UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

PROTÓTIPO DE UM SISTEMA TUTOR DE ORIENTAÇÃO A OBJETOS

Kim Willian Forest

BLUMENAU
2004

2004/2-30

Kim Willian Forest

PROTÓTIPO DE UM SISTEMA TUTOR DE ORIENTAÇÃO A OBJETOS

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Mauro Marcelo Mattos

BLUMENAU

2004

2004/2-30

PROTÓTIPO DE UM SISTEMA TUTOR DE ORIENTAÇÃO A OBJETOS

Por

KIM WILLIAN FOREST

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Mauro Marcelo Mattos, Dr. – Orientador, FURB
Membro:	Prof. Carlos E. Negrão Bizzotto, Dr. - FURB
Membro:	Prof. Jomi Fred Hubner, Dr. - FURB

Blumenau, dia 04 de dezembro de 2004

Dedico este trabalho a toda minha família que sempre esteve do meu lado durante a minha jornada.

Os bons livros fazem “sacar” para fora o que a pessoa tem de melhor dentro dela.

Lina Sotis Francesco Moratti

AGRADECIMENTOS

À Deus, pela sua benção nessa caminhada.

À minha família, que mesmo longe, sempre esteve presente.

Aos meus amigos, pelos conselhos e idéias.

RESUMO

O presente trabalho apresenta uma proposta de sistema tutor para auxílio a aprendizagem de orientação a objetos nas turmas introdutórias, a qual faz uma extensão ao trabalho que vem sendo desenvolvido como projeto de pesquisa na FURB e que diz respeito a um sistema tutor de algoritmos. O problema é contextualizado e é apresentada uma extensão ao projeto de pesquisa utilizando a linguagem de programação JAVA visando obter a ferramenta de auxílio, que além de abranger a aprendizagem de algoritmos, ofereça recursos que permitam ao aluno o aprendizado da orientação a objetos.

Palavras chaves: Informática na Educação; Sistemas Tutores; Orientação a Objetos.

ABSTRACT

This work describes a tool to aid students in introductory courses in learning object orientation concepts. The project is an extension to a research project being developed at FURB that aims to build a tutor to help students how to develop algorithms. The problem is contextualized and a software prototype developed in java is described.

Key-Words: Object-Orientation Learning Tool; Tutor; Learning Object Orientation.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

CAI - *Computer Aided Instruction*

OO – Orientação a Objetos

POO – Projeto Orientado a Objetos

STI – Sistema Tutor Inteligente

UML - *Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO

O mercado de software, cada vez mais, exige uma qualidade maior em seus produtos, que estão ficando cada vez maiores e mais complexos. Essa complexidade, e tamanho, são problemas significativos para técnicas tradicionais de programação. Montenegro e Pacheco (1994) ressaltam a utilização da programação orientada a objetos (POO) para controle de complexidade e manutenção dos sistemas. O programador que está totalmente familiarizado com as técnicas de programação estruturada encontra uma certa dificuldade para fazer a mudança da forma de programar.

A atividade de desenvolvimento de software vem utilizando cada vez mais as técnicas de orientação a objetos (OO). Segundo Deitel e Deitel (2003), “orientação a objetos é um modo natural de pensar sobre o mundo e de escrever programas de computador”. O projeto orientado a objeto (OOD) modela objetos para chegar a solução de um problema. A implementação vem por meio de uma linguagem OO (por exemplo, o JAVA).

Uma linguagem OO apresenta vantagens sobre uma linguagem estruturada (como maior facilidade de manutenção e reutilização de código), mas como aprender a utilizar essas vantagens, nem sempre é fácil entender, principalmente para quem está iniciando “os primeiros passos” em lógica de programação.

Segundo Silva I. (2000), o esforço de aprendizado dos conceitos de OO deve, necessariamente, estar associado ao uso de novas técnicas de ensinar a pensar, de forma a motivar as pessoas a avançar na assimilação desses novos conceitos e identificar as diferenças em relação aos métodos estruturados ao qual essas pessoas estão familiarizadas.

Mattos (2000) afirma que os estudantes que iniciam o curso de graduação em informática, normalmente encontram dificuldades relacionadas com a disciplina de introdução a programação (ou nome similar), cujo principal objetivo é o de introduzir os conceitos básicos de lógica de programação. Segundo o autor, “os alunos encontravam dificuldades em extrair as informações necessárias para iniciar a solução de problemas (...). O aluno geralmente não vem com uma boa noção do 2º grau, e não consegue visualizar certos problemas. Transcrevê-los então fica difícil. Sem essa capacidade aprimorada não terá facilidade para solucionar alguns problemas, e então, compreender a lógica de programação” (Mattos, 2000).

Na Universidade Regional de Blumenau (FURB) a disciplina introdutória de OO (denominada Programação II) aparece no 3º semestre do curso de Ciências da Computação. Até lá os alunos são ensinados a desenvolver programas usando técnicas estruturadas. Fowler e Scott (2000) colocam que para se conseguir utilizar as vantagens da OO, tem-se que fazer a "maldita troca de paradigma" (Fowler; Scott, 2000), ou seja, aprender a pensar OO.

Mattos (2000) fez um acompanhamento entre o primeiro semestre de 1996 e segundo semestre de 1998 sobre as turmas de introdução a programação (ou nome similar). Nesse acompanhamento, constatou que os alunos poderiam ser separados em dois grupos: aqueles que haviam entendido o "como fazer" e superado as dificuldades e aqueles que não conseguiam superá-las. Verificou-se também que os alunos do segundo grupo, na grande maioria, conseguiam descrever a solução intuitivamente (sem o formalismo necessário na computação), quando induzidos a pensar sobre o problema através de perguntas direcionadas.

No trabalho de Mattos (2000) foi construído um protótipo de um tutor de algoritmos que baseado em perguntas direcionadas, conduz o aluno no processo de desenvolvimento do algoritmo desejado.

Gubler (2002) fez uma extensão ao trabalho descrito em Mattos (2000), desenvolvendo um protótipo que, além de possuir uma simples interface com o usuário (que era primitiva e de difícil entendimento no trabalho anterior), suporta as estruturas de repetição em algoritmos. Nesse trabalho, Gubler também reformulou as perguntas aplicadas em relação às utilizadas no trabalho desenvolvido em Mattos (2000).

O protótipo desenvolvido em Gubler (2002), através da interface gráfica, apresenta perguntas ao usuário (aluno de disciplina de introdução a programação) e, de acordo com as respostas, constrói um algoritmo que atenda as necessidades desse aluno. O trabalho citado atende bem as necessidades de introdução a programação, mas não atende os requisitos de OO. Este foi o ponto de partida para o presente trabalho.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver um protótipo nos moldes do trabalho desenvolvido em Gubler (2002) que conduza o aluno no processo de desenvolvimento de uma simples aplicação OO.

Este trabalho terá como objetivos específicos:

a) o protótipo fará perguntas ao aluno, e de acordo com as respostas dadas por este, produzirá um código fonte OO que corresponderá as intenções do aluno;
b) o software deverá auxiliar o aluno no processo de aprendizagem de criação do diagrama de contexto, da produção de casos de uso, da analise de verbos e substantivos e na classificação dos verbos e substantivos para a geração de um diagrama de classes preliminar;
c) o protótipo deverá permitir a geração de código fonte em JAVA.

1.2 ESTRUTURA DO TRABALHO

O próximo capítulo apresenta os conceitos básicos de orientação a objetos.

O capítulo 3 introduz os conceitos de informática na educação, sistemas tutores e apresenta alguns trabalhos correlatos.

Os capítulos 4 e 5 apresentam a especificação e operacionalidade do protótipo construído e o capítulo 6 apresenta um estudo de caso utilizando a ferramenta.

Finalmente são apresentadas as conclusões e futuras extensões.

2 A ORIENTAÇÃO A OBJETOS

A origem da orientação a objetos, segundo Cox (1991), deve-se à necessidade de aumentar a produtividade no desenvolvimento de software. Na década de 70 surgiu a linguagem de programação Smalltalk, que foi conseqüência do amadurecimento dos princípios enunciados na década de 60 na linguagem Simula. Essas linguagem são consideradas as primeiras que trouxeram os princípios da orientação a objetos (OO). Mais tarde esses conceitos foram sendo aperfeiçoados e introduzidos a outras linguagens de programação.

Deitel e Deitel (2003) definem a OO como sendo “um modo natural de pensar sobre o mundo e de escrever programas de computador”. O projeto orientado a objeto (OOD) modela objetos para chegar a solução de um problema. A implementação vem por meio de uma linguagem OO (por exemplo, o JAVA).

Deitel e Deitel (2003) explicam que “os objetos estão no mundo real, você os vê, pessoas, animais, plantas são objetos, cada qual em sua categoria, cada qual com seus atributos e ações”. Os objetos possuem atributos e métodos (ou ações). No exemplo do objeto pássaro, atributos do pássaro seriam o peso, a cor, etc. Informações sobre ele. Os métodos seriam: voe, cante, etc. Ações que o objeto pássaro tem. Métodos são mensagens que um objeto usa para interagir com outro objeto.

Montenegro e Pacheco (1994) colocam que um programa estruturado tem como característica a existência de vários procedimentos (ou funções) que são chamados com a passagem de dados. Assim quando se chama novo procedimento os dados são “entregues” a esse procedimento. Já um programa orientado a objetos os dados e os “procedimentos” (agora chamados atributos e métodos, respectivamente) estão dentro de uma classe, ou seja, encapsulados em um só elemento (não mais tendo a presença de vários elementos – procedimentos).

Segundo Cox (1991), a análise orientada a objetos tem vantagens sobre a análise estruturada. Entre as mais citadas estão a maior facilidade de manutenção do software, o maior reaproveitamento de código e o menor código gerado por um programa OO em relação a um programa estruturado.

Cox (1991) resume em 2 palavras-chave a orientação a objetos: encapsulamento e herança. Encapsulamento, segundo ele, significa que o consumidor (aquele que está usando a classe) não mais faz operações sobre os dados, e sim escolhe o operador entre uma tabela de coisas que o objeto sabe fazer. Já a herança é definida por ele como uma forma de definir alguma construção útil em um local central, que possa ser utilizada em todos os locais que ela seja necessária.

2.1 CONCEITOS BÁSICOS

Na seqüência as definições de classe, objeto, atributos, métodos, acesso, herança e associação baseadas nas definições de Montenegro e Pacheco (1994). Os exemplos são desse trabalho.

2.1.1 Classe

Uma classe descreve um conjunto de objetos semelhantes. Na classe são encontrados os atributos e métodos que resumem as características comuns de vários objetos. A classe é a moldura para os objetos que poderão ser criados de acordo com ela.

2.1.2 Objeto ou instância

É quando se cria uma “variável” que é do tipo de uma classe. Como exemplo: o Totó é uma instância da classe “Cachorro”.

2.1.3 Atributos

Um atributo representa uma informação sobre uma determinada classe. É um dado, uma informação de estado. Por exemplo, o peso do (objeto) Totó, é um atributo dele, assim como a cor do pelo, etc. O atributo peso pertence a classe Cachorro e a instância Totó tem o seu peso.

2.1.4 Métodos

Um método refere-se a uma ação do objeto, uma mensagem que ele passa a outro(s) objeto(s). No exemplo, podemos citar que o (objeto) Totó late. A ação latir pertence a classe Cachorro, e está sendo executada na instância Totó.

A figura 1, na notação *Unified Modeling Language* (UML), que é um padrão de representação de Projeto Orientado a Objeto (POO), representa a classe Cachorro com seus atributos e métodos.

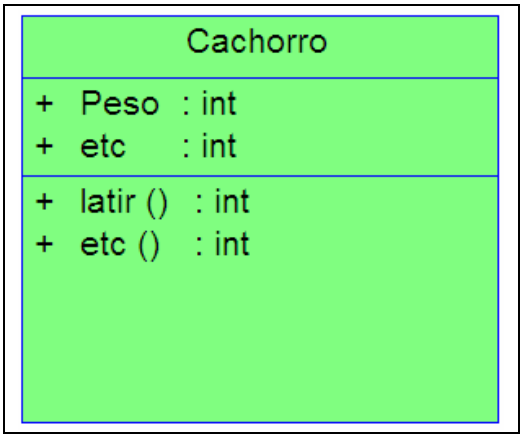


Figura 1: Exemplo de representação de uma classe utilizando a UML (Diagrama de classes)

2.1.5 Acesso

A classe ainda pode definir tipos de acesso a seus atributos e métodos. O acesso se refere as possibilidades de outras classes (ou objetos) poderem executar ou não seus métodos ou visualizar/alterar seus atributos. Os tipos de acessos podem ser:

a) **público:** o método ou atributo está disponível para todas as classes do sistema;
b) **protegido:** o método ou atributo somente poderá ser acessado pela própria classe ou sua hierarquia (ver herança, 2.1.6);
c) **privado:** somente a própria classe poderá acessar o atributo ou método.

Atributos e métodos também podem ser chamados de membros da classe.

2.1.6 Herança

Conforme Montenegro e Pacheco (1994), aqui está a grande oportunidade de reutilização de código. A herança permite que uma classe possa ser definida como sub-classe de uma classe pai (que poderá ser chamada de super-classe). Essa sub-classe herdará todos os métodos e atributos da classe pai, ou seja, esses métodos e atributos a partir de agora serão dela também (a não ser que o método seja privado – como visto em 2.1.5 Acesso, nesse caso, somente a própria classe poderá acessá-lo). A classe filha também poderá implementar os seus métodos, definindo então suas peculiaridades. Aqui o exemplo fica como a classe

Cachorro herda da classe Mamífero. Os métodos que pertencem a Mamífero não são necessários serem implementados novamente na classe Cachorro. A classe mamífero conterá, por exemplo, o atributo peso que não precisará ser declarado em Cachorro. A classe Pastor-Alemão, por sua vez, herda da classe Cachorro.

A figura 2, que é parte de um diagrama de classes da UML, mostra o exemplo clássico de herança.

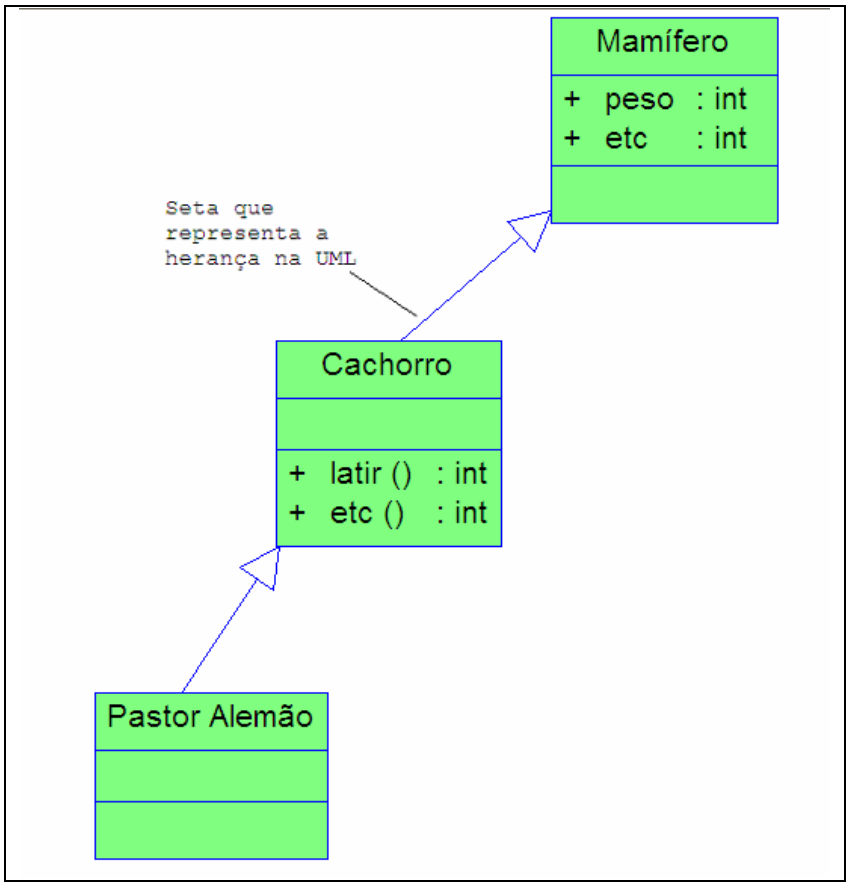


Figura 2: Exemplo de herança na representação de diagrama de classes da UML

2.1.7 Associação

Segundo Montenegro e Pacheco (1994), a associação é uma relação entre classes, portanto, é utilizada quando duas ou mais classes se relacionam. Como exemplo, a classe Cachorro possui um dono, que é da classe Pessoa. A instância da classe Cachorro: Totó, teria como dono a instância chamada Fulano, da classe Pessoa. Podemos dizer que o Cachorro poderá ter um dono (que é uma Pessoa). Não pode-se dizer que uma Pessoa obrigatoriamente tem um Cachorro, essa Pessoa pode não ter Cachorro, mas também pode até ter mais de um,

quantos quiser. Essa relação está representada na figura 3, utilizando o diagrama de classes na notação UML.

