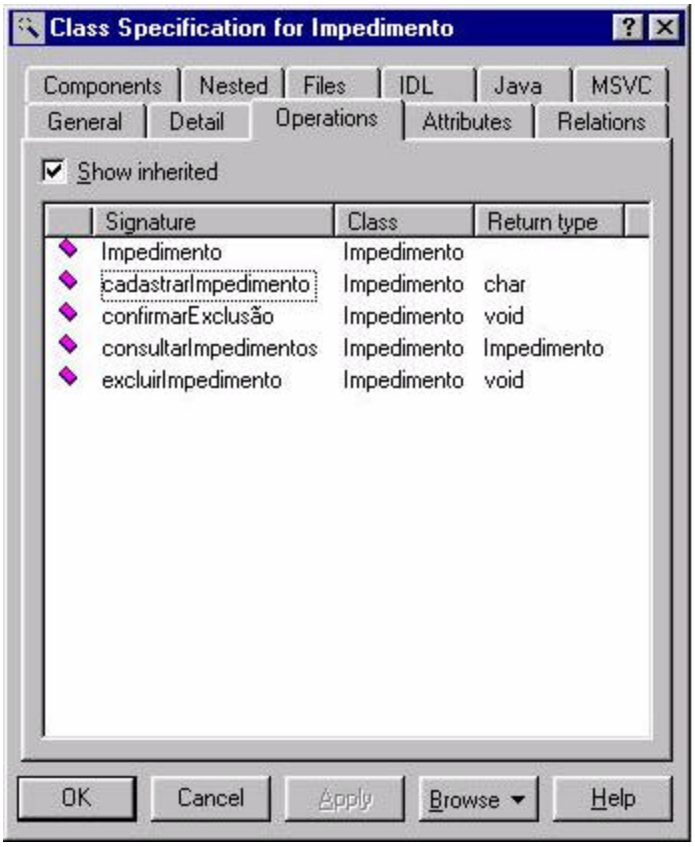

Class Specification for Fornecedor
Components
Nested
Files
IDL
Java
MSVC
General
Detail
Operations
Attributes
Relations
Show inherited
Signature
Class
Return type
consultarFornecedores
Fornecedor
Fornecedor
excluirFornecedor
Fornecedor
void
cadastrarFornecedor
Fornecedor
char
confirmarExclusao
Fornecedor
void
Fornecedor
Fornecedor
OK
Cancel
Apply
Browse
Help

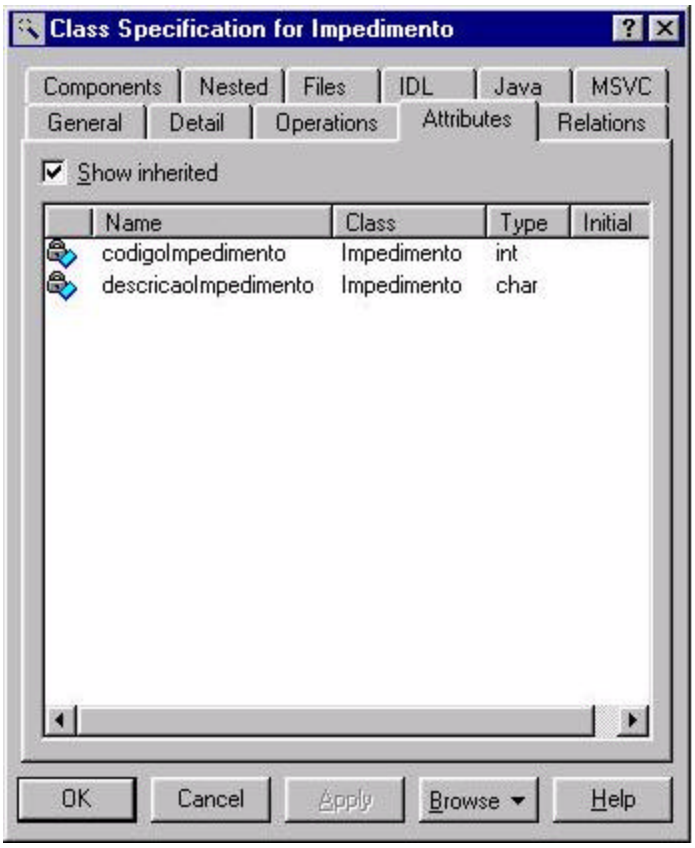
- **CLASSE: HISTORICODOCLIENTE**



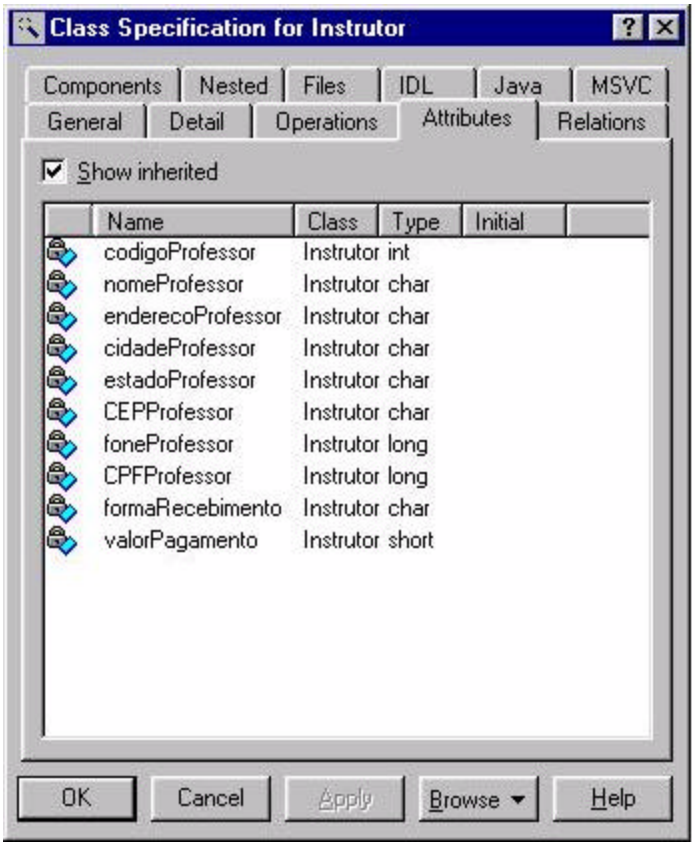

Class Specification for HistoricodoCliente
Components
Nested
Files
IDL
Java
MSVC
General
Detail
Operations
Attributes
Relations
Show inherited
Signature
Class
Return type
visualizar
HistoricodoCliente
HistoricodoClie
HistoricodoCliente
HistoricodoCliente
OK
Cancel
Apply
Browse
Help