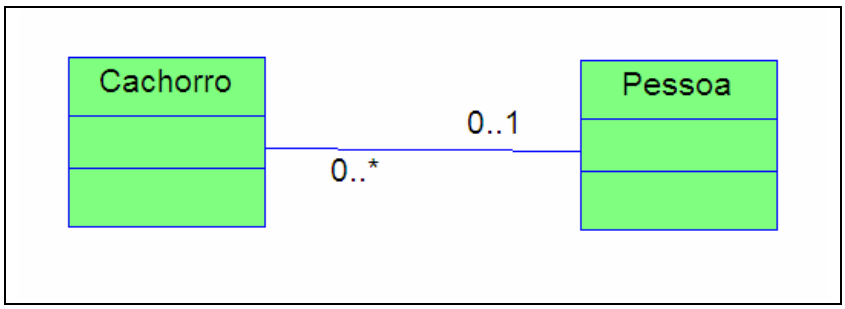


Figura 3: Exemplo de associação na notação UML (Diagrama de classes)

Uma relação de agregação se diferencia de uma relação de associação pelo fato do vínculo entre as classes ser mais forte, ou seja, uma classe depende de outra para existir. O exemplo clássico de agregação, onde a situação é bem clara, é o de NotaFiscal e ItemNotaFiscal. A classe NotaFiscal pode possuir vários itens de nota fiscal (classe ItemNotaFiscal). Um ItemNotaFiscal não poderá existir se não existir a NotaFiscal. Ao contrário do exemplo utilizado na associação de que um Cachorro pode existir sem uma Pessoa.

Para uma leitura mais abrangente sobre conceitos de OO recomenda-se a leitura de Cox (1991). Para se aprofundar mais na linguagem padrão de modelagem de objetos UML recomenda-se a leitura de Fowler e Scott (2000).

3 INFORMÁTICA NA EDUCAÇÃO

Para Coelho e Haguenauer (2004), o desenvolvimento de um sistema de tutoria adequado permite várias vantagens no ensino: a flexibilidade de horário, onde o aluno pode adiantar ou atrasar o estudo (é importante o acompanhamento do professor para que o aluno não deixe de estudar), o aumento do fluxo de informação, onde as informações estão a disposição do aluno, a comunicação mediada pelo ambiente virtual de aprendizagem, etc.

Segundo Educação Pública(2003), a idéia de ensino através de máquinas surgiu em 1924, usada por Dr. Sidney Pressey que inventou uma máquina para corrigir testes de múltipla escolha. O ensino através da informática tem suas raízes no ensino através de máquinas. Com a grande utilização de computadores pessoais esse método de ensino foi se espalhando pelo mundo.

Campos (1994) afirma que no Brasil, a primeira iniciativa na área de informática na educação aconteceu em 1981 em Brasília, no I Seminário de Informática na Educação que foi liderada pela SEI em esforço conjunto com o Ministério da Educação – MEC e o Conselho Nacional de Desenvolvimento Científico Tecnológico – CNPq. O I Seminário de Informática na Educação foi dirigido a pesquisadores em educação, psicologia, sociologia e informática, além de representantes da Sociedade dos Usuários de Computadores e Equipamentos Subsidiários – SUCESU.

Campos (1994) faz uma cronologia da informática na educação no Brasil (quadro 1).

1981: Realização do I Seminário de Informática na Educação. MEC/CNPq. Brasília – DF.
1981: Aprovação do documento MEC/SEI/CNPq/FINEP: Subsídios para implantação do Programa Informática na Educação.
1982: Realização do II Seminário de Informática na Educação. Salvador/BA.
1982: Criação do Centro de Informática do MEC – CENIFOR.
1983: Criação da Comissão Especial de Informática na Educação.
1983: Publicação do documento n-11 – Diretrizes para o estabelecimento de informática no setor educação, cultura e desportos.
1983: Publicação do Comunicado SEI solicitando às universidades apresentação de projetos para implantação de Centros-Pilotos do projeto EDUCOM.
1984: Criação do Centro de Informática Educativa – CENIFOR/FUNTEVE/MEC.
1984: Comunicado que informa quais universidades foram selecionadas para sediarem o Projeto EDUCOM.
1984: Formulação e assinatura de protocolos de intenções, passando ao CENIFOR/MEC a coordenação e supervisão do Projeto EDUCOM.
1984: Jornada sobre Informática na Educação – INEP.
1985: Implantação dos Centros-Pilotos do Projeto EDUCOM.
1985: Aprovação pelo CONIN do Plano Setorial de Educação e informática.
1985: Realização do Seminário de Informática e Educação do CFE/MEC.
1986: Criação do Comitê Assessor de Informática para a Educação de 1º e 2º graus (CAIE/MEC).
1986: Aprovação do programa de Ação Imediata em Informática na Educação de 1º e 2º graus (PAI).
1986: Implantação do 1º Concurso Nacional de Software Educacional Brasileiro.
1987: Implantação do Projeto FORMAR.
1987: Realização da Jornada de Trabalho de Informática na Educação: Subsídios para Políticas – Florianópolis/SC.
1988: Criação da PRONINFE – Programa Nacional de Informática na Educação (1988 – 1991).
1988: Documento A Informática na Educação – Uma Proposta do Conselho Federal de Educação.
1988: Apresentação do Projeto de Cooperação Internacional junto aos países latino-americanos. Projeto COEEBA.
1989: Realização da Jornada de Trabalho Luso-Latino-Americana de Informática na Educação.
1989: Apresentação do Projeto Multinacional de Informática aplicada à Educação Básica. OEA.
1990: Inclusão de metas e objetivos do PRONINFE no II Plano Nacional de Informática e Automação-II PLANIN.
1991: Aprovação do I Plano de Ação Integrada. PLANINFE.
1992: Criação de rubrica orçamentária específica no orçamento da União para atividades do setor de Informática na Educação.

Fonte: adaptado de Campos (1994).

Quadro 1: Cronologia da Informática na Educação.

Campos (1994) coloca que com o crescimento acelerado da informática no mercado mundial, muitos países precisam desenvolver seus próprios softwares educacionais. No Brasil ocorre o mesmo. Estão sendo desenvolvidos muitos softwares para os mais diversos conteúdos programáticos.

Cada vez mais se utiliza a tecnologia para transpor conceitos estudados para estudos de casos reais. A necessidade de estar cada vez mais especializado no mercado de trabalho segundo Galhardo e Zaina (2004) vem mudando e aumentando o uso das tecnologias da informação.

Galhardo e Zaina (2004) também salientam que a necessidade de receber estímulos durante o processo de aprendizagem fica claro nos alunos de um modo geral. Por isso e com objetivo de tornar as aulas mais motivadoras e interativas tem-se utilizado cada vez mais recursos tecnológicos para auxílio na aprendizagem.

Na área de informática ocorre um crescente processo de transformação. É preciso utilizar desses conhecimentos e técnicas novas, para se desenvolver novos métodos de auxílio na aprendizagem. Gubler (2002) coloca o desenvolvimento de novos softwares educativos como um exemplo da crescente onda da informática na educação. Ainda é salientado que o uso do computador na educação tem causado uma grande mudança no processo de ensino-aprendizagem.

Bizzotto (2003) salienta a importância do processo de aprendizagem para profissionais de novos tipos de empregos que existirão daqui a 10 anos. Esses novos empregos exigirão uma postura pró-ativa, com destaque a criatividade. Será necessário um processo de aprendizagem centrado nos aprendizes, de forma a desenvolver a autonomia dos mesmos. O autor ressalta a importância da criação de ambientes de aprendizagem para a participação ativa do aprendiz.

Para Campos (1994), três aspectos ficam evidenciados a respeito de informática na educação: a informática favorece o raciocínio lógico, favorece o raciocínio cognitivo e favorece a motivação.

Silva C. (2000) coloca que o computador pode ser utilizado como meio para auxiliar a construção do conhecimento pelo aluno. O aluno, através da interação com a máquina tem a oportunidade de construir o seu próprio conhecimento. O conhecimento não é passado a ele, e sim ele fica como construtor do seu próprio conhecimento. O autor ainda identifica essa situação como o “paradigma construcionista”, ou seja, enfatiza a construção e não a instrução.

Segunda Silva C. (2000) também existe o “paradigma instrucionista”, neste caso o computador é usado como ferramenta para ensinar, ou seja, são implementadas no computador informações que serão passadas ao aluno na forma de um tutorial.

Com a utilização cada vez maior das máquinas para atividades de ensino, o aluno tem a oportunidade de aprender através da interação com a máquina, e não somente através dos conceitos. O “paradigma construcionista” de Silva C. (2000) está sendo cada vez mais usado. Mais programas com esse fim são criados com o objetivo de facilitar o aprendizado, ou de passar a experiência e não mais somente as definições.

Bizzotto (2003) informa que, em sua grande maioria, os processos de aprendizagem são “centralizados” no professor. O conteúdo passado através de um software é entendido como um “produto completo e acabado”, não dando ao aluno a possibilidades de adaptação do material as suas necessidades.

Em função dessa “centralização” e “inflexibilidade” Bizzoto (2003) observa que “os ambientes de aprendizagem criados (softwares educacionais, páginas da internet, etc) possuem, em sua grande maioria” características como: o reforço do papel passivo do aluno na recepção do conteúdo, a falta de incentivo ao trabalho cooperativo, não evoluem em função das necessidades do aluno, do professor ou do grupo e “não permitem que o aluno seja autor de seu próprio trabalho” (Bizzotto, 2003).

O processo de aprendizagem “deve ser sempre aberto para a imprevisibilidade”, salienta Bizzotto (2003), dando suporte ao aluno no “trato com o devir”. Assim, o ambiente permite que o aluno tenha uma participação ativa no processo de aprendizagem, evoluindo em termos de criatividade, tornando-se uma pessoa mais pró-ativa.

Segundo Campos (1994) não se deseja um determinismo tecnológico. Procura-se utilizar a tecnologia para tornar os indivíduos situados no tempo e no espaço, onde seja ele o agente efetivo de transformações.

A tecnologia computacional permite um maior poder de interação com o usuário do que outros meios tecnológicos usualmente difundidos na educação como slides, filmes, etc. afirma Campos (1994), ressalta que é justamente o poder interativo do computador que reside a sua maior potencialidade em educação, Campos et al (1994).

Quando o aluno interage com o computador se torna um criador do seu próprio conhecimento. Bizzotto (2004) afirma que sobre o desenvolvimento de um ambiente de aprendizagem, é essencial que se possibilite que os alunos, “sejam autores do seu próprio aprendizado”. Assim os alunos não serão mais somente leitores passivos de um conteúdo previamente preparado, mas sim, passam a construir o seu próprio conhecimento.

3.1 CLASSIFICAÇÃO

A tutoria em geral, especialmente os cursos online, devem contribuir para a motivação e interesse do aluno, facilitando o processo de aprendizagem sem lhe diminuir a autonomia colocam Coelho e Haguenauer (2004).

Os sistemas tutores serão divididos aqui em 2 tópicos: os sistemas tutores e os sistemas tutores inteligentes. Será apresentada a definição de cada tipo e diferença. Ao fim será relacionado esse trabalho a esses conceitos.

3.1.1 Sistemas Tutores

Com relação ao objetivo de sistemas tutores, Campos (1994) coloca que os programas tutoriais “podem introduzir conceitos novos, apresentar habilidades, pretender a aquisição de conceitos, princípios e/ou gereralizações através da transmissão de determinado conteúdo ou da proposição de atividades que verifiquem a aquisição deste conteúdo”.

Campos (1994) coloca como algumas das principais características de um sistema tutor:

a) a intenção de chamar a atenção do aluno para que ele (o aluno) tenha interesse em usar esse sistema tutor;
b) a interação do mesmo com o aluno para que se possa chegar a um objetivo final;
c) uso de exemplos;
d) fazer questionamentos e receber respostas do aluno;
e) respostas analisadas pelo professor para que o aluno possa ser encaminhado a uma atividade que ele esteja necessitando para completar o aprendizado.

Campos (1994) também cita a importância de se ter uma avaliação do aluno para se determinar se os objetivos foram alcançados. Um sistema tutor, portanto poderá desempenhar um papel de se passar um problema e dar suporte para que esse problema seja resolvido, o

professor poderá mais tarde avaliar a solução alcançada pelo aluno. Um sistema tutor poderá fazer uma simulação de um exemplo para que o aluno tenha a idéia de como se resolver um determinado problema.

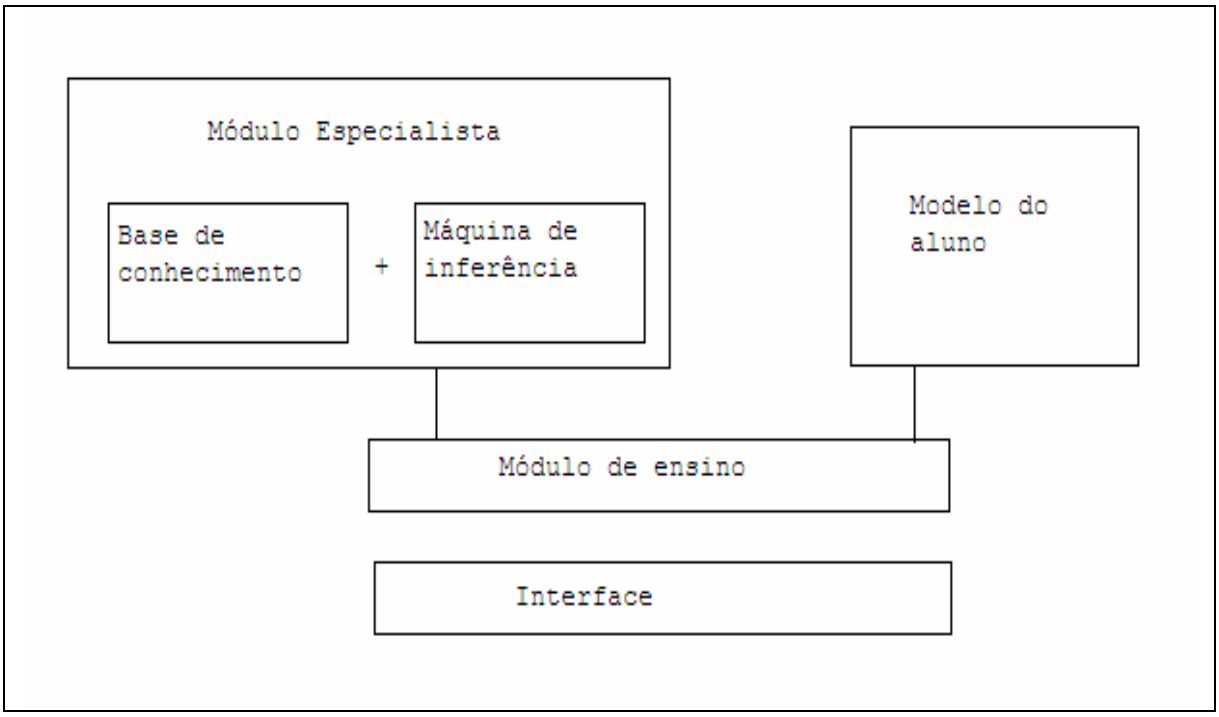
3.1.2 Sistemas Tutores Inteligentes

Segundo Campos (1994) os Sistemas Tutores Inteligentes (STI) tem como objetivo trazer maior flexibilidade e interatividade “no domínio da tutoria”. Campos et al (1994) define esses sistemas como sendo uma tentativa de integrar técnicas de inteligência artificial (IA) e uma teoria de aquisição de conhecimento dentro de um plano de ensino.

Os STIs são evoluções dos sistemas CAI (*Computer Aided Instruction*). A diferença é que um STI utiliza técnicas de inteligência artificial (IA), procurando proporcionar uma “experiência customizada de aprendizagem para o estudante simulando as interações professor-aluno” (Ricesu, 2003).

Campos (1994) ressalta que um STI deve ser muito exato, pois deverá oferecer a capacidade de resolução de problemas no domínio da aplicação. Também segundo o autor é importante o conhecimento do perfil do aluno que vai utilizar esse sistema, afim de ser sensível ao comportamento desse aluno.

A figura 4 representa a estrutura de um STI segundo Campos (1994 apud Fischetti e Gisolfi, 1990).



Fonte: Adaptado de Fischetti e Gisolfi (1990 apud Campos, 1994)

Figura 4: Arquitetura de um Sistema Tutor Inteligente

De acordo com a figura 4, Campos (1994) explica que o módulo especialista possui o conhecimento sobre o assunto a ser ensinado ao aluno. O modelo do aluno faz a gerencia do entendimento do aluno sobre esse conhecimento a ser ensinado. O módulo de ensino faz a explicação para se encontrar a estratégia de tutoria adequada para o ensino. A forma que esse conhecimento vai ser comunicado ao aluno é o trabalho da interface.

De acordo com Ricesu (2003 aput JONASSEN, 1993), um sistema tutor deve passar por 3 testes antes de ser considerado inteligente:

a) o conteúdo do tema ou especialidade pode ser acessado pelo sistema, o sistema pode fazer inferência e resolver problemas;
b) o sistema deve ter a capacidade de avaliação no que diz respeito a verificação se o conteúdo foi adquirido pelo aluno;
c) o sistema deve ser desenvolvido estrategicamente de forma a diminuir a discrepância entre o conhecimento do especialista e o conhecimento do aluno que está usando a ferramenta.

As principais características de um STI, segundo Campos (1994) são o conhecimento do domínio do problema, a possibilidade de reconhecimento do raciocínio do aluno, diálogo

com o aluno, possibilidade de análise do conhecimento do aluno sobre o problema, possibilidade de ajuda para a solução de um determinado problema, possibilidade de inferência do domínio do aluno sobre o tópico, etc.

O protótipo desenvolvido para esse trabalho não pode ser considerado um sistema tutor inteligente porque não utiliza técnicas de inteligência artificial. Embora o conhecimento da área do domínio (adquirido a partir de um especialista humano) tenha sido implementada em forma de uma árvore de decisões, não há efetivamente uma máquina de inferência operando sobre estas informações.

Assim sendo, este trabalho enquadra-se na categoria de sistema tutor uma vez que, como será apresentado no capítulo 5, a interação entre o aluno e o sistema conduz o aluno no processo de aprendizagem dos conceitos de OO.

3.2 TRABALHOS CORRELATOS

No trabalho de Mattos (2000) foi construído um protótipo de um sistema que formula perguntas ao aluno que de acordo com as respostas, conduz o aluno no processo de desenvolvimento do algoritmo desejado.

Gubler (2002) fez uma extensão ao trabalho desenvolvido em Mattos (2000), implementando um protótipo que, além de melhorar a interface com o usuário (que era primitiva e de difícil entendimento), suporta as estruturas de repetição em algoritmos. Nesse trabalho, Gubler também reformulou as perguntas aplicadas em relação as utilizadas no trabalho desenvolvido em Mattos (2000).

O protótipo desenvolvido em Gubler (2002) apresenta perguntas ao usuário (aluno de disciplina de introdução a programação - ou nome similar) e de acordo com as respostas, constrói um algoritmo que atenda as necessidades desse aluno. São utilizadas árvores de decisão nesse trabalho. Conforme a resposta dada pelo aluno é tomada uma direção na árvore. O trabalho citado atende bem as necessidades de introdução a programação (ou nome similar), mas não atende os requisitos de OO.

Além dos trabalhos de Mattos (2000) e Gubler (2002), tem-se os trabalho de Klotz (2002), que também implementou sistema tutor como ferramenta de auxílio. No trabalho de Klotz (2002) foi implementado um sistema de apoio à escrita de redações. O sistema

desenvolvido pelo autor faz uma interação com o aluno (através de perguntas), e auxilia o aluno a chegar ao seu objetivo (a redação), informando dicas para a sua redação, de acordo com a base de regras desenvolvida pelo autor.

Galhardo e Zaina (2004) construíram uma ferramenta que simula os conceitos de OO e que permite a interação do aluno durante o processo de aprendizagem. A ferramenta construída através de páginas HTML fornece conceitos de OO, e também proposta de exercícios, aos quais o aluno pode resolver nas páginas, criando classes, métodos e atributos. A ferramenta também gera um código fonte em JAVA referente a solução proposta pelo aluno que pode ser salvo em arquivo.

O trabalho de Galhardo e Zaina (2004) também tem em comum com o presente trabalho a idéia do professor criar um exercício que será mais tarde resolvido pelo aluno. A ferramenta criada permite que o professor cadastre e salve um enunciado de exercício que será resolvido pelo aluno utilizando a ferramenta.

4 DESENVOLVIMENTO DO TRABALHO

Conforme descrito anteriormente, o objetivo maior desse projeto é fazer uma extensão aos trabalhos de Mattos (2000) e Gubler (2002). Isso implica que o protótipo deve suportar a funcionalidade do protótipo de Gubler (2002), e, então, implementar a extensão da parte de orientação a objetos.

4.1 REQUISITOS PRINCIPAIS DO PROBLEMA E ESPECIFICAÇÃO

Como definido na proposta original, os requisitos funcionais do protótipo são:

a) implementar a metodologia de desenvolvimento de aplicações utilizada em Mattos (2003);
b) manter as características funcionais do protótipo desenvolvido no trabalho de Gubler (2002);
c) o protótipo deverá permitir a geração de código fonte JAVA para uma determinada área de aplicação.

Os requisitos não funcionais do sistema são:

a) implementar a ferramenta utilizando a linguagem de programação JAVA;
b) conduzir o aluno no processo de desenvolvimento de aplicação OO.

Em linhas gerais o sistema funciona em duas partes: há um módulo professor que permite o cadastramento do exercício a ser resolvido e há um módulo aluno o qual é utilizado pelo aluno para solucionar o problema proposto (figura 5).

No módulo do professor, o professor irá cadastrar um exercício, informando as classes que deverão existir com seus possíveis atributos e métodos.

Uma vez que o exercício tenha sido descrito, o aluno poderá abrir o exercício e realizá-lo. O sistema gera como produto final o código fonte em JAVA, o qual poderá ser avaliado pelo professor.

A figura 5 apresenta o diagrama de casos de uso (notação UML) do problema.

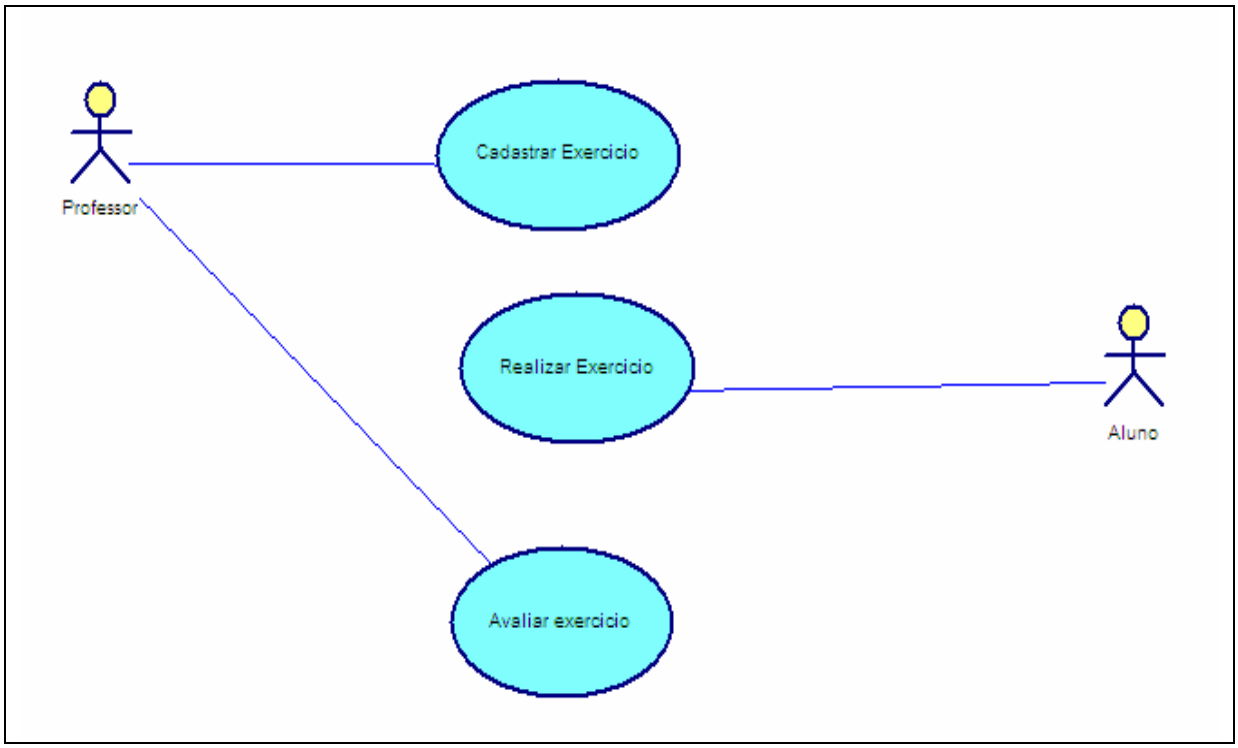


Figura 5: Casos de uso do problema

O diagrama de Casos de Uso e o diagrama de classes (que seguem) foram desenvolvidos na ferramenta *PowerDesigner* da *Sybase* versão 9.0.

A figura 6 apresenta o diagrama de classes que descreve a estrutura de um exercício. Um exercício pode ser composto por uma ou várias classes. Cada classe pode ter um certo número de atributos e métodos. Os métodos implementam um algoritmo o qual pode receber parâmetros. O algoritmo também pode ter variáveis locais.

Tanto os atributos como os parâmetros e variáveis do algoritmo são descritos como um dado o qual possui um nome e um tipo. Um tipo de dado pode ser: String, int, boolean, float, Date. Já o nome é o nome a ser dado para o tipo de dado.

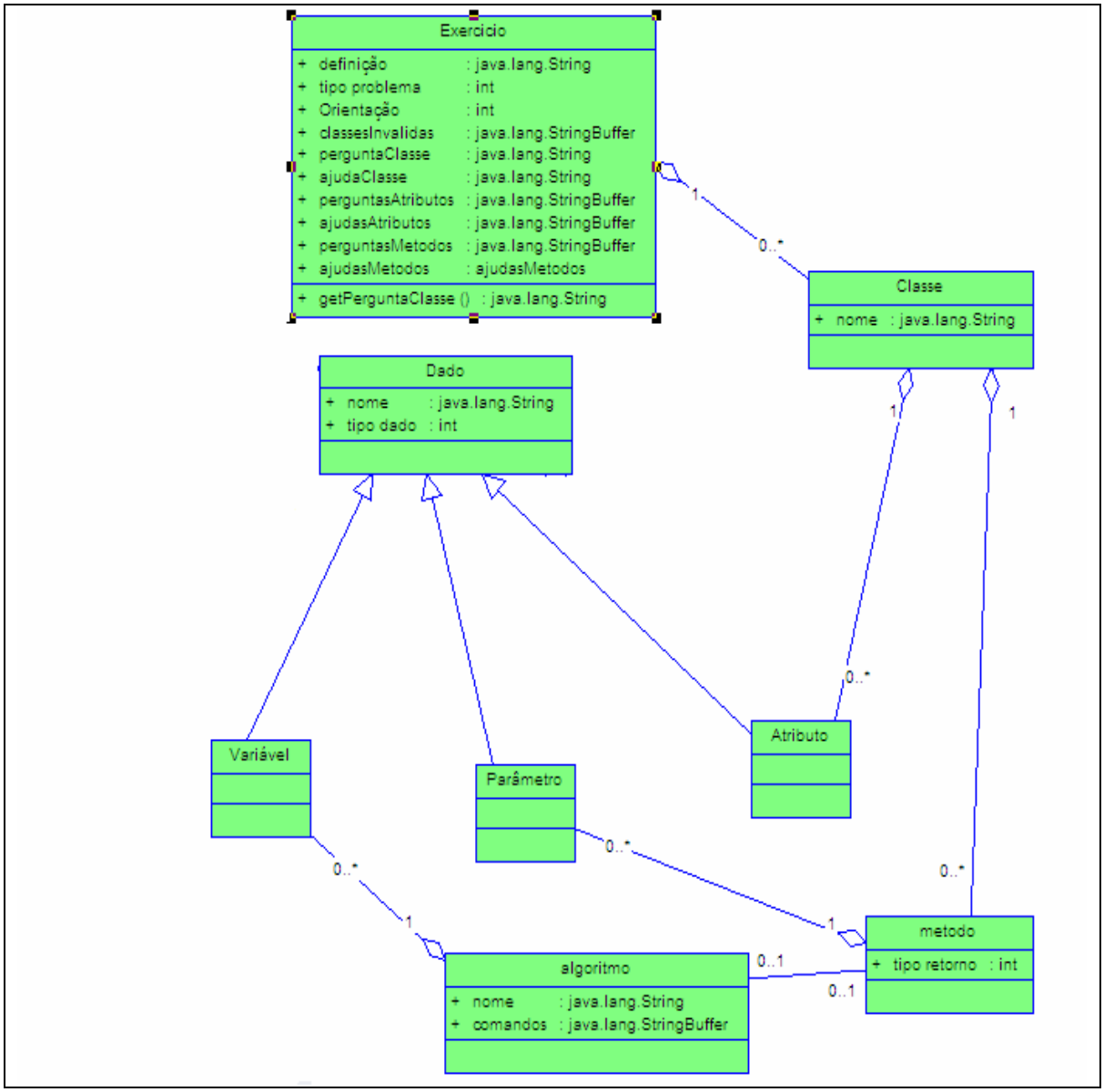


Figura 6: Parte do diagrama de classes

Conforme diagrama, no exercício serão cadastradas classes, com seus respectivos atributos e métodos. Também na classe **Exercício** (figura 6) foi implementado um conjunto de classes inválidas (nomes inválidos de classes, presente no atributo **classesInvalidas**) que tem como objetivo fazer com que o aluno identifique a classe entre um conjunto de classes (que somente terá uma válida – ficará mais claro na apresentação da tela do aluno). E também as perguntas e dicas referente a classes, atributos e métodos que serão identificados pelo aluno. A pergunta e dica de classe é uma só para o problema. As perguntas e dicas de atributos e métodos serão feitas da forma de uma para cada classe (também ficará mais claro no decorrer desse trabalho – ver 7.2 Protótipo do Professor). No diagrama são as perguntas

são os atributos: **perguntaClasse, ajudaClasse, perguntasAtributos, ajudasAtributos, perguntasMetodos** e **ajudas Métodos.** No diagrama de classes da figura 6, as classes **Parâmetro**, **Atributo** e **Variável** herdam da classe **Dado** por motivos de implementação da geração de código fonte, devido diferença nas estruturas desses tipos em termos de código fonte.

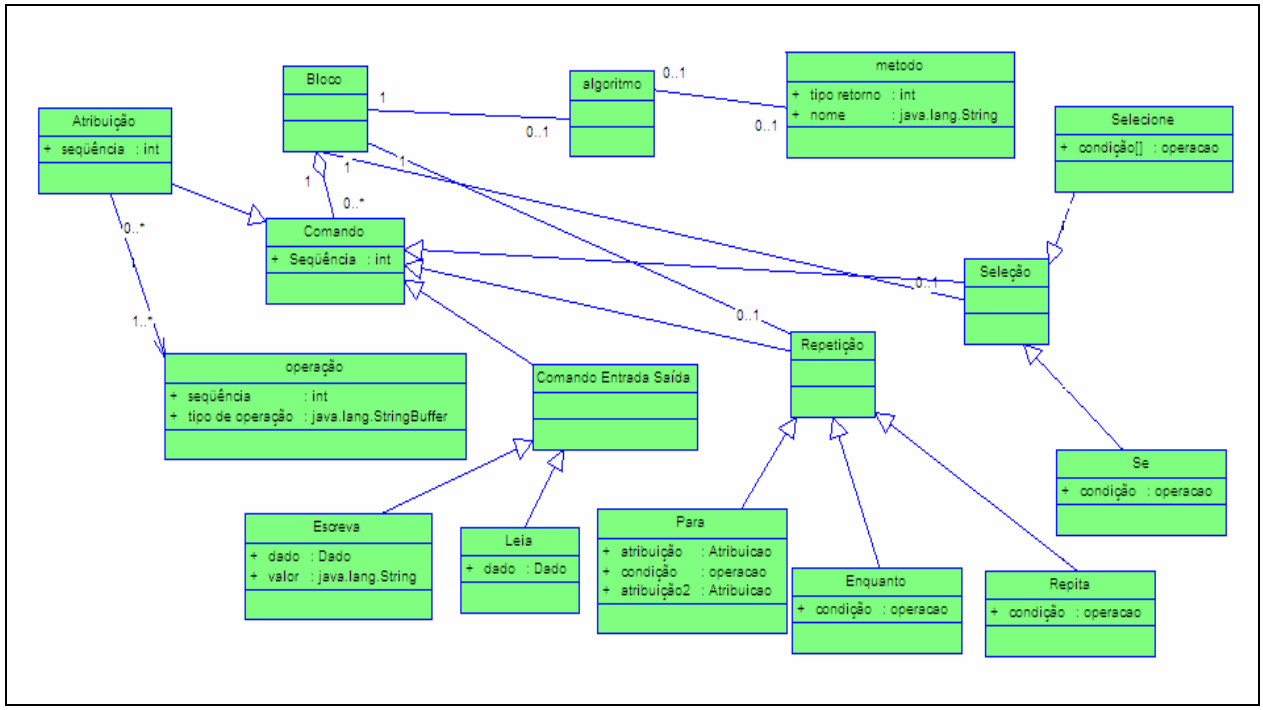


Figura 7: Parte do diagrama de classes

Conforme apresentado na figura 7, um método é implementado na forma de um algoritmo. O **Algoritmo** possui um **Bloco**. Esse Bloco pode conter vários **Comandos**. Os Comandos implementam as estruturas lógicas de um algoritmo, como o comando **Se**, o **Enquanto**, uma **Atribuição**, etc. Há certos comandos que possuem um Bloco dentro deles, como os comandos de **Repetição** e de **Seleção**.

Cabe destacar que, um algoritmo pode ser descrito em termos de estruturas lógicas, ou seja, não há uma amarração à linguagem Java. Futuras extensões ao trabalho poderão permitir a geração de código fonte em outras linguagens além do Java simplesmente implementando as estruturas que descrevem a sintaxe da linguagem. (Figura 7).

A seguir são descritos em maior nível de detalhe os módulos professor e aluno.

4.2 MÓDULO DO PROFESSOR

O módulo do professor foi desenvolvido com o objetivo de permitir o cadastramento de um exercício que mais tarde ser resolvido pelo aluno. Esse exercício, além da definição deverá conter as classes (com seus respectivos atributos e métodos) e as perguntas principais.

O diagrama de atividades da UML na figura 8 apresenta o funcionamento geral do protótipo do professor. A descrição do diagrama segue a seguir.