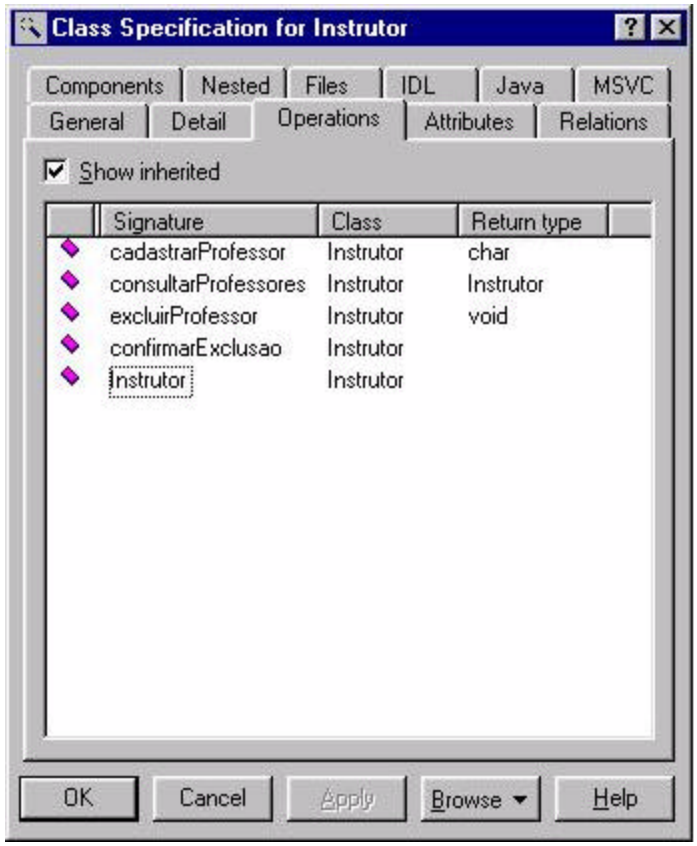
- **CLASSE: IMPEDIMENTO**



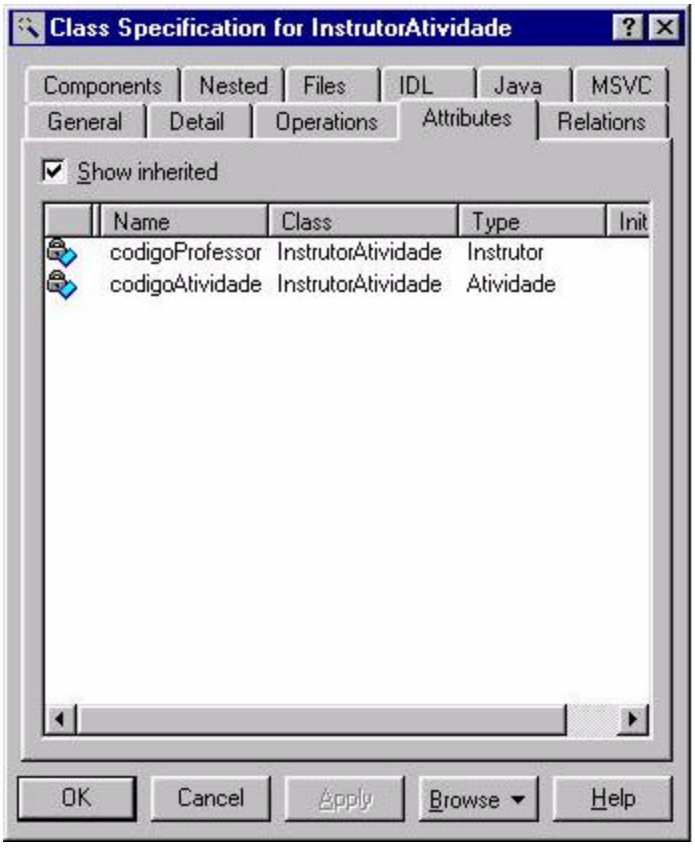

Class Specification for Impedimento
Components
Nested
Files
IDL
Java
MSVC
General
Detail
Operations
Attributes
Relations
Show inherited
Name
Class
Type
Initial
codigoImpedimento
Impedimento
int
descricaoImpedimento
Impedimento
char
OK
Cancel
Apply
Browse
Help