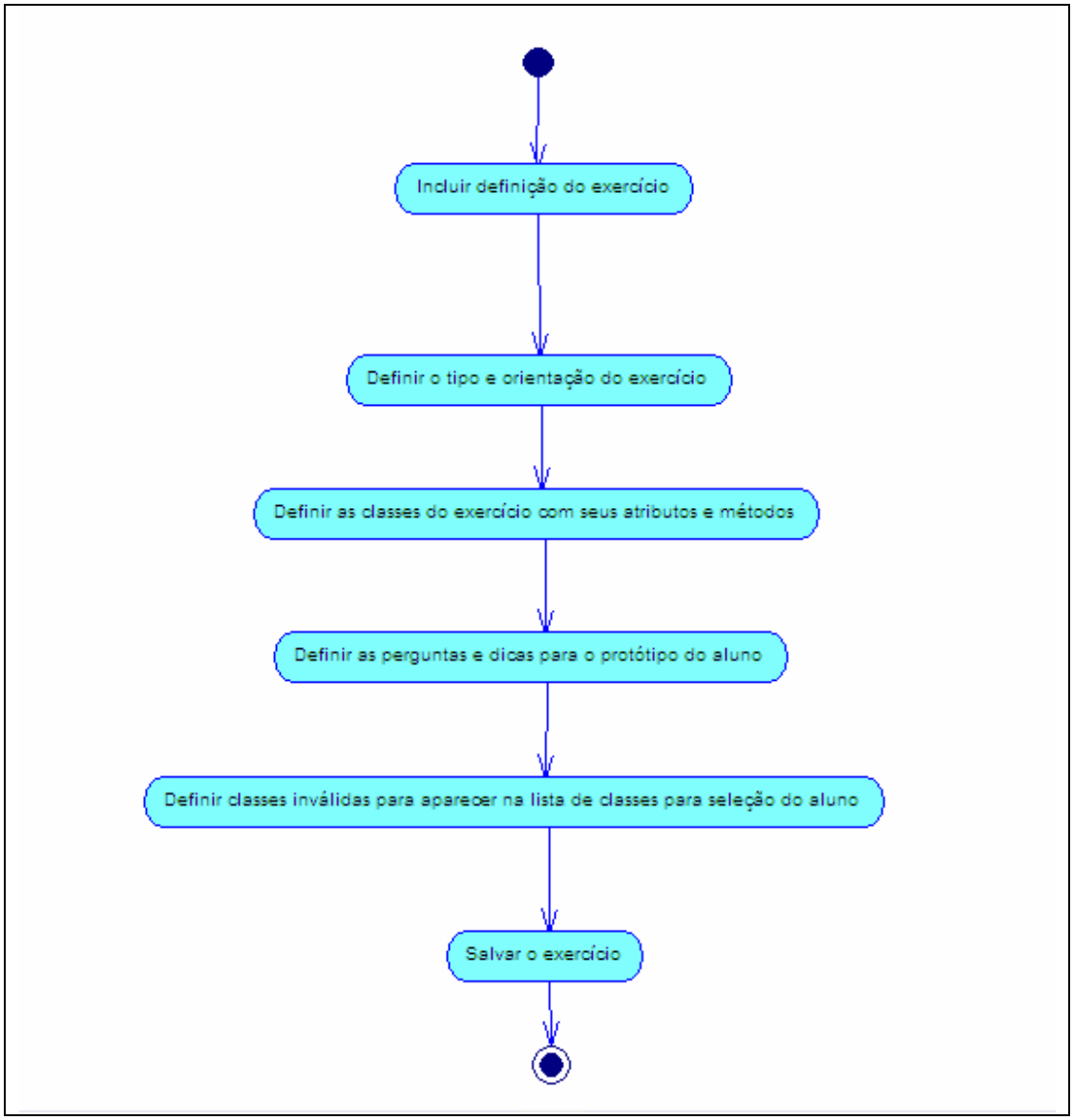


Figura 8: diagrama de atividades do módulo do professor

Inicialmente o professor receberá uma tela onde deverá ser registrada a definição do problema a ser resolvido. Depois virá a tela que permite identificar o tipo e orientação do exercício (essas telas serão descritas no decorrer do texto). O tipo identifica se o exercício é de OO, estruturado ou possui características tanto de OO como estruturadas e a orientação identificará para quais alunos se recomenda a realização do exercício, se somente para leigos em OO, ou leigos em programação estruturada ou para ambos.

O professor deverá registrar as classes com seus atributos e métodos que o aluno deverá criar na realização de um exercício (essa tela será descrita no decorrer do texto). Na seqüência, serão definidas as perguntas e dicas que deverão ser apresentadas ao aluno durante o processo de solução do problema. O protótipo do aluno exibirá as perguntas cadastradas no módulo professor, permitindo ao professor construir exercícios com dicas personalizadas ao contexto do exercício.

Para finalizar o processo, o professor vai definir um conjunto de nomes de classes inválidas. O objetivo deste sub-conjunto de classes é introduzir o aluno no processo de identificação das classes do domínio do problema.

No protótipo do aluno, a cada momento de criação de uma nova classe, será exibida uma lista de nomes de classes. O aluno deverá identificar qual das classes daquela relação deverá ser desenvolvida para a solução do problema, dentre um conjunto de classes apresentadas existirá uma classe que é a correta naquele momento.

A figura 9 mostra a tela do professor, onde o professor cadastra um exercício que será resolvido pelo aluno.

No final do processo, o professor deve salvar o exercício para que o(s) aluno(s) possa(m) resolver. O arquivo salvo contém um objeto do tipo **ControladorExercicio.** A classe **ControladorExercicio** faz uma ligação entre a interface e a classe exercício. Essa classe contém as informações necessárias para exibição na tela (interface com usuário) de um exercício criado pelo professor.

4.2.1 Detalhamento da interface do módulo professor

Nesta seção é apresentado um detalhamento da interface do módulo professor. A figura 9 mostra a tela do professor, onde o professor cadastra um exercício que será resolvido pelo aluno.

Nessa tela (Figura 9), o professor irá definir o exercício. O menu (número 1 na Figura 9) tem as funcionalidades básicas: Novo, Abrir, Salvar, Salvar Como e Sair.

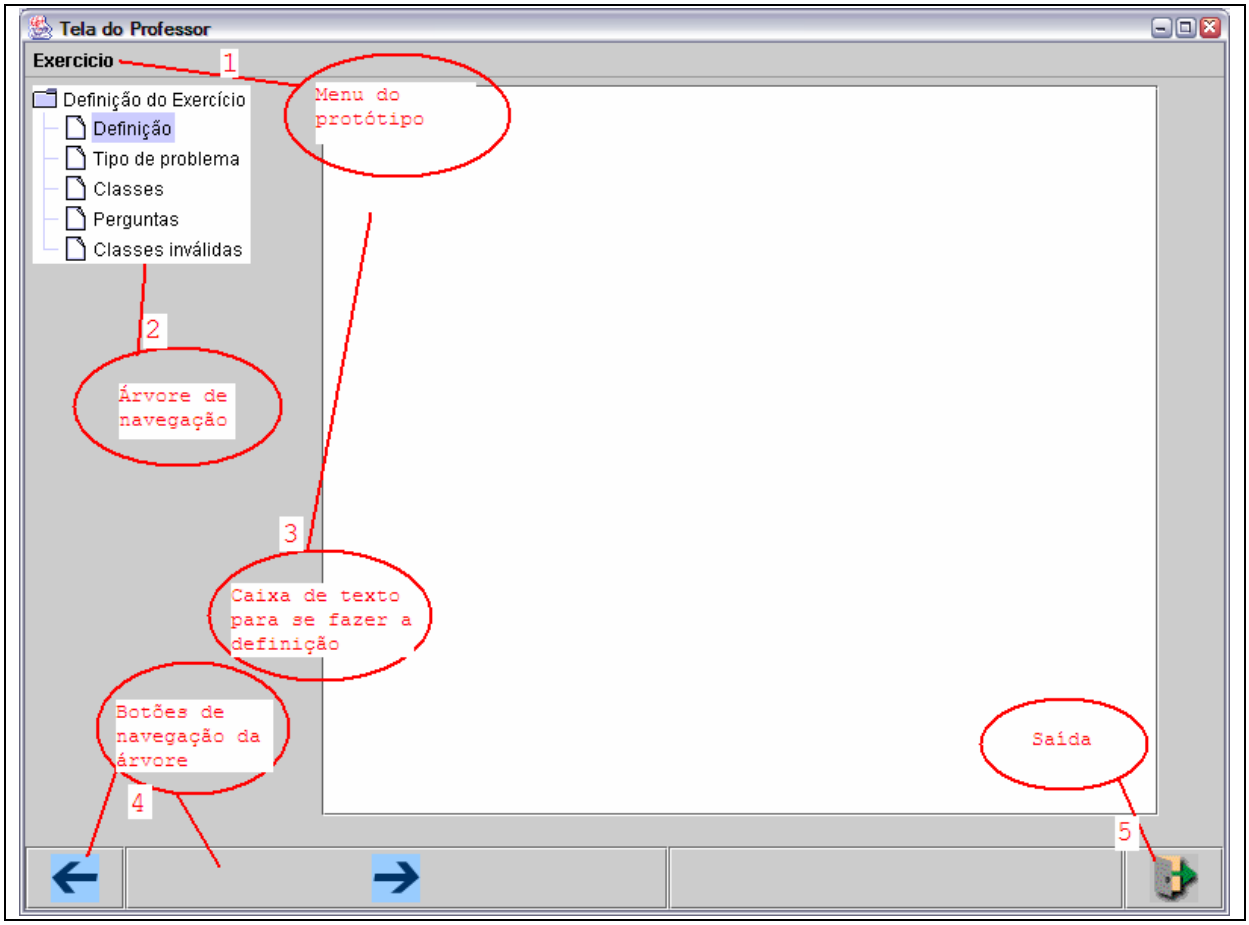


Figura 9: Tela inicial do protótipo do professor

A árvore de navegação (número 2 na figura 9), que é uma JTree do Swing, mostra inicialmente a tela de definição, onde o professor vai inserir o texto com a definição do problema (número 3 na figura 9). No item 4 estão os botões de navegação na árvore (item 3), onde se pode alterar a tela que se está editando. O botão de saída é o item 5.

A figura 10 mostra o segundo painel, referente ao tipo de problema da árvore de navegação.

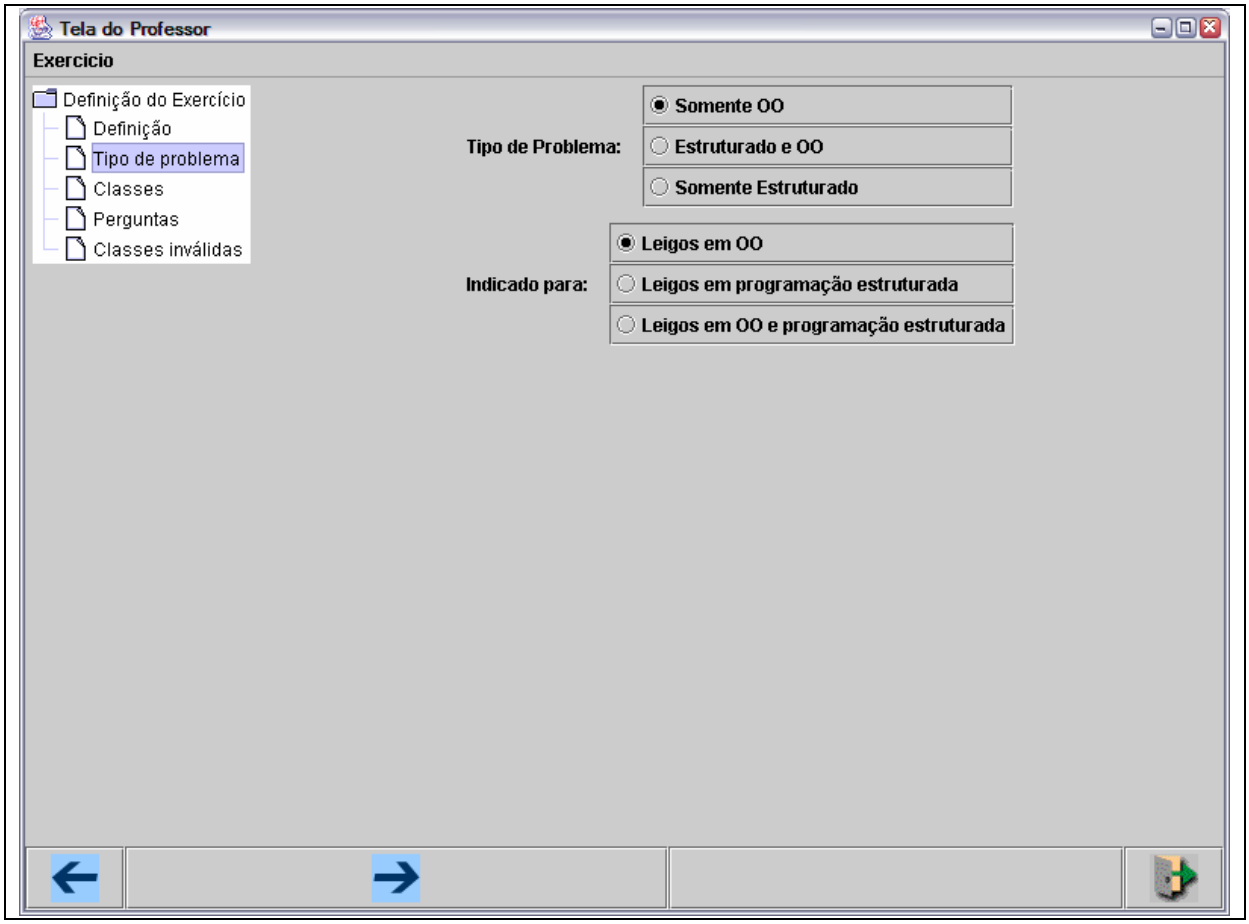


Figura 10: Tela do professor – definição do tipo de problema

A tela de tipo de problema permite a especificação de que tipo de problema o professor está cadastrando e qual a indicação desse problema. Essa funcionalidade não foi implementada nessa versão do sistema. Contudo foi mantida como possibilidade de futuras extensões ao projeto. A idéia é que o professor possa configurar um problema de OO ou um problema somente de Algoritmo (solução estruturada) e/ou com mais níveis de dificuldade.

A figura 11 apresenta a terceira tela, que é a tela de identificação das classes do problema:

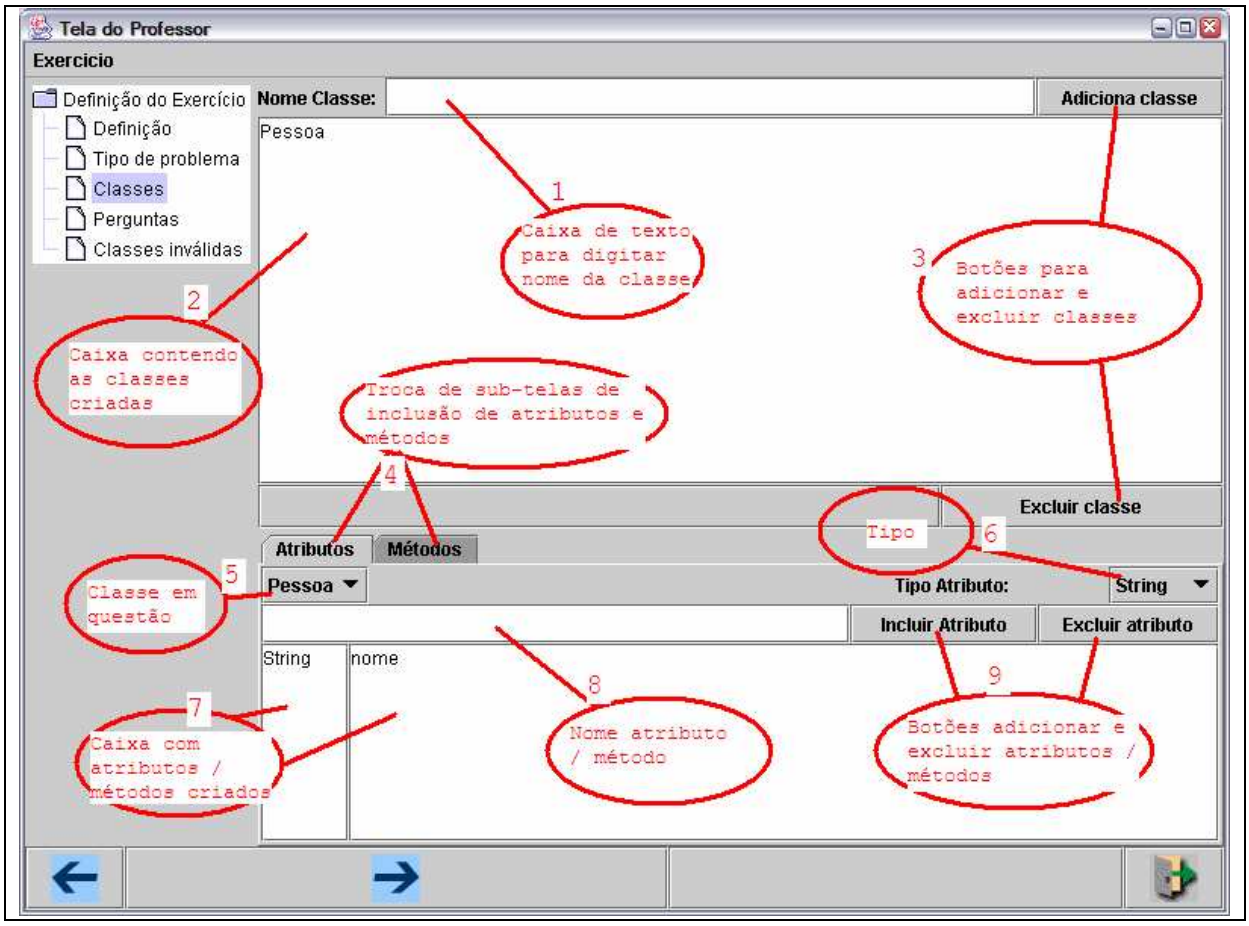


Figura 11: Tela do professor – Identificação das classes

Na tela de identificação de classes, o professor vai identificar as classes envolvidas no problema que o aluno deverá criar, com seus respectivos atributos e métodos.

Nessa tela, na parte superior, além dos botões de inclusão e exclusão de classes (item 3 na figura 11) e da caixa de texto para digitar o nome da classe (item 1, figura 11) existe um componente que contém as classes criadas pelo professor até o momento (item 2, figura 11). Esse componente (item 2, figura 11) simula um TextGrid do Delphi. Como o Swing não possui um elemento com as características de um Grid, foi implementado a classe JTextGrid que herda de JTextArea do Swing. Esse componente contém métodos para tratar a String como linhas, simulando assim a característica do Grid. Mais detalhes sobre esse componente criado no item 4.4 (Implementação) abaixo.

Na parte superior da tela existe um painel em forma de Tab (item 4, figura 11), isto é, de acordo com a Tab selecionada um painel é exibido (item 4 na figura 11). Há dois painéis: o

primeiro contém informações sobre os atributos que o professor está cadastrando para as classes e o segundo informações sobre os métodos das classes. O JTextGrid, componente criado, também é utilizado, para as caixas de texto com os atributos/tipos e métodos/tipos retornos (ver item 7 na figura 11).

O combo identificado como “classe em questão” (item 5 da figura 11) relaciona qual classe o professor está criando os atributos e métodos. Conterá todas as classes criadas.

O item 8 na figura 11 é a caixa de texto para se digitar o nome do atributo ou método. No item 6 (figura 11) identifica-se o tipo do atributo ou o tipo de retorno para um método. O item 9 da figura 11 representa os botões para inclusão e exclusão de atributos ou métodos. O que identifica se está criando atributo ou método é o painel (item 4, figura 11).

A figura 12 representa a tela de definição das perguntas pelo professor, é o “galho Perguntas” da árvore de navegação.

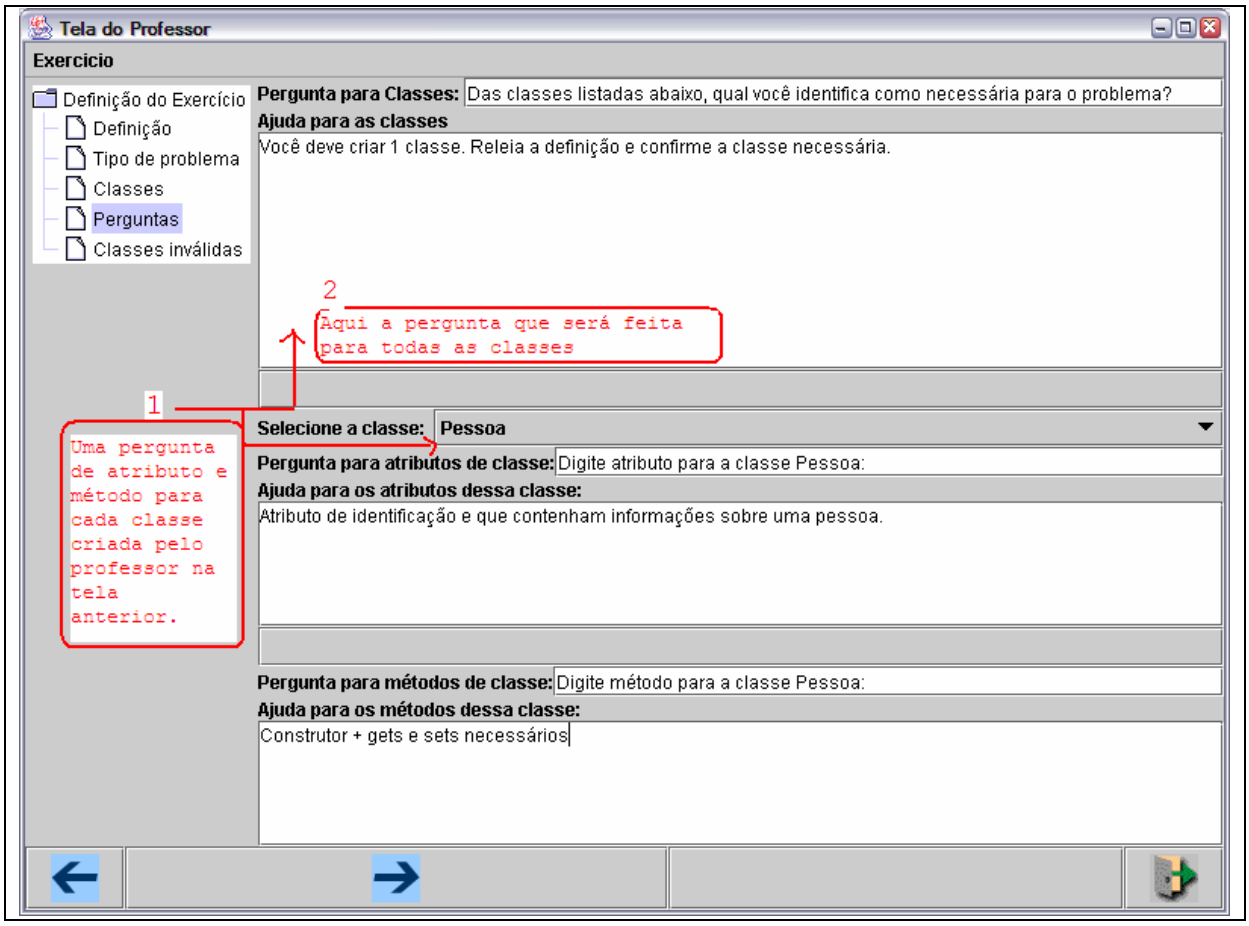


Figura 12: Tela de definição das perguntas

O professor poderá cadastrar perguntas e ajudas (ou dicas), que serão apresentadas no módulo do aluno, quando o aluno estiver resolvendo o problema receberá as perguntas cadastradas pelo professor e também terá acesso as dicas (ou ajuda) que o professor cadastra na tela (figura 12).

Na tela de definição das perguntas representada na figura 12 o professor vai identificar uma única pergunta e uma única ajuda (dica) para as classes do problema (item 1 e 2 da figura 12). Será uma única pergunta que será feita cada vez que o protótipo do aluno precisar que o aluno identifique uma classe numa relação de possíveis classes (abaixo, nas classes inválidas, mais sobre o assunto).

Já as perguntas e ajudas de atributos e métodos são uma para cada classe, porém para todos os atributos e métodos da mesma classe será a mesma pergunta e ajuda (item 1 da figura

12). Também existe um combo (item 1, figura 12) para seleção da classe que se está fazendo a pergunta ou ajuda do atributo ou método.

O último item de navegação da árvore é o de classes inválidas, representado na figura 13.

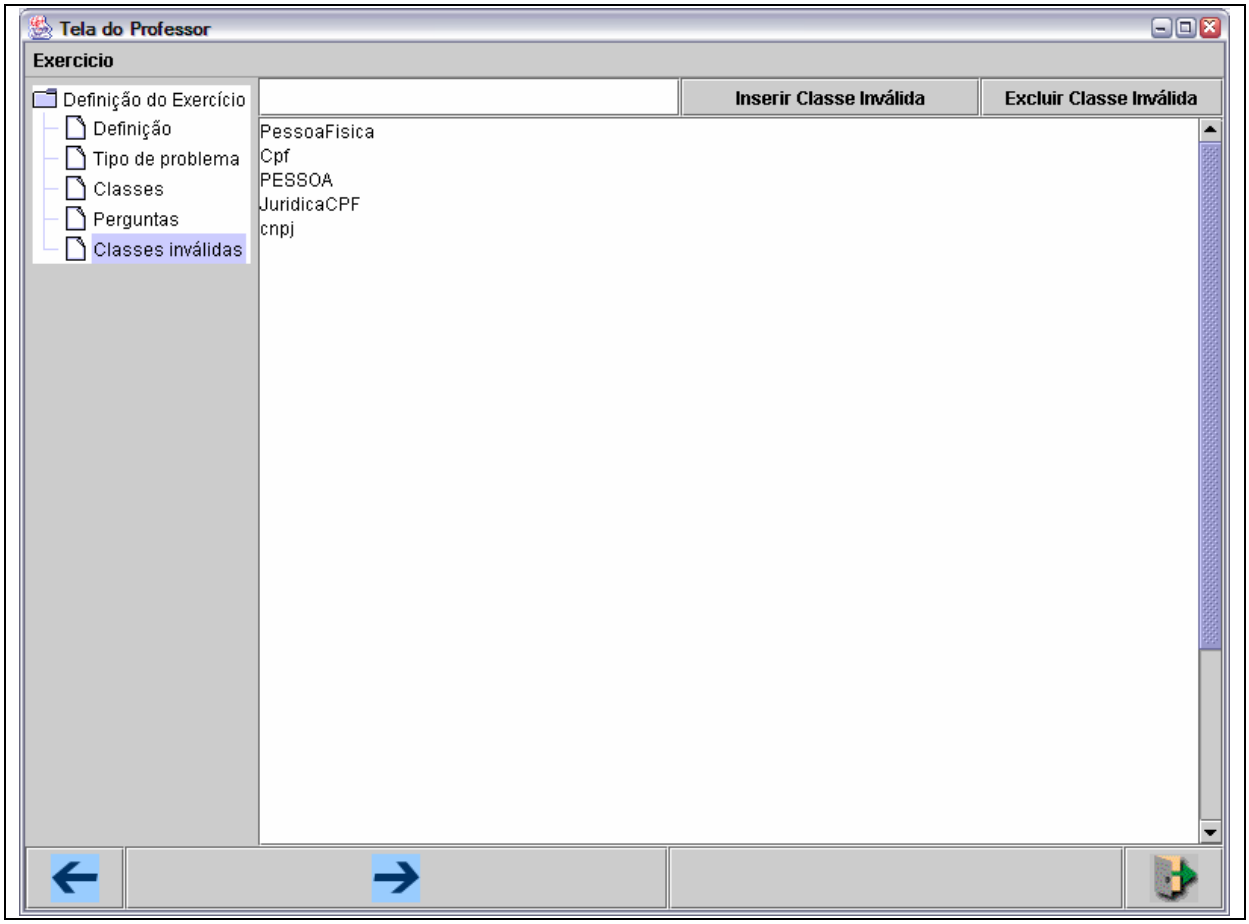


Figura 13: Tela de classes inválidas

Na tela de classes inválidas o professor vai cadastrar nomes de classes que estejam fora do contexto do problema ou que estejam fora dos padrões já passados aos alunos, para serem exibidas no protótipo do aluno com a classe com nome correto entre elas. O aluno deverá identificar a classe com nome correto (de acordo com o contexto e padrões já passados pelo professor). Somente quando ele identificar a classe principal ele poderá seguir adiante no protótipo. Mais detalhes sobre essa situação abaixo na apresentação do protótipo do aluno.

4.3 O MÓDULO DO ALUNO

O módulo do aluno tem como objetivo a realização de um exercício previamente cadastrado pelo professor. O aluno somente poderá interagir com o programa após a abertura de um exercício.

Depois de aberto um exercício, o módulo do aluno iniciará a interação com ele (o aluno). Inicialmente o módulo questiona a existência de uma classe, o módulo forçará o aluno a identificar todas as classes que o professor definiu.

A figura 14 representa o diagrama de atividades referente a abertura de um exercício e identificação de classes.

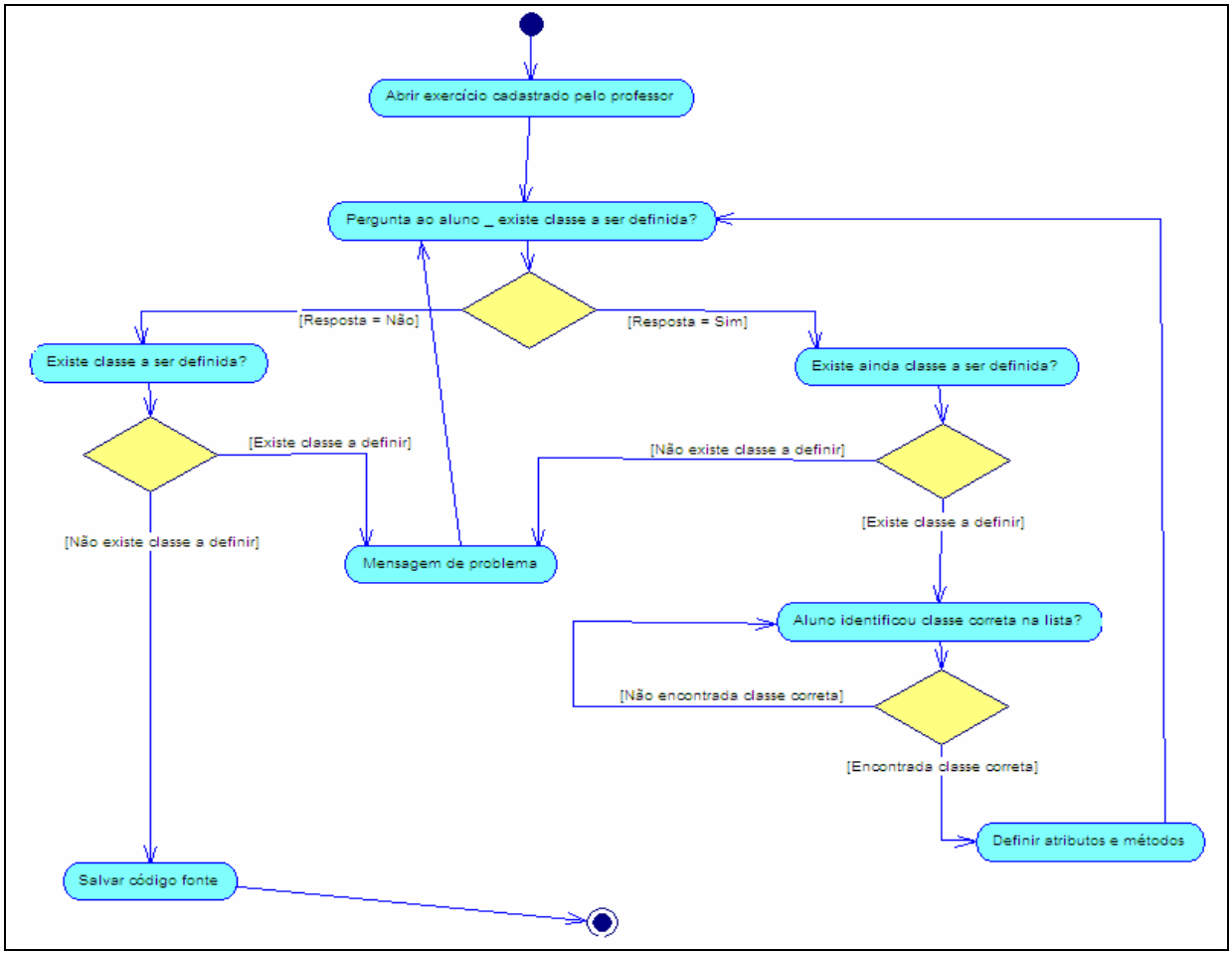


Figura 14: Diagrama de atividades da identificação das classes

Inicialmente o aluno deverá abrir um exercício. Entrarão as questões sobre identificação de nova classe. Depois de identificada uma classe informada, então, o aluno irá detalhar essa classe, identificando os atributos e métodos da mesma. Da mesma forma, o módulo exibe perguntas questionando se existem e quais os nomes dos atributos e métodos.

Na identificação dos atributos e métodos, o aluno poderá identificar mais ou menos métodos que o professor havia identificado. No caso de inconsistência, isto é, do aluno identificar um número diferente de atributos ou métodos com relação ao professor, o protótipo apenas notifica o aluno.

A figura 15 representa o diagrama de atividades da identificação de atributos e métodos de uma classe (continuação da atividade “definir atributos e métodos” do diagrama de atividades exibido na figura 14).

O diagrama de atividades da figura 15 mostra que após a identificação de uma nova classe, o aluno receberá pergunta do protótipo, questionando se existem atributos. Caso o aluno informe que existe ainda atributo, sendo que já ultrapassou o número de atributos cadastrado pelo professor, o protótipo apenas exibe advertência para o aluno informando que há diferença em relação a definição do professor.

A funcionalidade não é bloqueada porque o aluno pode ter uma solução diferente da elaborada pelo professor. Esta característica reforça o aspecto de imprevisibilidade citada por Bizzotto (2003).

Cada resposta **sim** implica na criação de um atributo. O aluno vai informar o nome e tipo de dado do atributo. Caso a resposta seja **não**, o protótipo verifica se ainda deveria ter atributo, caso positivo exibe advertência de diferença e prossegue da mesma forma.