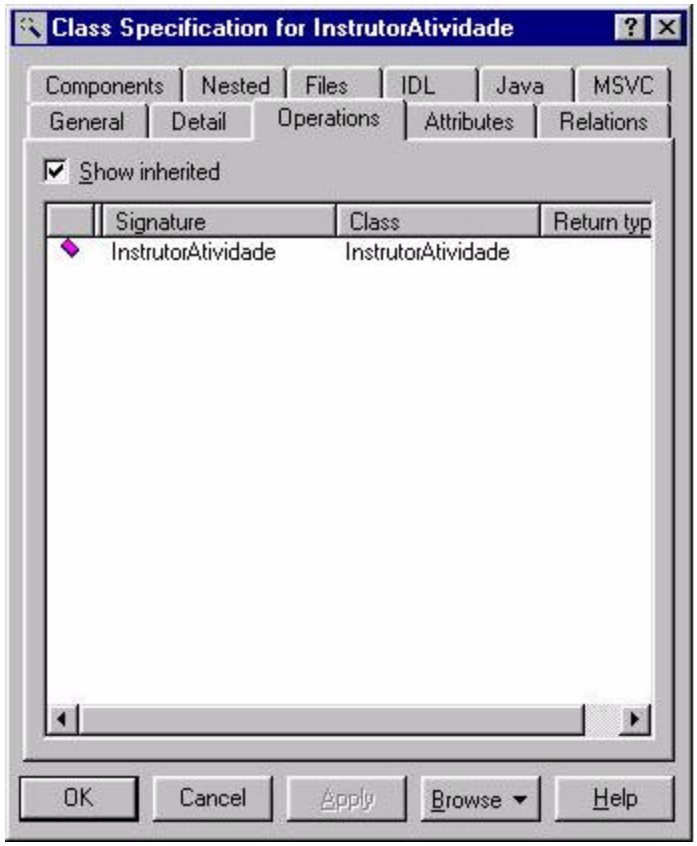
- **CLASSE: INSTRUTOR**



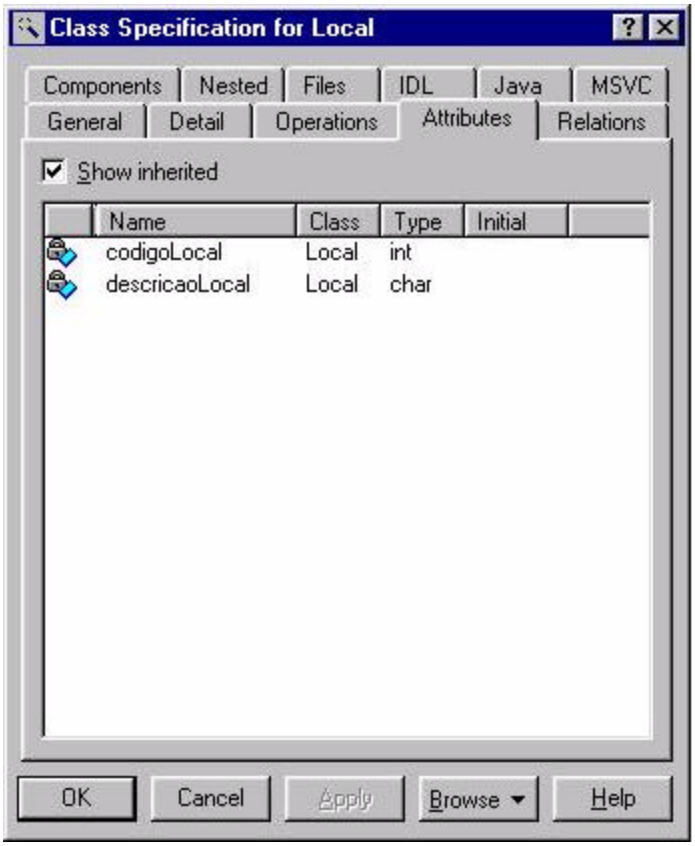

Class Specification for Instrutor
Components
Nested
Files
IDL
Java
MSVC
General
Detail
Operations
Attributes
Relations
Show inherited
Signature
Class
Return type
cadastrarProfessor
Instrutor
char
consultarProfessores
Instrutor
Instrutor
excluirProfessor
Instrutor
void
confirmarExclusao
Instrutor
Instrutor
Instrutor
OK
Cancel
Apply
Browse
Help

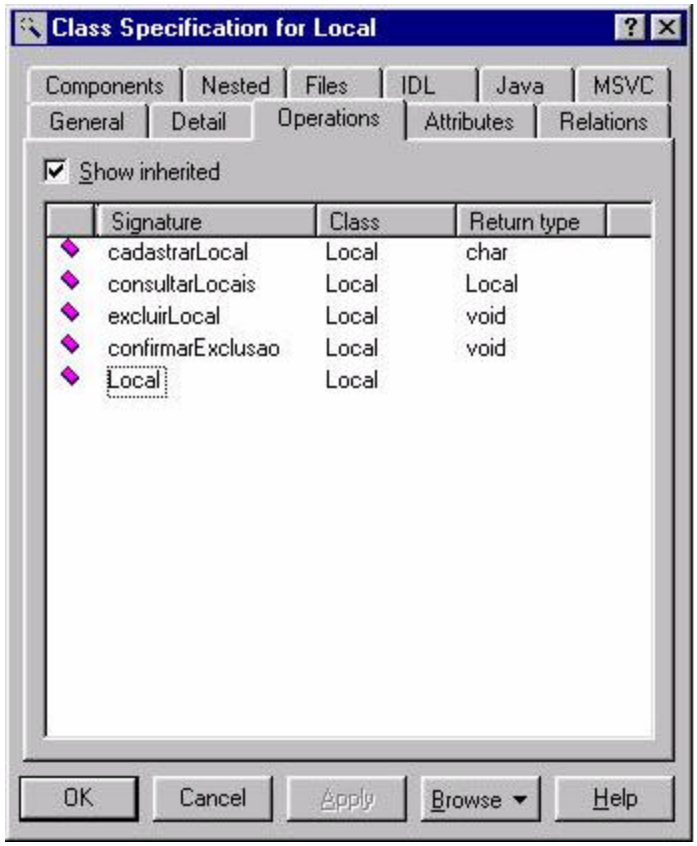
- **CLASSE: INSTRUTORATIVIDADE**



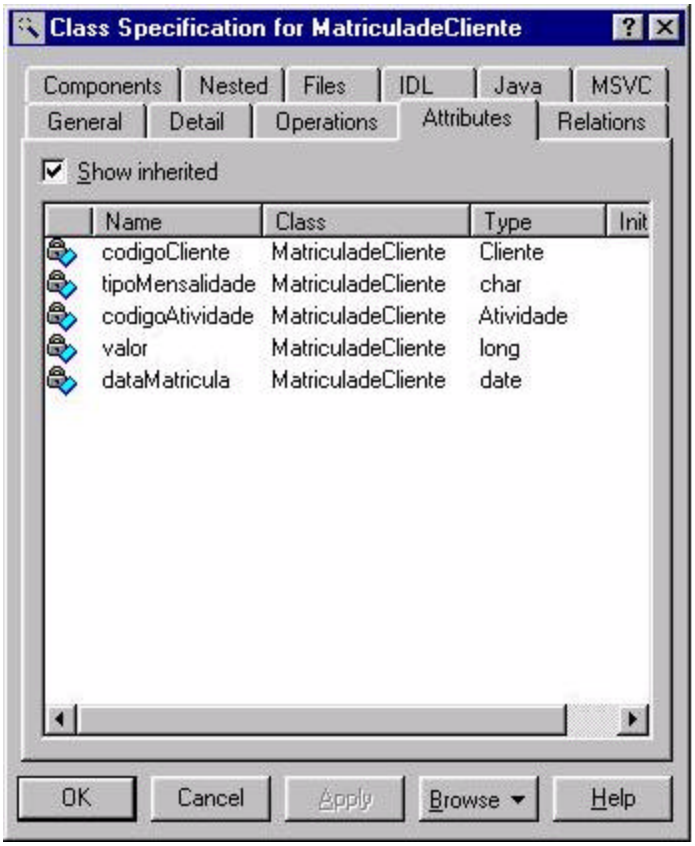

Class Specification for InstrutorAtividade
Components
Nested
Files
IDL
Java
MSVC
General
Detail
Operations
Attributes
Relations
Show inherited
Signature
Class
Return typ
InstrutorAtividade
InstrutorAtividade
OK
Cancel
Apply
Browse
Help