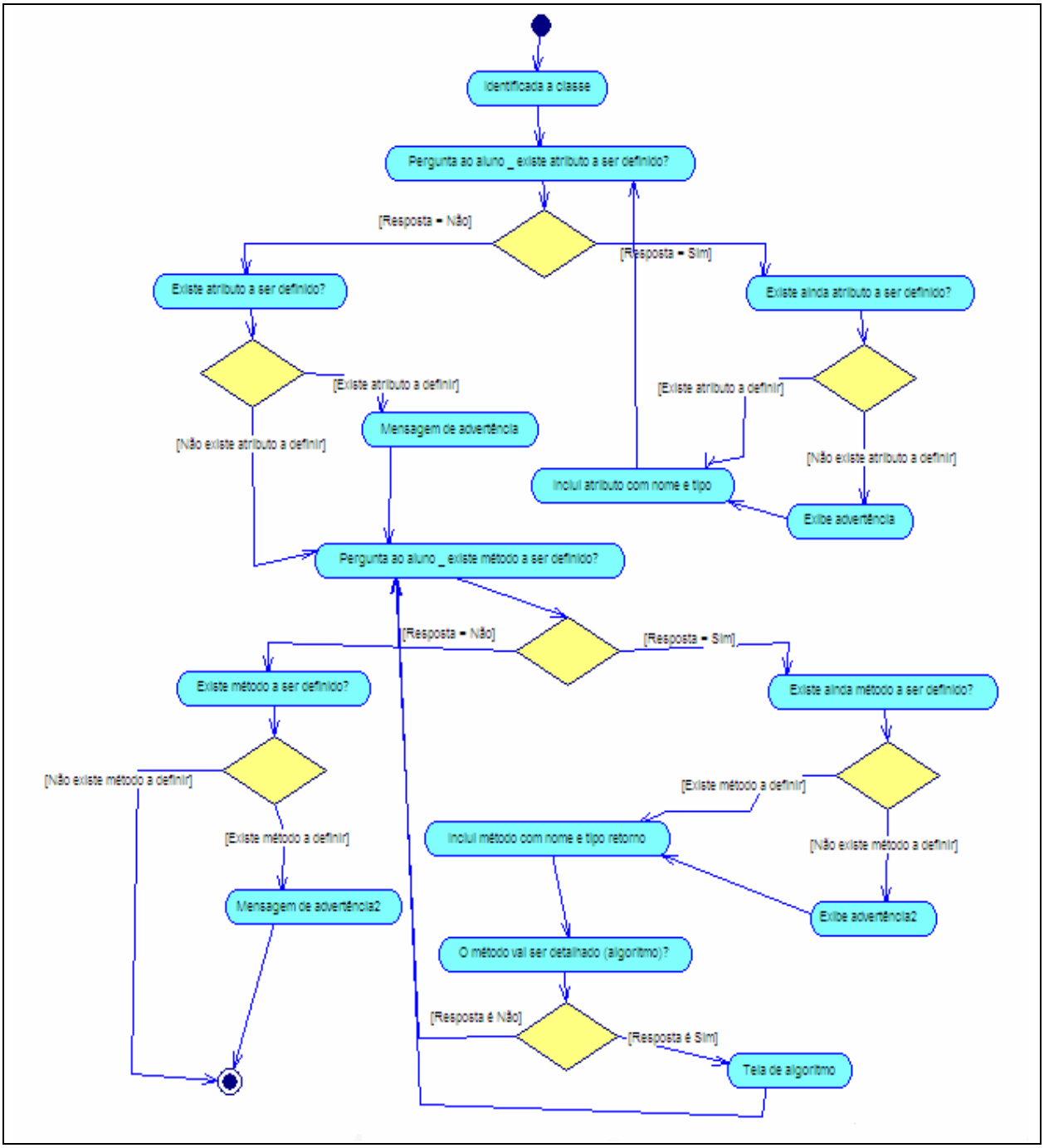


Figura 15: Diagrama de atividades da criação de métodos e atributos pelo aluno

Assim que o aluno informar que não há mais atributos a serem definidos, o módulo passará a apresentar as perguntas referentes aos métodos.

O padrão é o mesmo. Não é exigido que o aluno identifique os mesmos atributos e métodos identificados pelo professor. Apenas são dadas advertências nos casos de

inconsistência com a definição do professor. Quando o aluno informa a existência de um método é solicitado o nome e tipo de retorno do método. Depois disso o módulo questiona o aluno sobre a necessidade de se detalhar esse método. Caso o aluno informe que deseja detalhar o método, é chamada a tela de definição de algoritmo (Figura 25) com as perguntas para se chegar ao algoritmo pretendido pelo aluno.

Assim que o aluno informar que não existe mais método a ser identificado o módulo volta a pergunta sobre a existência de nova classe, de acordo com o diagrama de atividades da figura 14.

No detalhamento de um método, quando se está definindo o algoritmo de um método existem várias perguntas para identificar as estruturas.

O diagrama de atividades dessa parte é muito complexo e não será mostrado. Referências a funcionalidade deste módulo podem ser obtidas em Gubler(2003).

Inicialmente nessa tela (Figura 25) são identificados os parâmetros do método. Depois são apresentadas as perguntas para identificação do algoritmo. As perguntas são exibidas na ordem: pergunta de estrutura de repetição, pergunta de teste, pergunta de atribuição, pergunta de entrada de dados, pergunta de saída e pergunta sobre retorno. Para cada resposta sim nessa parte, o aluno identifica a resposta, por exemplo, se o aluno diz que tem um teste a ser realizado no ponto atual, ele terá que informar a condição do teste e a próxima pergunta a ser feita é a primeira (que é a de estrutura de repetição). A exceção é quando o aluno informa que tem um retorno a ser feito. Neste caso é solicitado o valor a ser retornado e um bloco é finalizado. Caso haja blocos pendentes de definição volta-se à pergunta de repetição (que é a primeira), mas agora de um outro bloco.

Inicialmente, um bloco principal é criado. Cada **Repetição** cria um novo bloco de comandos. A estrutura **Se** cria um bloco para condição verdade e um bloco para a condição falso. Cada um desses blocos será detalhado com seus comandos. O próximo capítulo apresenta um estudo de caso onde essa funcionalidade será detalhada.

A figura 16 representa a tela inicial do protótipo do aluno.

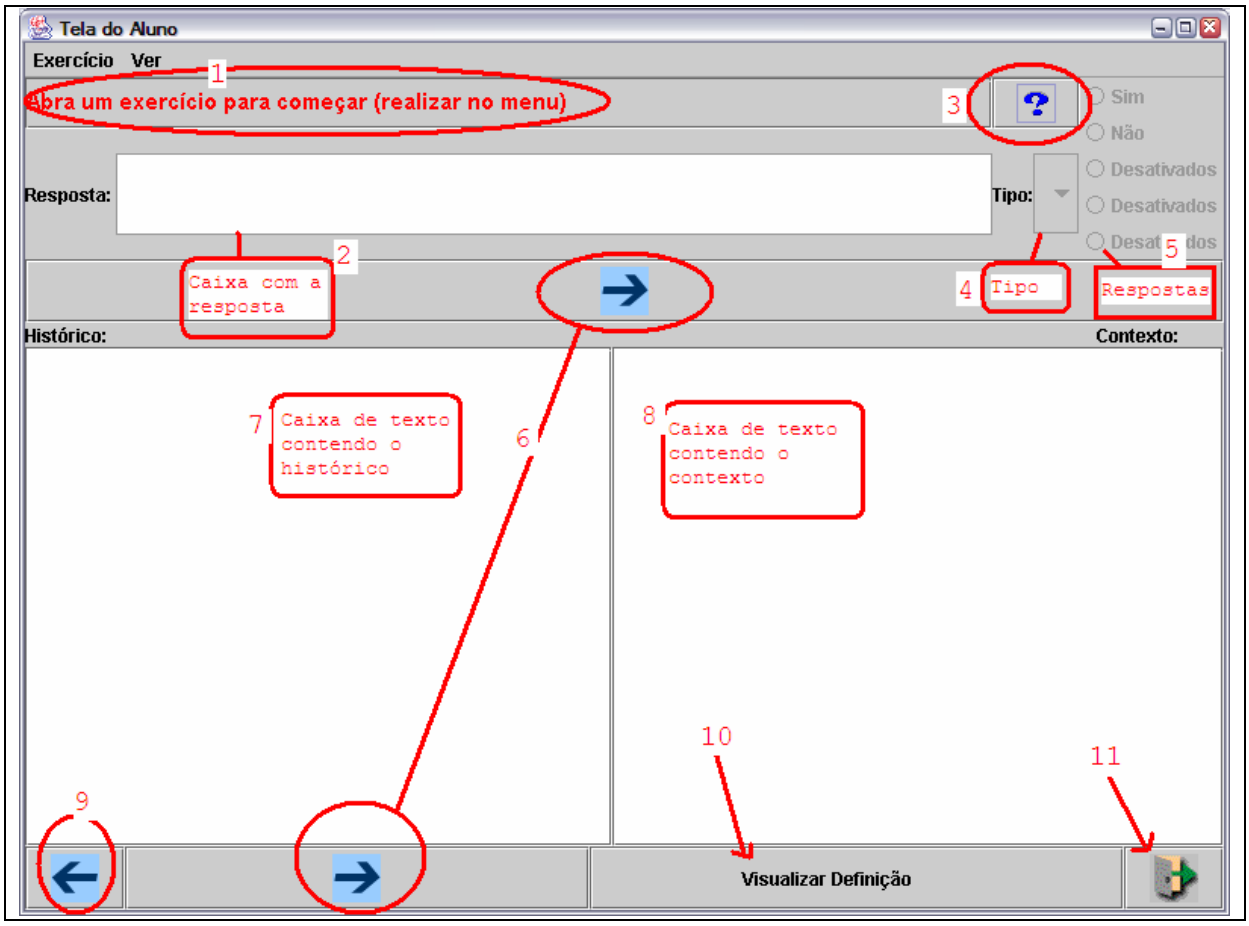


Figura 16: Tela inicial do protótipo do aluno

A tela inicial apresenta o menu com opções de Exercício / Realizar e Sair, e Ver / Definição e Código fonte. A descrição do item 1 (um **JLabel** do Swing na figura 16) é onde são apresentadas as perguntas para interação. É importante ressaltar que esse label que conterá a pergunta atual, tem suas cores alteradas a cada nova pergunta, com o objetivo de chamar a atenção do aluno para que alguma coisa está mudando no contexto. Esse procedimento foi adotado como padrão, tanto na tela de identificação de classes, quanto na tela de detalhamento de método. O label com a pergunta da tela de identificação de classes (figura 16, item 1), inicialmente informa que o aluno deve realizar um exercício, na interação ele trará as perguntas. Ao lado existe um botão de ajuda que mostrará a ajuda cadastrada pelo professor no protótipo do professor.

Existe também um campo Resposta (item 2, figura 16), um combo Tipo (item 4, figura 16) e RadioGroup (item 5, figura 16) no lado esquerdo da tela que receberão as respostas do aluno. O campo de resposta que contém inicialmente um campo para editar, também tem possibilidade de alterar para outras estruturas, de acordo com o tipo de resposta desejado.

Nessa tela também se encontram botões de navegação (itens 6 e 9 da figura 16) e 2 caixas de texto. A caixa de texto “histórico” (item 7 da figura 16) informa todas as perguntas feitas e respostadas dadas pelo aluno. Já a caixa Contexto (item 8 da figura 16) mostra (de uma forma resumida) as classes criadas pelo aluno com seus respectivos atributos e métodos na resolução do exercício.

O item 3 da figura 16 é um botão que permite a visualização de uma ajuda (cadastrada pelo professor no módulo professor, ver figura 12). O item 10 permite a revisão da definição cadastrada no módulo professor (ver figura 11). Para fechar o sistema o aluno pode usar o botão de saída (item 11 da figura 16).

Quando um aluno deseja detalhar um método, e somente nesse caso, é apresentada a última tela (Figura 25). A tela de detalhamento do método tem como objetivo interagir com o aluno para definição de um algoritmo para o método. O código gerado é direto na linguagem JAVA.

Como o aluno somente acessa essa tela quando já tiver definido uma classe e pelo menos um método, é mostrado diretamente o contexto, informando a classe e qual método o aluno está definindo.

A tela de detalhamento do método, que está representada na figura 17, tem o mesmo formato da tela inicial do aluno, mas com algumas diferenças. No menu, o Ver fonte significa visualizar apenas o fonte desse algoritmo (já em JAVA). A opção histórico, que na tela anterior era explícito, agora está acessível apenas pelo menu Ver / histórico.

O importante nessa tela é o código fonte que é exibido automaticamente na caixa de texto “método”. Para facilitar o contexto exibe claramente o bloco que o aluno está definindo, lembrando que o Se contém 2 blocos (um verdade e outro falso) e o enquanto também contém um bloco de comandos.

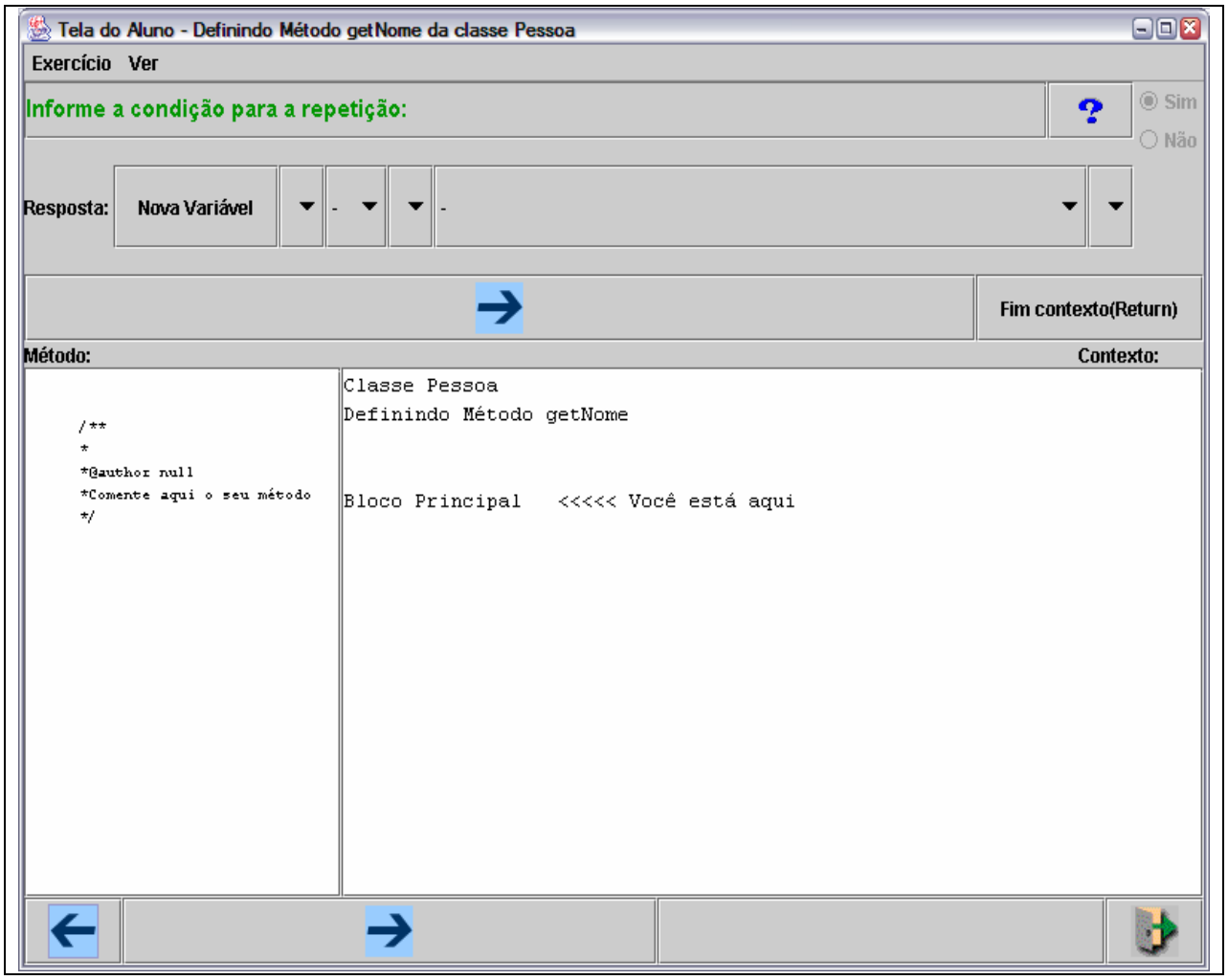


Figura 17: Tela de detalhamento de um método

Na tela de detalhamento (Figura 17) o contexto também é representado no título da tela, deixando claro ao aluno qual método e de que classe ele está definindo.

O botão **“Fim contexto(Return)”** da tela de algoritmos faz com que o aluno possa passar diretamente para a última pergunta (que é a pergunta de retorno), evitando assim com que o mesmo tenha que responder vários “não” até chegar a essa última pergunta. Seria um atalho para a última pergunta.

4.4 IMPLEMENTAÇÃO E FERRAMENTAS UTILIZADAS

Inicialmente foi definida a metodologia de desenvolvimento de prototipação. Devido a complexidade da interface, houve uma preocupação em se desenvolver em JAVA utilizando o Swing. Conforme definido na proposta, devido à complexidade de implementação de uma

interface de boa qualidade no JAVA, já que o JAVA não tem um bom programa para implementação visual da interface.

A opção pela utilização do Swing deveu-se a independência de sistema operacional. Assim, a implementação torna-se disponível em um maior número de plataformas computacionais não limitando o aluno a um particular ambiente para utilização. Para mais leitura sobre Swing ver Deitel e Deitel (2003).

4.4.1 Problemas Encontrados e Soluções Adotadas

A implementação da inteface no JAVA utilizando o Swing não é uma tarefa trivial. Para fazer os ajustes desejados nos componentes, para que esses ficassem de acordo com o desejado, fez-se necessário à criação de muitos painéis (classe JPanel do Swing). Quando esses painéis, começam a crescer em número, a forma de trabalho fica cada vez mais difícil. Portanto a parte de interface ficou bem complexa no protótipo, tanto do modelo do professor, quanto do modelo do aluno. A classe do módulo do professor responsável pela interface tem 1167 linhas de código fonte. A classe do módulo do aluno responsável pela interface tem 783 linhas.

Na implementação da tela do professor, seria necessário a troca de painéis, de acordo com o ramo selecionado na árvore de navegação. No JAVA um painel não pode ocupar o mesmo lugar que outro. Para resolver esse problema foram declarados os 5 painéis existentes (ver árvore de navegação – Figura 9) como atributos da classe da tela (classe MainProfessor). Assim quando a navegação deriva para um ramo diferente retiram-se os 5 painéis do painel principal e insere-se o painel desejado.

O mesmo problema surgiu no módulo do aluno, na caixa de resposta, uma vez que pode haver diferentes tipos de resposta. Esse problema foi resolvido da mesma forma.

Para simular o componente TextGrid do delphi, utilizado no trabalho de Gubler (2002), foi criado um componente com o nome de JTextGrid. Esse componente simula um TextGrid do Delphi. A classe JTextGrid que herda de JTextArea do Swing contém métodos para tratar a String como linhas, simulando assim a característica do Grid. Também foi construído métodos para inserir nessa caixa diretamente um HashMap ou ArrayList (que são lista de objetos) que contenham String. Essas listas geralmente contém um range de valores que seriam exibidos nesses tipos de elementos.

A tela de detalhamento do método foi construída no modelo da tela inicial do aluno. Essa tela inicial tem um objetivo bem mais simples, que é o de permitir a definição de classes, atributos e métodos. Tiveram que ser feitos vários ajustes para o funcionamento da tela de detalhamento de método. Para se saber o contexto (qual classe e método está sendo definido na tela de detalhamento), foi passado como parâmetro a tela inicial, assim como um HashMap que contém os atributos definidos nessa classe. Os atributos podem ser usados na definição desse algoritmo. E assim tenho todo o contexto necessário para exibição nessa tela. Além disso preciso devolver o algoritmo gerado para a tela inicial setar o método que estava sendo detalhado com o algoritmo criado na tela de detalhamento.

A parte estrutural ficou bem definida e de fácil manutenção. Já a parte de interface ficou de difícil entendimento devido aos vários painéis criados para se poder implementar uma tela conforme o desejado. Essas informações são válidas tanto para o módulo do professor, quanto para o módulo do aluno.

Sobre as classes criadas para os protótipos, MainProfessor, MainAluno e MainAlgoritmo implementam as telas. Para fazer a interação entre as telas e as classes base foram criadas as classes ControladorExercicio, ControladorSolucao e ControladorAlgoritmo. A classe ControladorExercicio faz o meio entre um Exercício e a MainProfessor. A classe ControladorSolucao faz o meio entre a classe Solucao e a classe MainAluno. Finalmente a classe ControladorAlgoritmo faz a mediação entre a classe MainAlgoritmo e Algoritmo.

A classe TelaApresentaDefinicao representa uma tela que tem como objetivo apenas mostrar uma caixa de texto com uma informação. Para criação desse tipo de tela basta passar como parâmetro um título e o texto a ser exibido. Essa tela é acessível pelo menu Ver do módulo do aluno, na escolha de qualquer opção, a única coisa que muda são os parâmetros.

Para implementação foi utilizada a ferramenta Eclipse, versão 3.0. Essa ferramenta se mostrou muito boa para implementação e manutenção de projetos JAVA.

O projeto foi estruturado da seguinte forma:

a) foram criados os pacotes (packages) **algoritmo**, **classes**, **gui** e **util**;
b) o pacote **algoritmo** ficou com as seguintes classes (arquivos .java) referente a parte de algoritmo: Algoritmo, Atribuicao, Bloco, Comando, ComandoEntradaSaida, ComandoRepeticao, ComandoSelecao, ControladorAlgoritmo, Enquanto, Escreva,

Leia, Operacao, PilhaAlgoritmo, PilhaBlocos, Retorno, Se, TipoOperacao, TipoPilhaComandos, TiposOperacoes e Variavel, de acordo com a ordem alfabética;

c) o pacote **classes** ficou com as classes referente a exercício, solução e parte de classes (parte OO): Atributo, Classe, ControladorExercicio, ControladorSolucao, Dado, Exercicio, Metodo, Parametro, Pergunta, PerguntaOpcao, PilhaControladorSolucao, Solucao, TipoDado e TipoRetorno, na ordem alfabética;

d) o pacote **gui** ficou com as classes que implementam telas. Essas classes são: TelaApresentaDefinicao, MainAlgoritmo, MainAluno e MainProfessor, seguindo a ordem alfabética. Esse pacote contém um outro pacote chamado **images**, que contém as imagens necessárias para o protótipo;

e) o pacote **util**, contém a classe JTextGrid, que é uma classe necessária para a infra-estrutura do projeto. Em um futuro andamento nesse projeto, as classes necessárias para a infra-estrutura devem ser organizadas nessa pasta.

5 UM ESTUDO DE CASO

Para demonstrar a funcionalidade e o modo de funcionamento dos módulos desenvolvidos para o presente trabalho, foi construído um estudo de caso para um problema clássico apresentado em introdução aos conceitos de orientação a objetos. O problema é fazer a definição das classes Pessoa, Física e Jurídica. Como o protótipo desenvolvido neste trabalho não abrange o conceito de herança da OO, apenas será necessário a definição das classes Pessoa, Física e Jurídica com seus métodos.

A definição que o professor irá cadastrar na tela de definição do protótipo do professor é: "*Resolver o exercício no protótipo definindo as classes Pessoa, Física e Jurídica, com seus respectivos atributos e métodos. A classe Pessoa deverá ter informações sobre nome, fone, data nascimento (aberto para inclusão de mais informações). A classe Física deverá conter o CPF (com possibilidade para mais dados) e a classe Jurídica o CNPJ (também permite-se mais informações).*"

5.1.1 Cadastrando o Exercício

Inicialmente o professor entrará na tela de definição de problema e incluirá a definição mencionada. Após este passo o professor poderá alterar a orientação e tipo de problema na segunda tela da árvore do protótipo, mas, como já citado, essas informações não estão sendo utilizadas no restante do processo. A figura 18 mostra a tela inicial do protótipo do professor, já com a definição incluída.

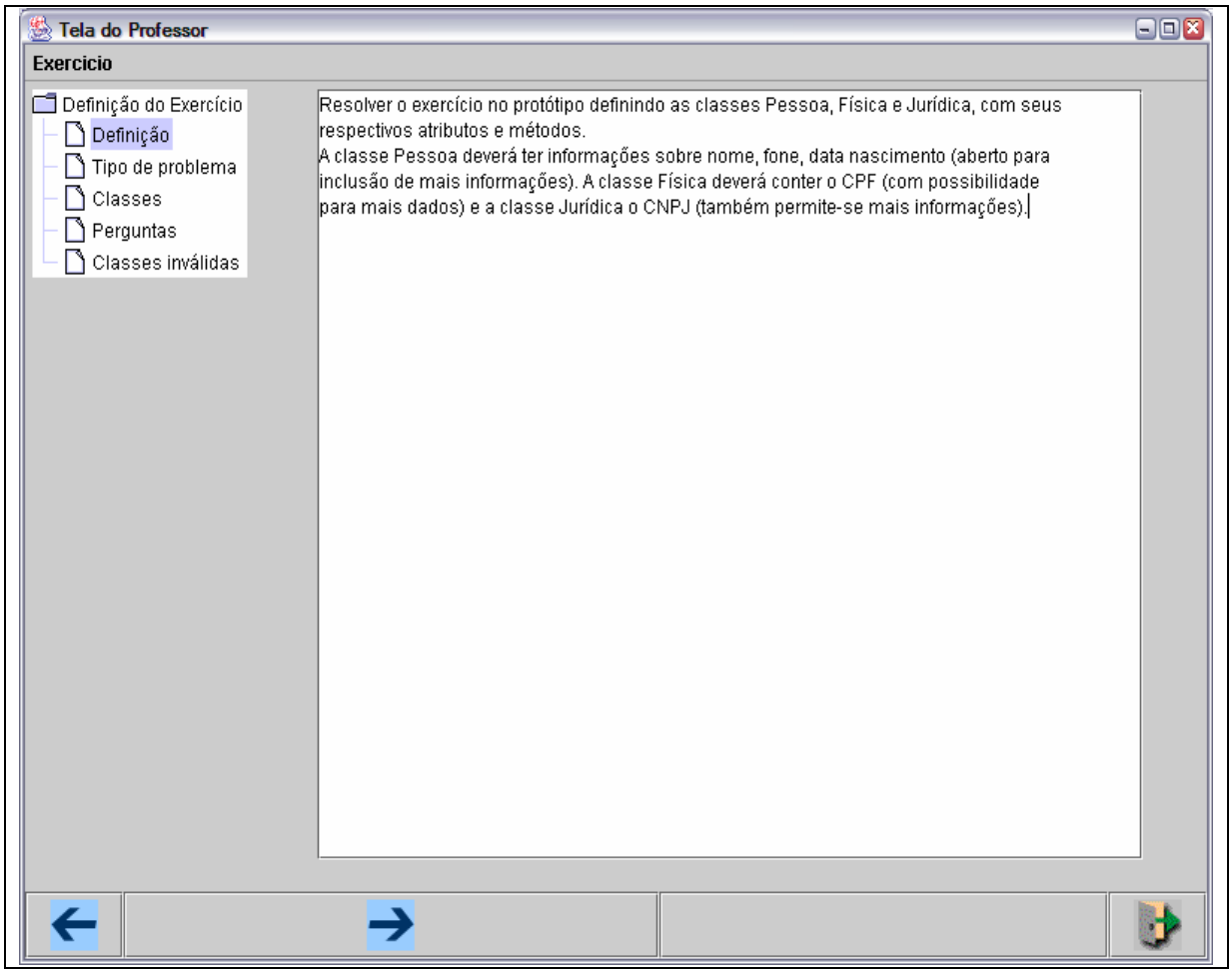


Figura 18: Definindo o problema, tela de definição

Depois de cadastrada a definição, foram definidas as classes que deverão ser identificadas no problema. Nessa tela foi cadastrado a classe Pessoa, para classe Pessoa cadastrado os atributos nome, fone e dataNascimento, a classe Fisica, para a classe Fisica cadastrado o atributo CPF, a classe Juridica, para a classe Juridica foi cadastrado o CNPJ. Os métodos cadastrados para essas classes são somente os métodos set e get, exemplo: para a classe Pessoa e atributo nome, cadastrado o método setNome e getNome.

O próximo item a ser cadastrado é na tela de perguntas. Foi definida a seguinte pergunta para a pergunta de classes: “Dentre as classes listadas, qual você identifica como necessária para o problema?”, como dica (ou ajuda) ficou: “Reveja definição, definir as classes Pessoa, Fisica e Juridica de acordo com o padrão passado” As perguntas de atributos de uma classe ficaram: “Identifique atributos para a classe xxx“, onde xxx é Pessoa para a classe Pessoa, Física para a classe Fisica e Jurídica para a classe Juridica. As ajudas para os atributos informam as informações que devem ser armazenadas. Para as perguntas de métodos

ficaram no mesmo formato das de atributos, apenas mudando a palavra “atributo” para “método”. As ajudas referente a pergunta de métodos ficaram: “Métodos get e set somente”.

A tela 19 representa a tela de perguntas com as perguntas referente a atributos e métodos da classe Pessoa.

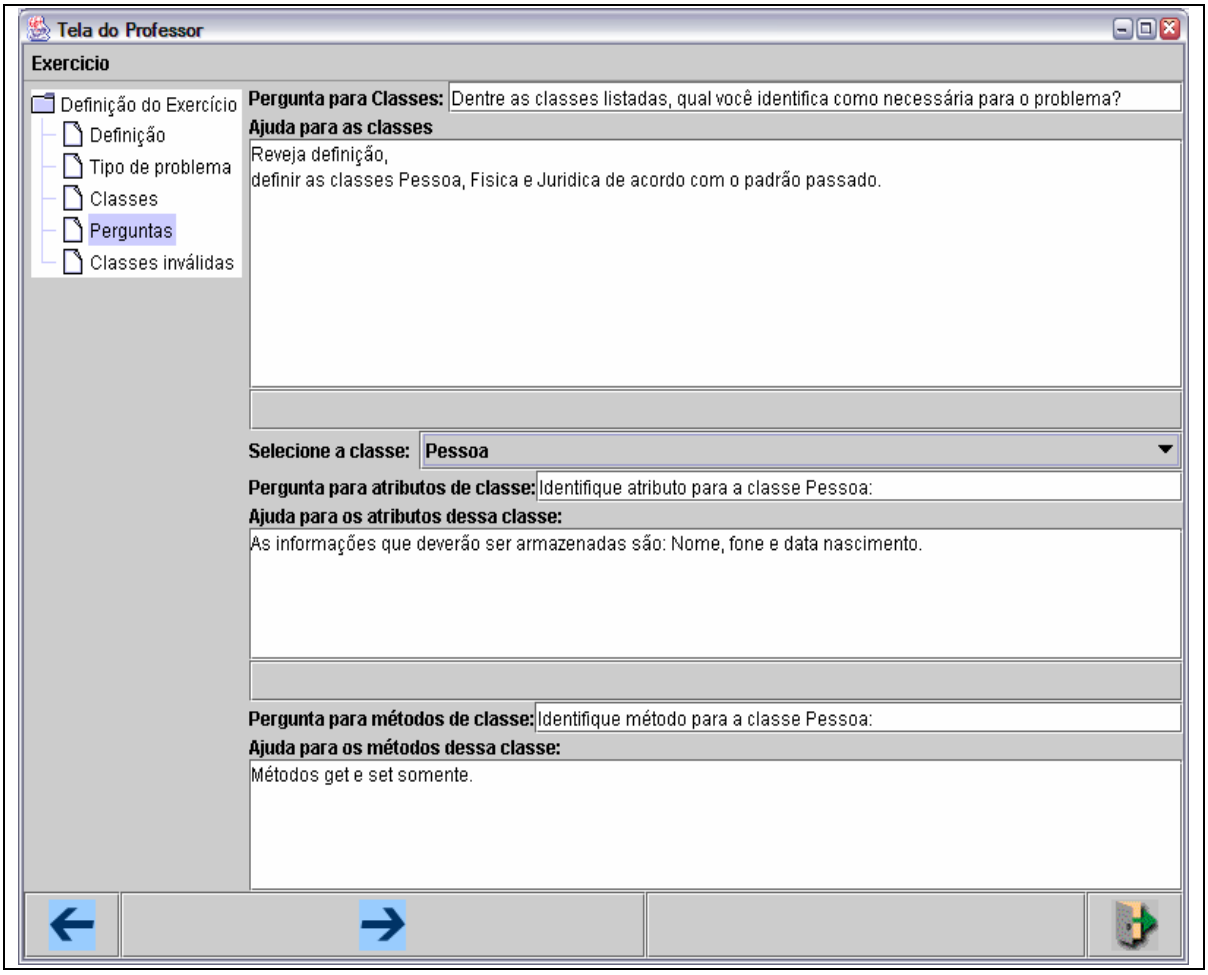


Figura 19: Tela de Perguntas – definindo as perguntas do exercício

Na última tela desse protótipo, foi definido os seguintes nomes como sendo de classes inválidas: PESSOA (porque está fora do padrão da Sun, que seria apenas a primeira letra maiúscula e o restante todas minúsculas – neste caso o professor teria que ter passado essa informação aos alunos), PessoaFISICA, PessoaCPF, CNPJ e Pjuridica.

Após essa seqüência de passos o exercício deve ser salvo para que mais tarde possa ser realizado.

5.1.2 Realizando o Exercício Cadastrado

Para a resolução do exercício cadastrado na tela do professor, é escolhido a opção de menu exercício / Realizar e aberto o exercício cadastrado pelo professor. Na abertura de um exercício é disparada a tela de visualização de definição do exercício, mostrando a definição cadastrada pelo professor.

A figura 20 mostra a tela de definição do exercício para o aluno, essa informação é cadastrada na primeira tela do protótipo do professor.

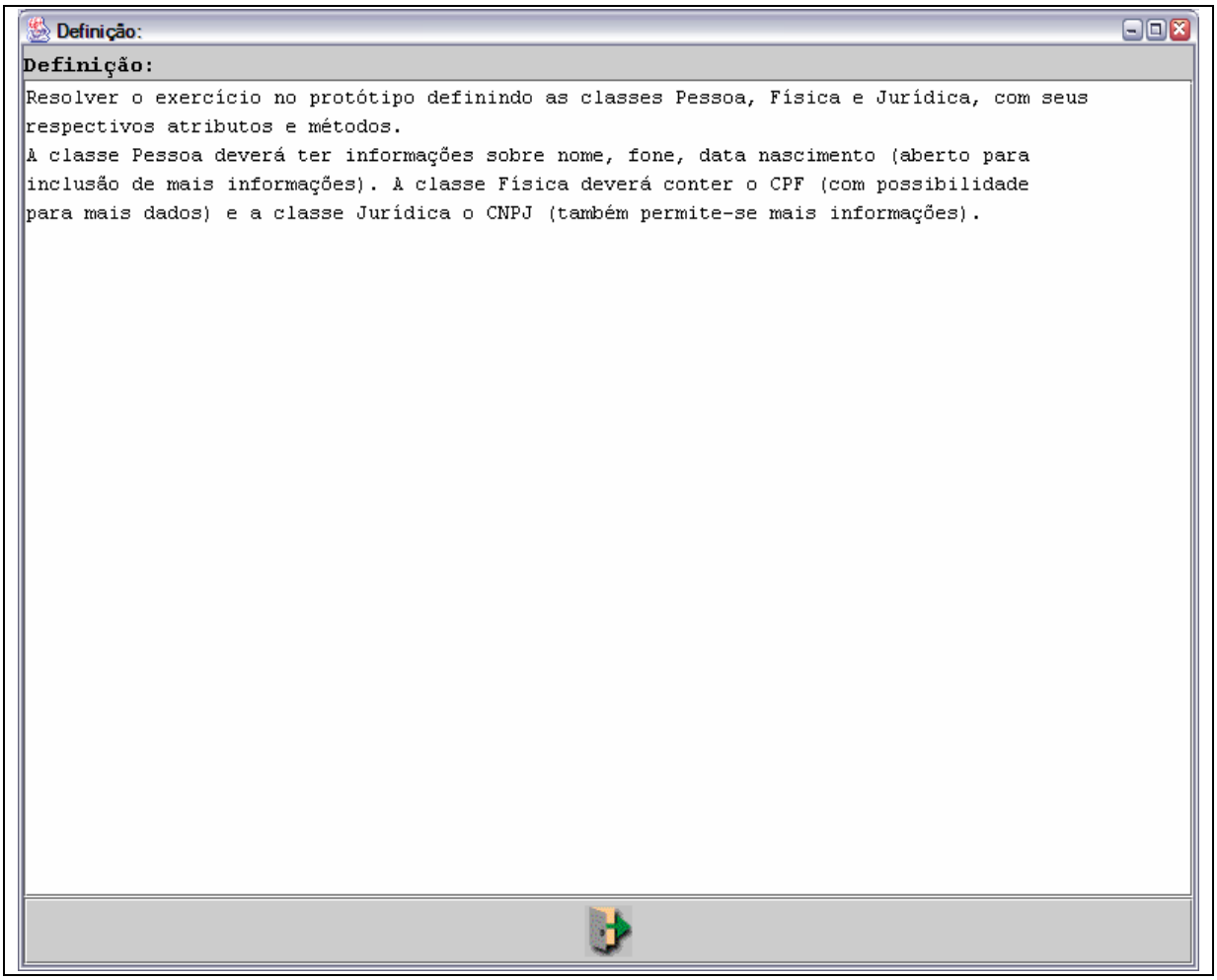


Figura 20: Tela de definição do protótipo do aluno

O aluno deverá ler a definição e fechar essa tela. Assim que a tela for fechada, a primeira pergunta já está exibida na tela. A pergunta “Existe ainda alguma classe identificada e não informada?”, com opções de resposta “Sim” ou “Não”. O protótipo irá forçar o aluno a identificar as 3 classes cadastradas pelo professor, no caso de uma resposta “Não” no atual

momento (onde não foram ainda identificadas nenhuma classe), mostra uma mensagem de erro informando que existem classes ainda a serem criadas, forçando o aluno a responder que “Sim”. Foi respondido “Sim” para essa pergunta.