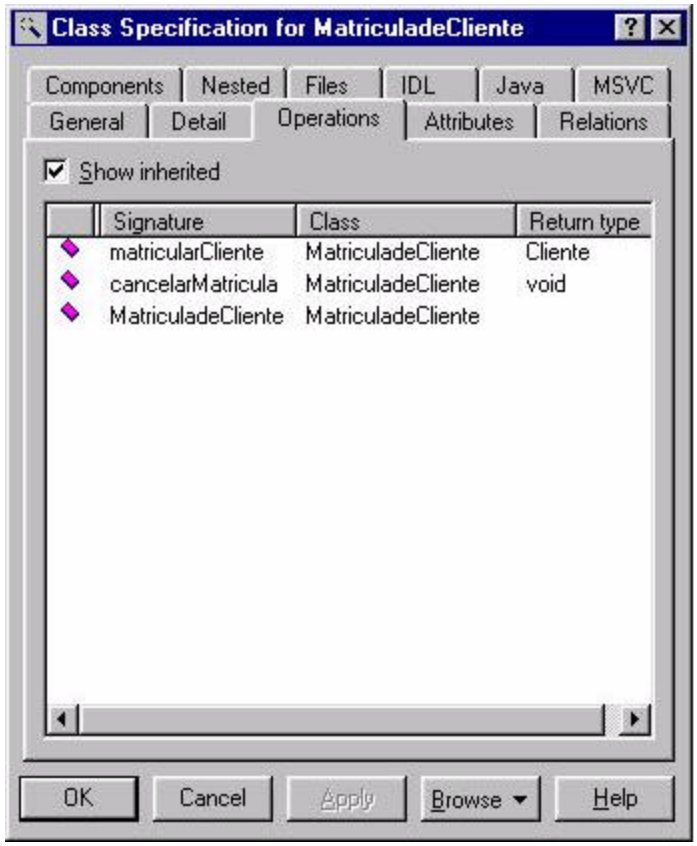
- **CLASSE: LOCAL**



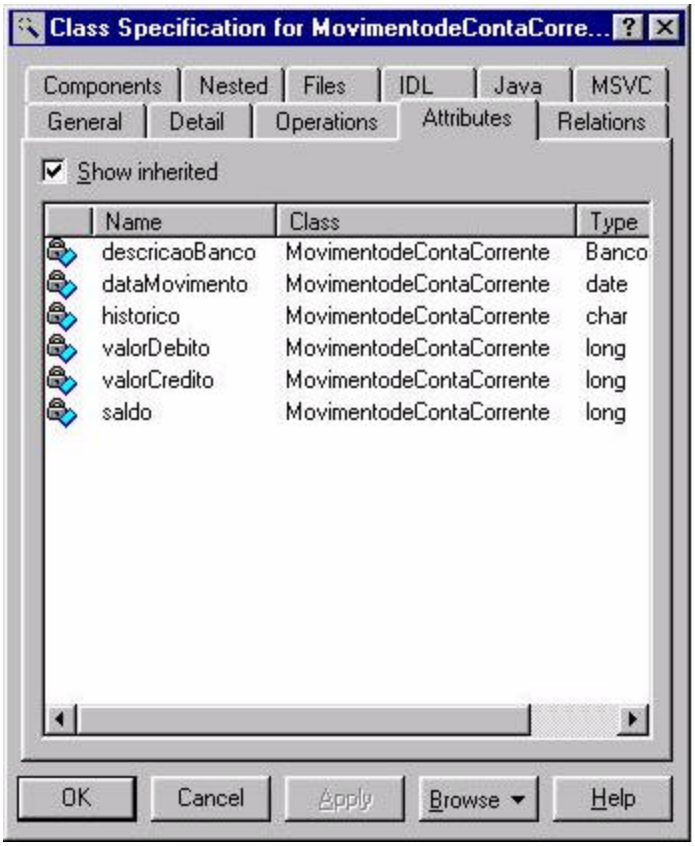

Class Specification for Local
Components
Nested
Files
IDL
Java
MSVC
General
Detail
Operations
Attributes
Relations
Show inherited
Signature
Class
Return type
cadastrarLocal
Local
char
consultarLocais
Local
Local
excluirLocal
Local
void
confirmarExclusao
Local
void
Local
Local
OK
Cancel
Apply
Browse
Help

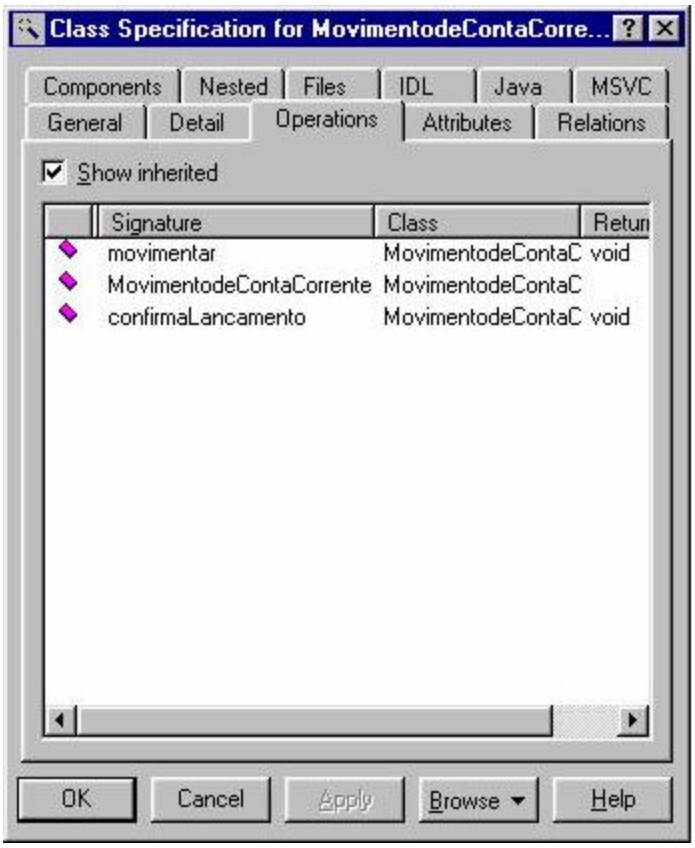
- **CLASSE: MATRICULADECLIENTE**



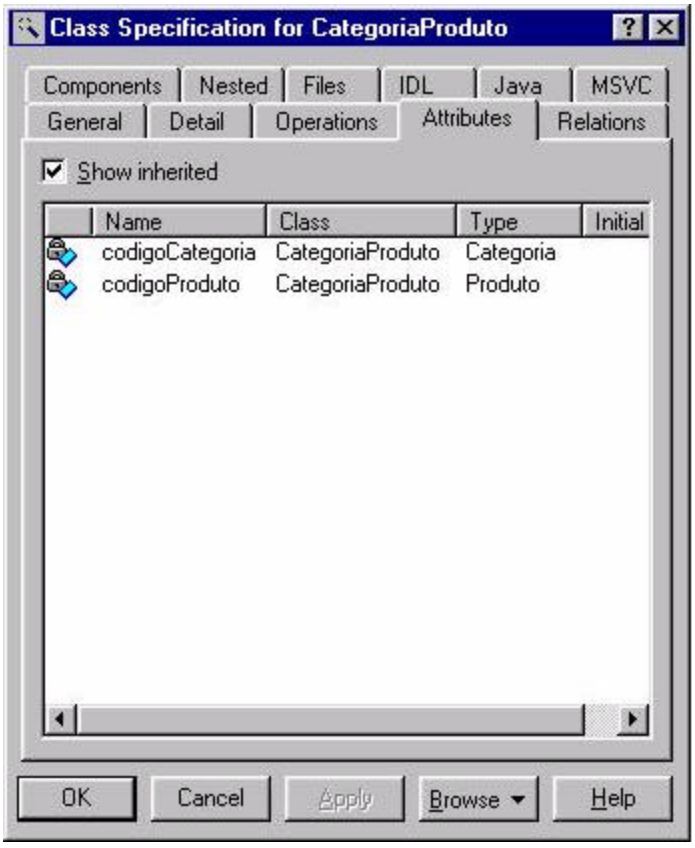

Class Specification for MatriculadeCliente
Components
Nested
Files
IDL
Java
MSVC
General
Detail
Operations
Attributes
Relations
Show inherited
Signature
Class
Return type
matricularCliente
MatriculadeCliente
Cliente
cancelarMatricula
MatriculadeCliente
void
MatriculadeCliente
MatriculadeCliente
OK
Cancel
Apply
Browse
Help

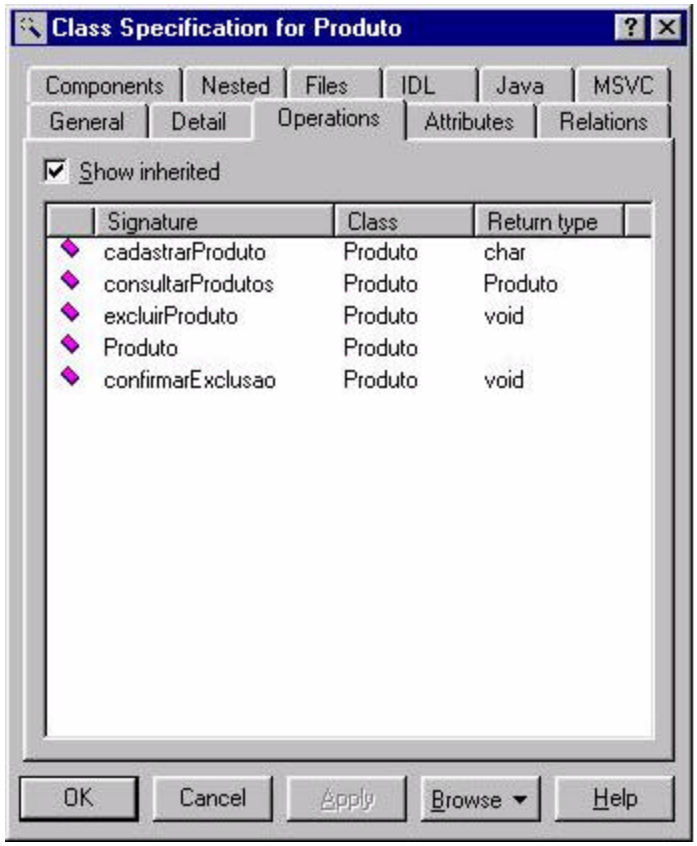
- **CLASSE: MOVIMENTODECONTACORRENTE**



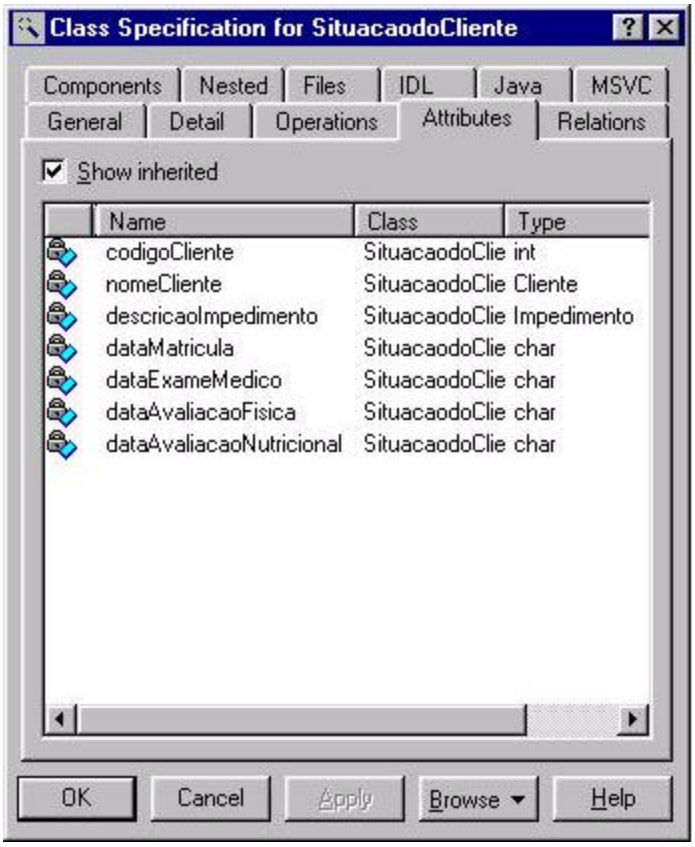

Class Specification for MovimentodeContaCorre...
Components
Nested
Files
IDL
Java
MSVC
General
Detail
Operations
Attributes
Relations
Show inherited
Signature
Class
Retur
movimentar
MovimentodeContaC
void
MovimentodeContaCorrente
MovimentodeContaC
confirmaLancamento
MovimentodeContaC
void
OK
Cancel
Apply
Browse
Help

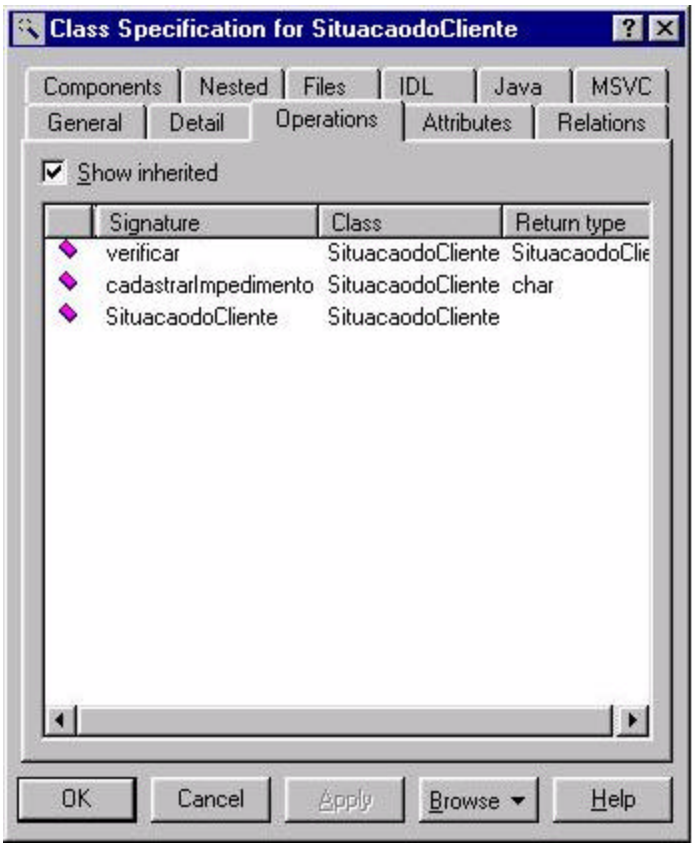
- **CLASSE: PRODUTO**



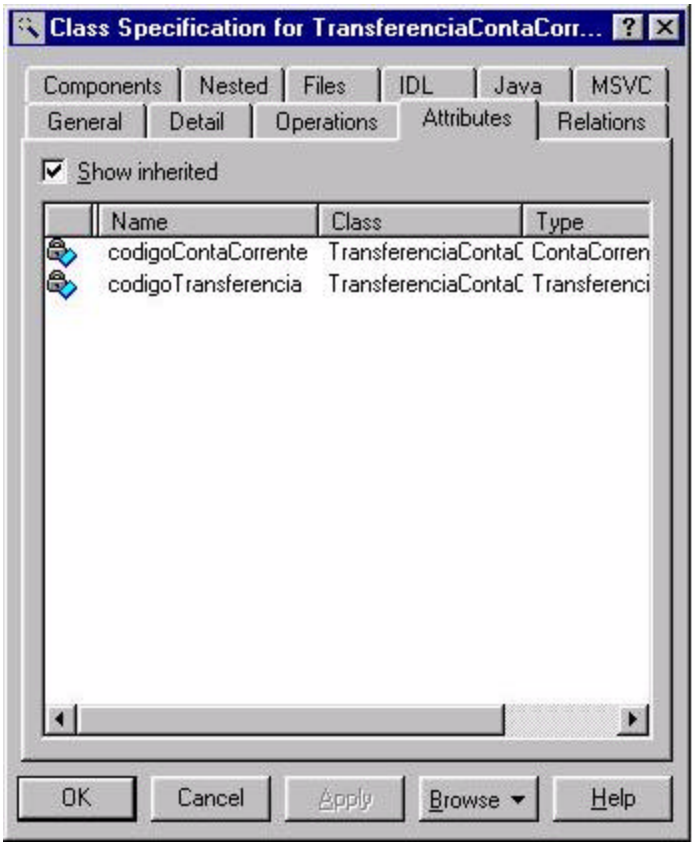

Class Specification for Produto
Components
Nested
Files
IDL
Java
MSVC
General
Detail
Operations
Attributes
Relations
Show inherited
Signature
Class
Return type
cadastrarProduto
Produto
char
consultarProdutos
Produto
Produto
excluirProduto
Produto
void
Produto
Produto
confirmarExclusao
Produto
void
OK
Cancel
Apply
Browse
Help

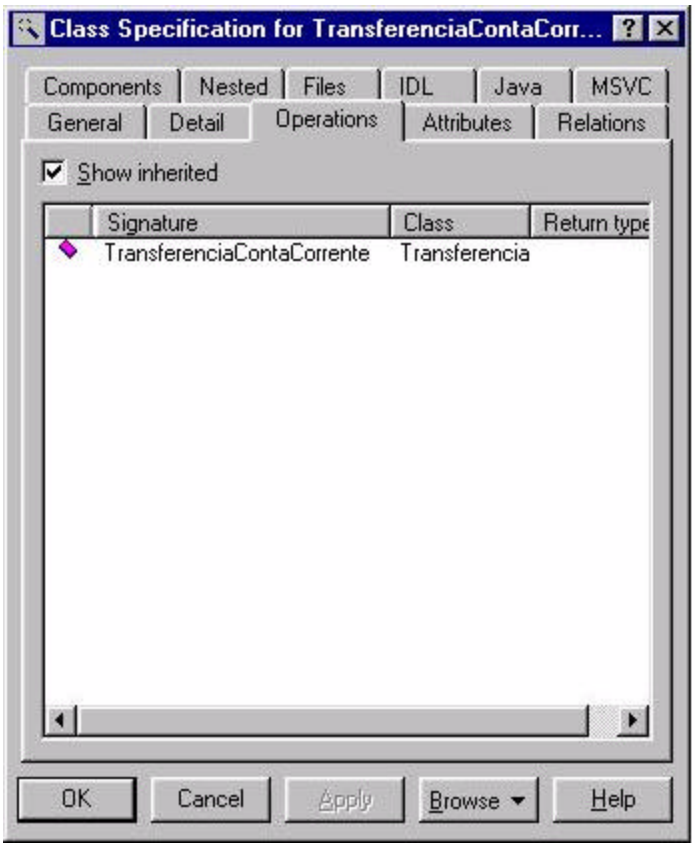
- **CLASSE: SITUACAODOCLIENTE**



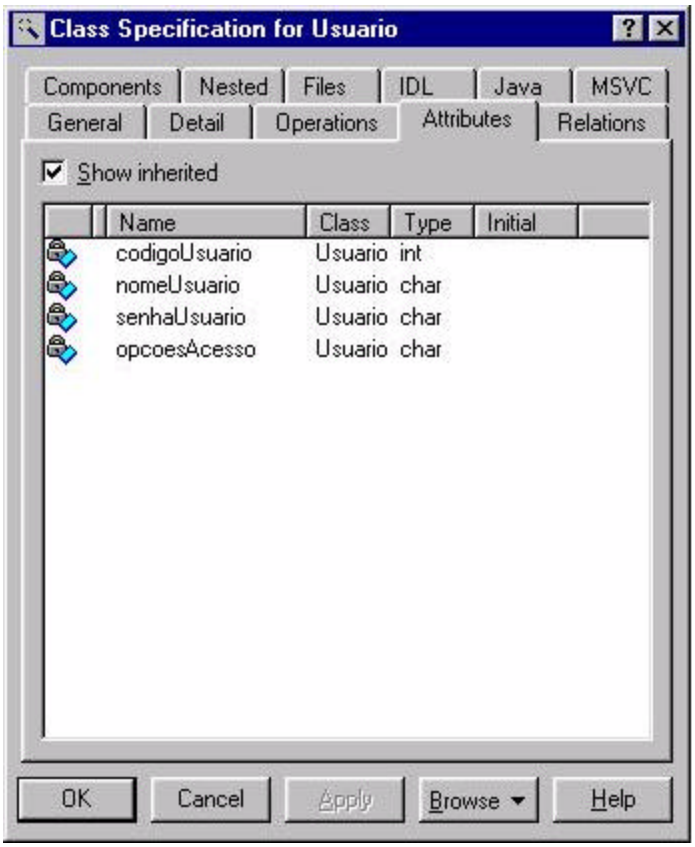

Class Specification for SituacaodoCliente
Components
Nested
Files
IDL
Java
MSVC
General
Detail
Operations
Attributes
Relations
Show inherited
Signature
Class
Return type
verificar
SituacaodoCliente
SituacaodoClie
cadastrarImpedimento
SituacaodoCliente
char
SituacaodoCliente
SituacaodoCliente
OK
Cancel
Apply
Browse
Help

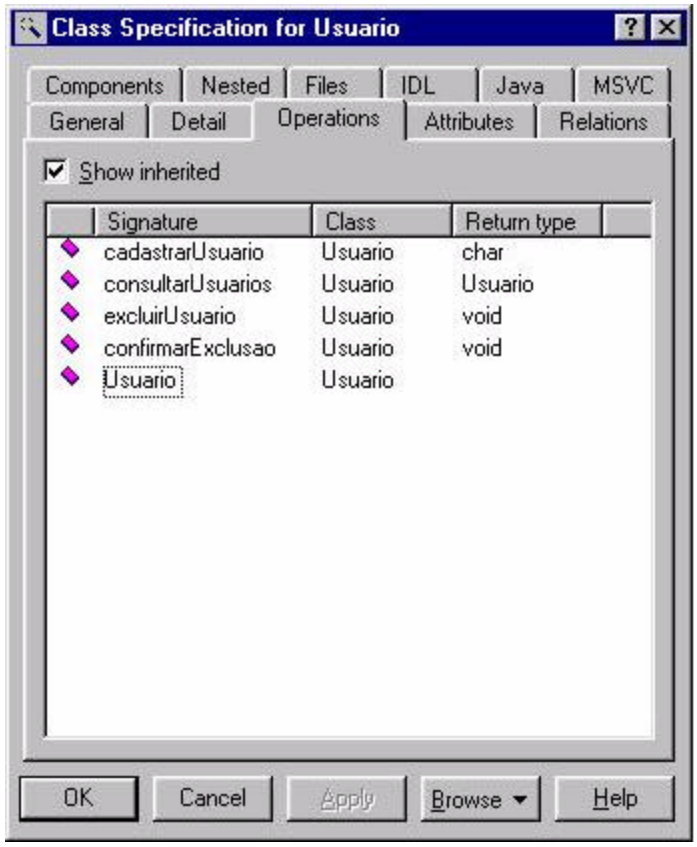
- **CLASSE: TRANSFERENCIACONTACORRENTE**



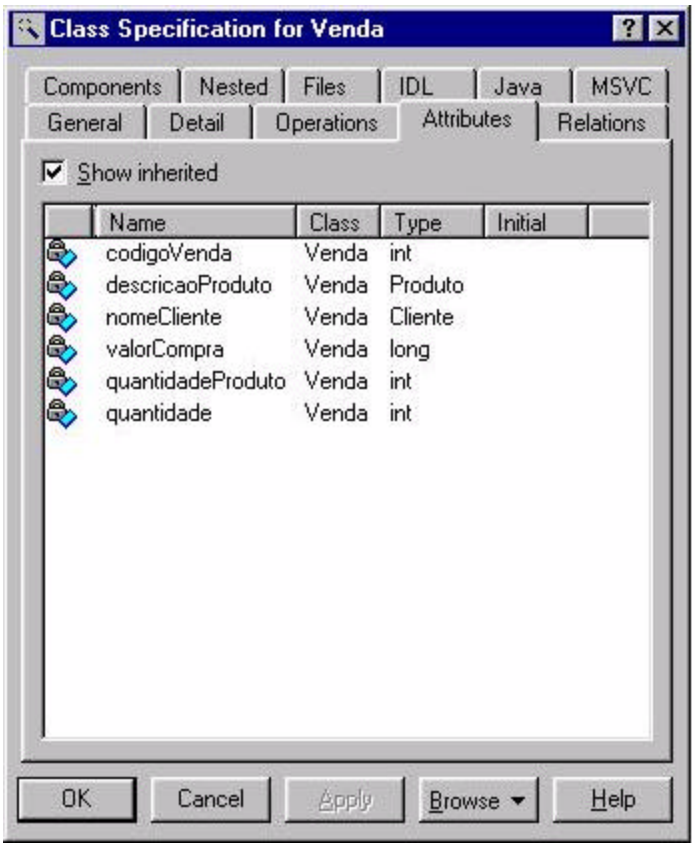

Class Specification for TransferenciaContaCorr...
Components
Nested
Files
IDL
Java
MSVC
General
Detail
Operations
Attributes
Relations
Show inherited
Signature
Class
Return type
TransferenciaContaCorrente
Transferencia
OK
Cancel
Apply
Browse
Help

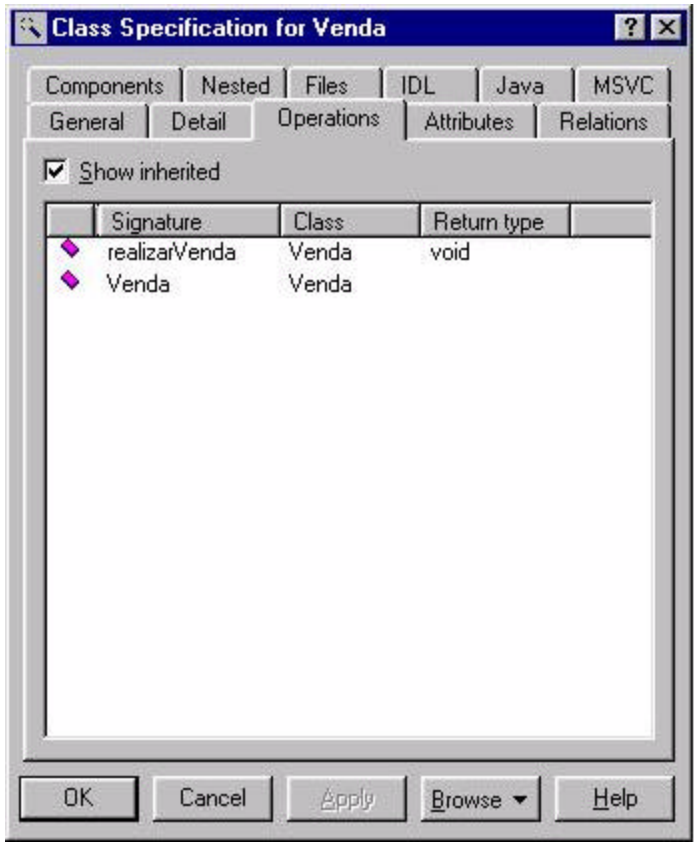
- **CLASSE: USUARIO**



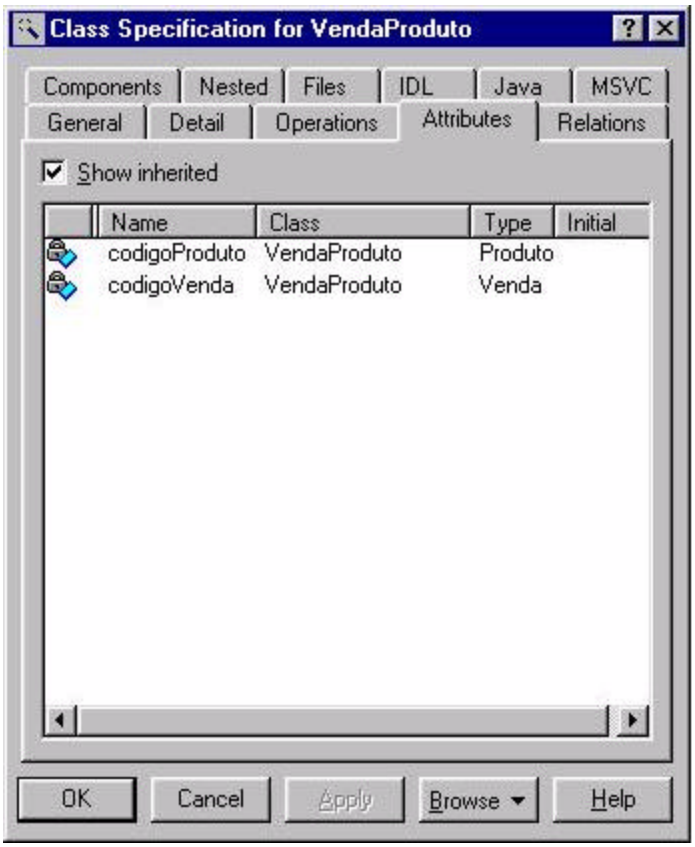

Class Specification for Usuario
Components
Nested
Files
IDL
Java
MSVC
General
Detail
Operations
Attributes
Relations
Show inherited
Signature
Class
Return type
cadastrarUsuario
Usuario
char
consultarUsuarios
Usuario
Usuario
excluirUsuario
Usuario
void
confirmarExclusao
Usuario
void
Usuario
Usuario
OK
Cancel
Apply
Browse
Help

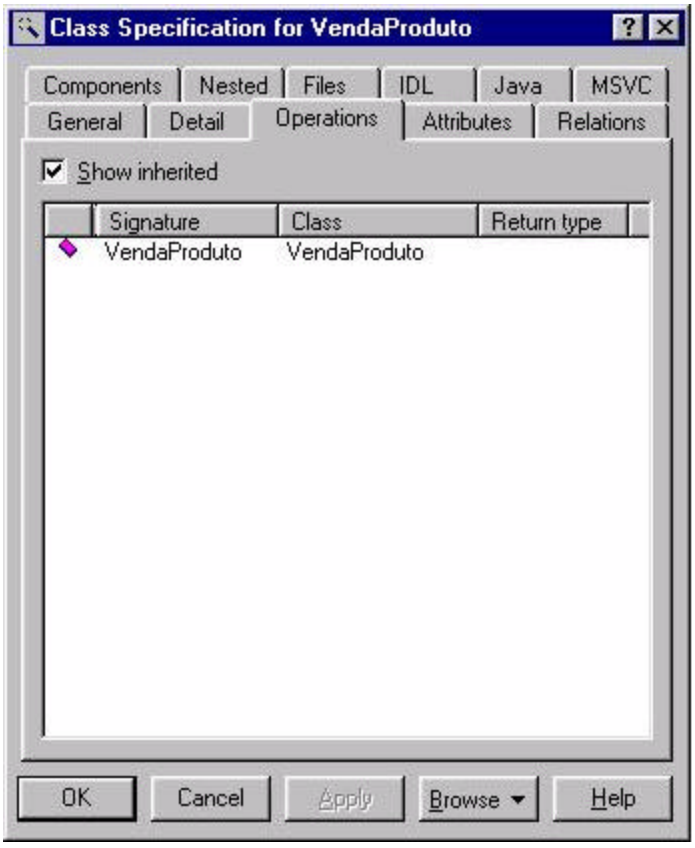
- **CLASSE: VENDA**



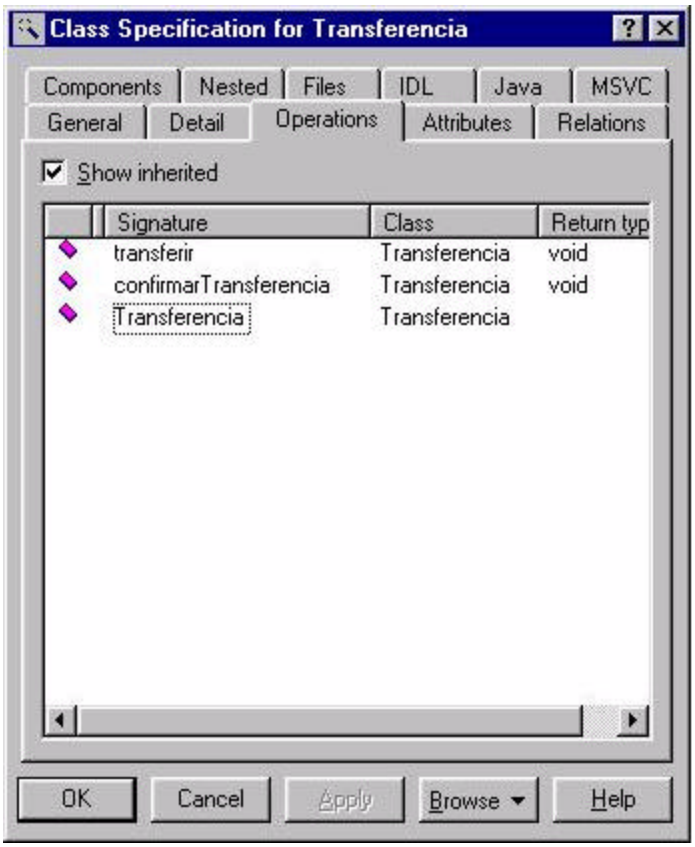

Class Specification for Venda
Components
Nested
Files
IDL
Java
MSVC
General
Detail
Operations
Attributes
Relations
Show inherited
Signature
Class
Return type
realizarVenda
Venda
void
Venda
Venda
OK
Cancel
Apply
Browse
Help

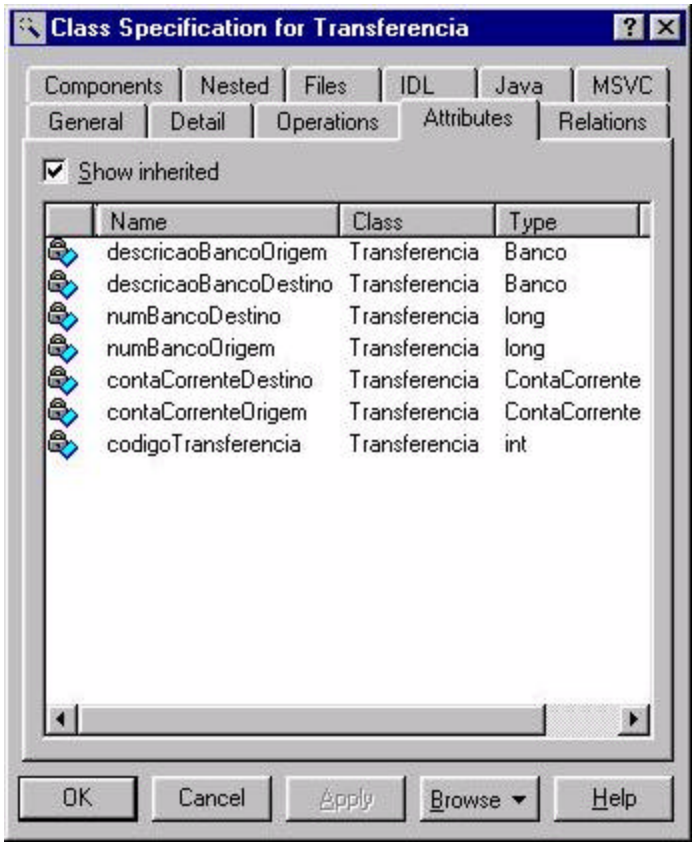
- **CLASSE: VENDAPRODUTO**




Class Specification for VendaProduto
Components
Nested
Files
IDL
Java
MSVC
General
Detail
Operations
Attributes
Relations
Show inherited
Signature
Class
Return type
VendaProduto
VendaProduto
OK
Cancel
Apply
Browse
Help

- **CLASSE: TRANSFERENCIA**




Class Specification for Transferencia
Components
Nested
Files
IDL
Java
MSVC
General
Detail
Operations
Attributes
Relations
Show inherited
Name
Class
Type
descricaoBancoOrigem Transferencia Banco
descricaoBancoDestino Transferencia Banco
numBancoDestino Transferencia long
numBancoOrigem Transferencia long
contaCorrenteDestino Transferencia ContaCorrente
contaCorrenteOrigem Transferencia ContaCorrente
codigoTransferencia Transferencia int
OK
Cancel
Apply
Browse
Help