A próxima pergunta é mostrada a pergunta de classes, cadastrada no protótipo do professor, tela de perguntas. A ajuda nessa tela é também a cadastrada pelo professor. As opções de respostas são as classes inválidas cadastradas pelo professor com a primeira classe definida pelo professor entre elas. O aluno terá que identificar a classe correta entre as erradas. Caso o aluno não identifique a correta é exibida mensagem e refeita a pergunta, forçando o aluno a responder a classe correta.

A figura 21 mostra a tela de identificação de nova classe. A classe Pessoa, é a que deve ser identificada, as demais são Classes inválidas do cadastro do professor.

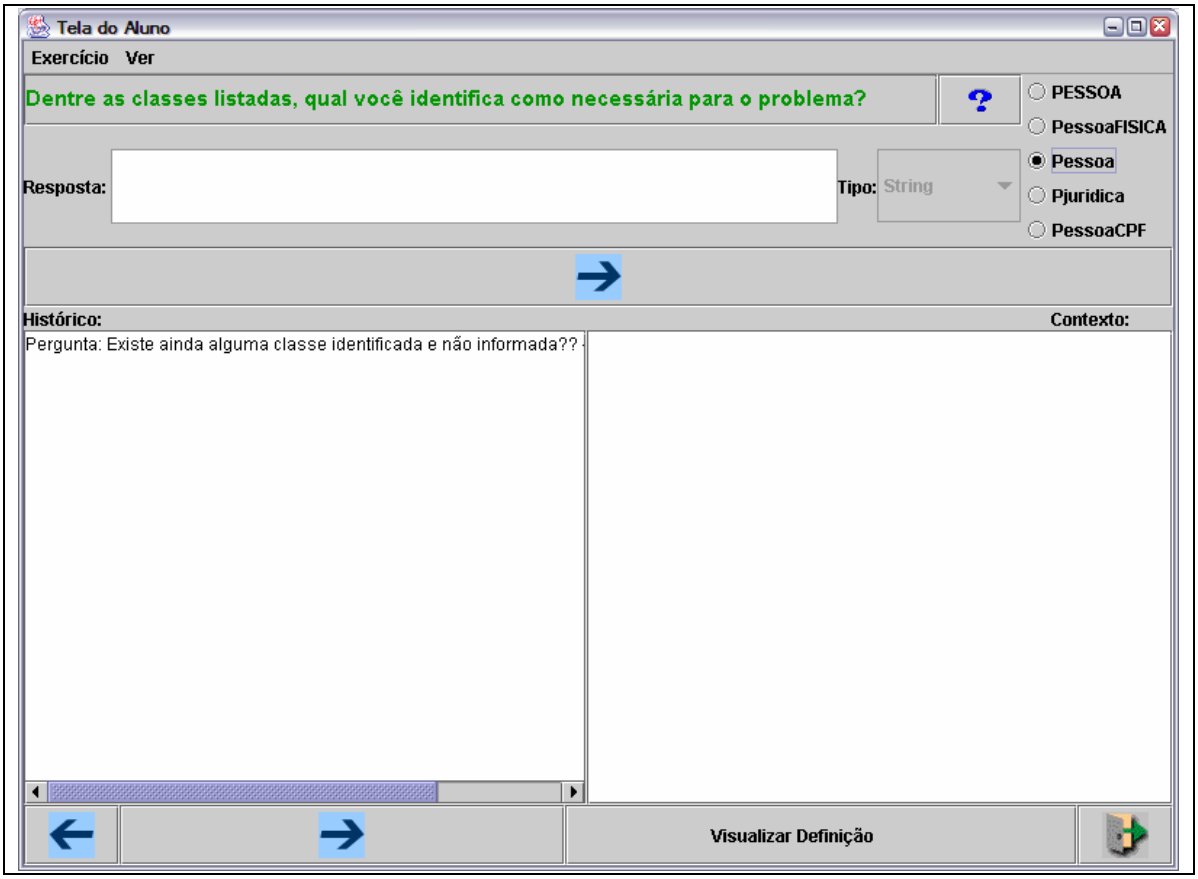


Figura 21: Identificando nova classe

Após a identificação da classe correta é exibida mensagem, conforme figura 22.

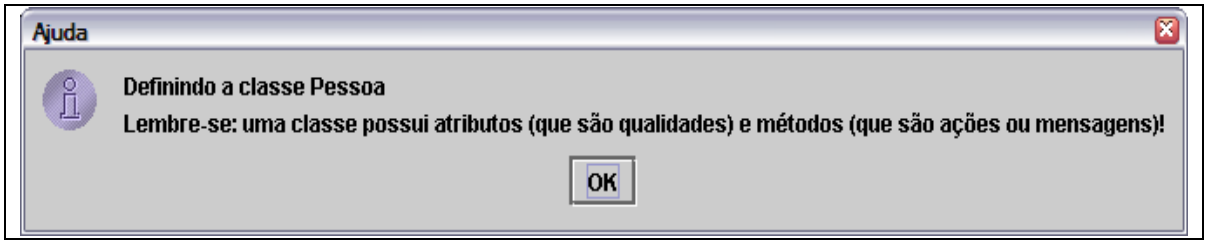


Figura 22: Mensagem após identificação de nova classe

Foi pressionado o botão OK da tela representada na figura 20. A próxima pergunta é sobre os atributos da classe Pessoa. É questionado se existe atributo para classe Pessoa. Caso a resposta seja não, apenas será exibido mensagem de alerta que há atributos que não foram identificados, mas vai ser possível passar para a próxima pergunta, ou seja, não é bloqueado. Para a atual demonstração foi respondido que “Sim”, existem atributos a serem criados. Na seqüência é exibida a mensagem sobre o nome do atributo. Essa mensagem é a mensagem cadastrada na tela do professor referente a atributos da classe Pessoa. A ajuda disponível nessa etapa também é a ajuda cadastrada na tela do professor, tela de Perguntas na ajuda de atributos da classe Pessoa. Foi respondido “nome” para o nome do atributo e escolhido o tipo String na lista de tipos, conforme figura 23.

A próxima pergunta é novamente a questão se existe ainda atributos a serem identificados. Até que seja dada uma resposta que “Não” existem atributos a serem identificados. No atual exemplo foi identificado ainda os atributos fone (String) e dataNascimento (Date). Após foi informado que não existem mais atributos a serem cadastrados. Caso fosse informado que existiam mais atributos a serem cadastrados, seria ultrapassado o número de atributos cadastrado para classe Pessoa no protótipo do professor. Seria exibida mensagem de erro, mas é permitido que o aluno cadastre mais atributos.

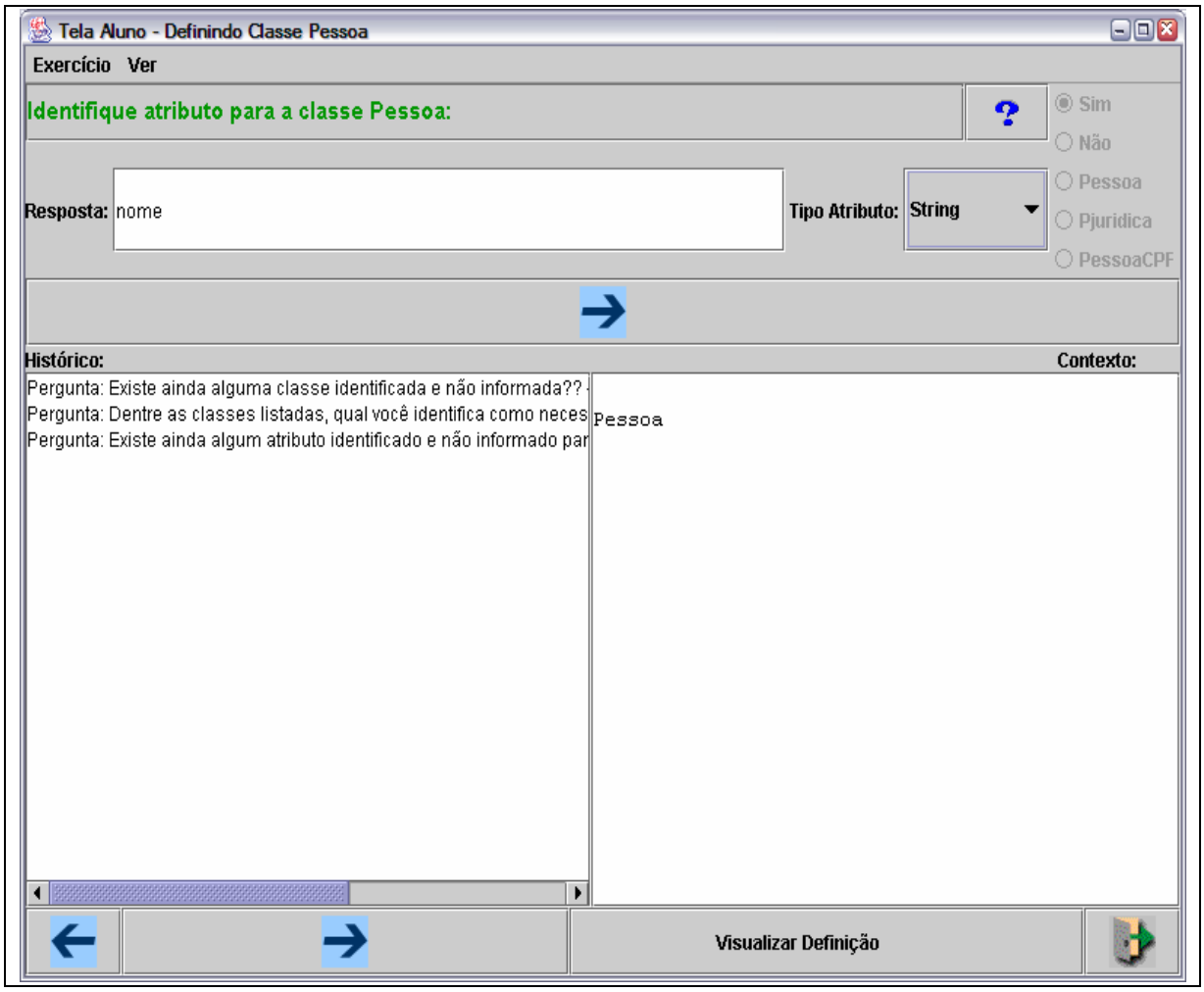


Figura 23: Identificando um novo atributo

Depois da resposta que “Não” existem mais atributos a serem definidos para classe Pessoa, vem as perguntas de métodos. A questão de se existem métodos ainda não cadastrados para a classe é exibida. O funcionamento é de acordo com as questões de atributos. A resposta foi “Sim” e a próxima pergunta é a pergunta cadastrada na tela do professor com relação a pergunta de métodos da classe Pessoa. A ajuda também é a ajuda cadastrada para métodos da classe Pessoa. A resposta foi setNome para o “nome” do método e void (vazio) para o tipo de retorno.

A figura 24 mostra a tela de identificação do método.

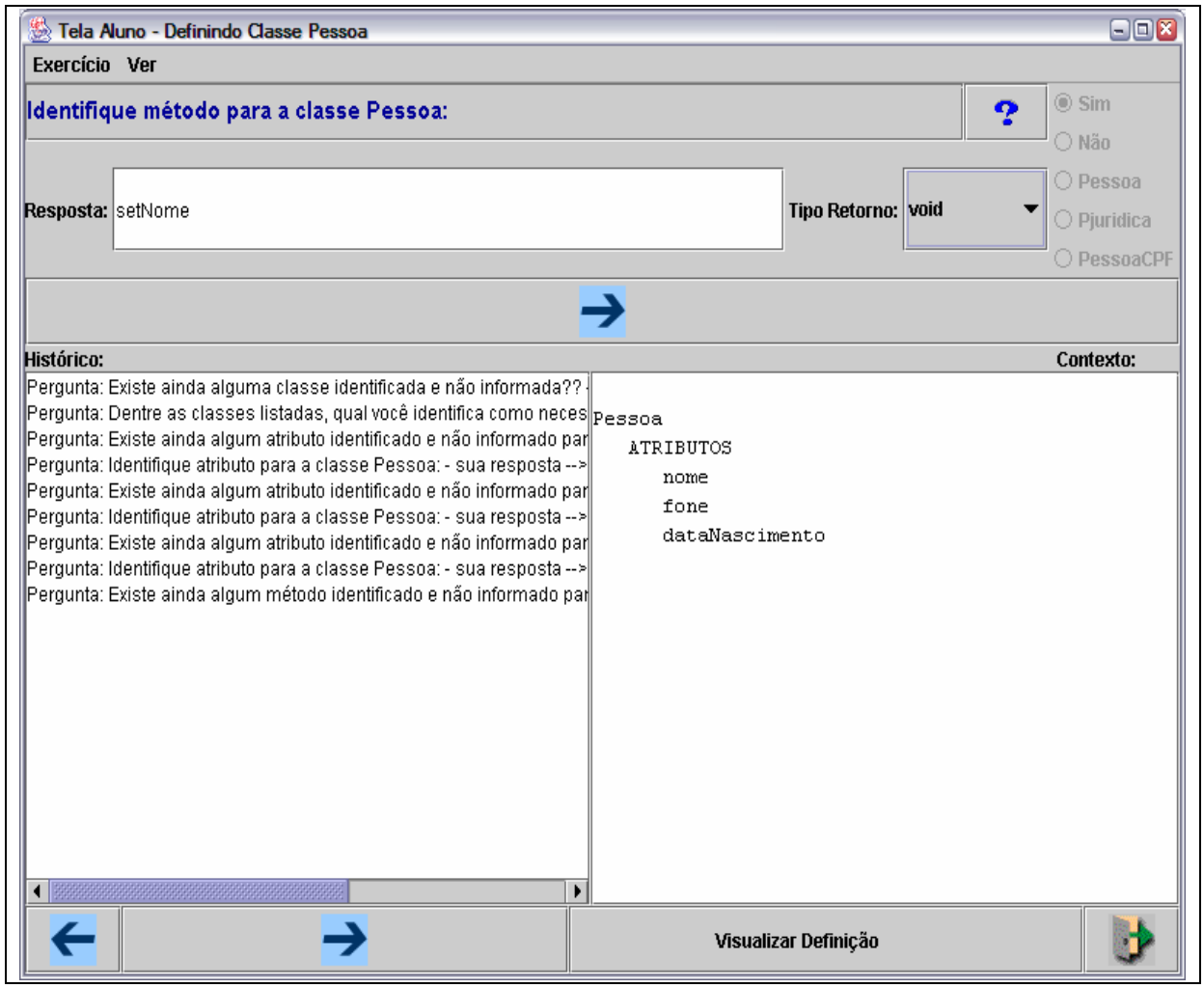


Figura 24: identificando novo método

Já é possível ver nessa tela a atualização do histórico e do contexto em tempo real. No contexto é possível ver a classe Pessoa com os atributos criados.

Após a criação do método é feita a pergunta se o aluno deseja ou “Não” detalhar esse método. Caso a resposta seja não a pergunta já é feita para o próximo método, no caso da resposta for “Sim” é exibida a tela de definição de algoritmo. Foi dada a resposta “Sim” na seqüência do exemplo.

Na tela de detalhamento do método, é solicitado inicialmente a criação de parâmetros (quantos existam) para o método. Para o exemplo foi criado o parâmetro pNome do tipo String. Após criados os parâmetros é feita a interação referente a algoritmos. A seqüência de perguntas feitas é: pergunta sobre repetição, pergunta sobre teste, pergunta sobre atribuição, pergunta sobre digitação (leia), pergunta sobre exibição de informação (escreva) e para

finalizar pergunta de retorno. Para cada resposta sim, com exceção da pergunta de retorno, é voltado para a pergunta de repetição.

No exemplo foi feita uma atribuição que o atributo nome recebe o parâmetro pnome. Para facilitar a navegação existe o botão de “Fim contexto(Return)”, foi pressionado nesse botão e o sistema foi diretamente para a última pergunta sobre a existência de um retorno, foi informado “Não” (não tem retorno).

A figura 25 mostra a tela de detalhamento do método setNome da classe Pessoa. O contexto, além de exibido na caixa de contexto também está no título da janela. A parte interna do método é exibida em tempo real na caixa método ao lado esquerdo da tela (figura 25).

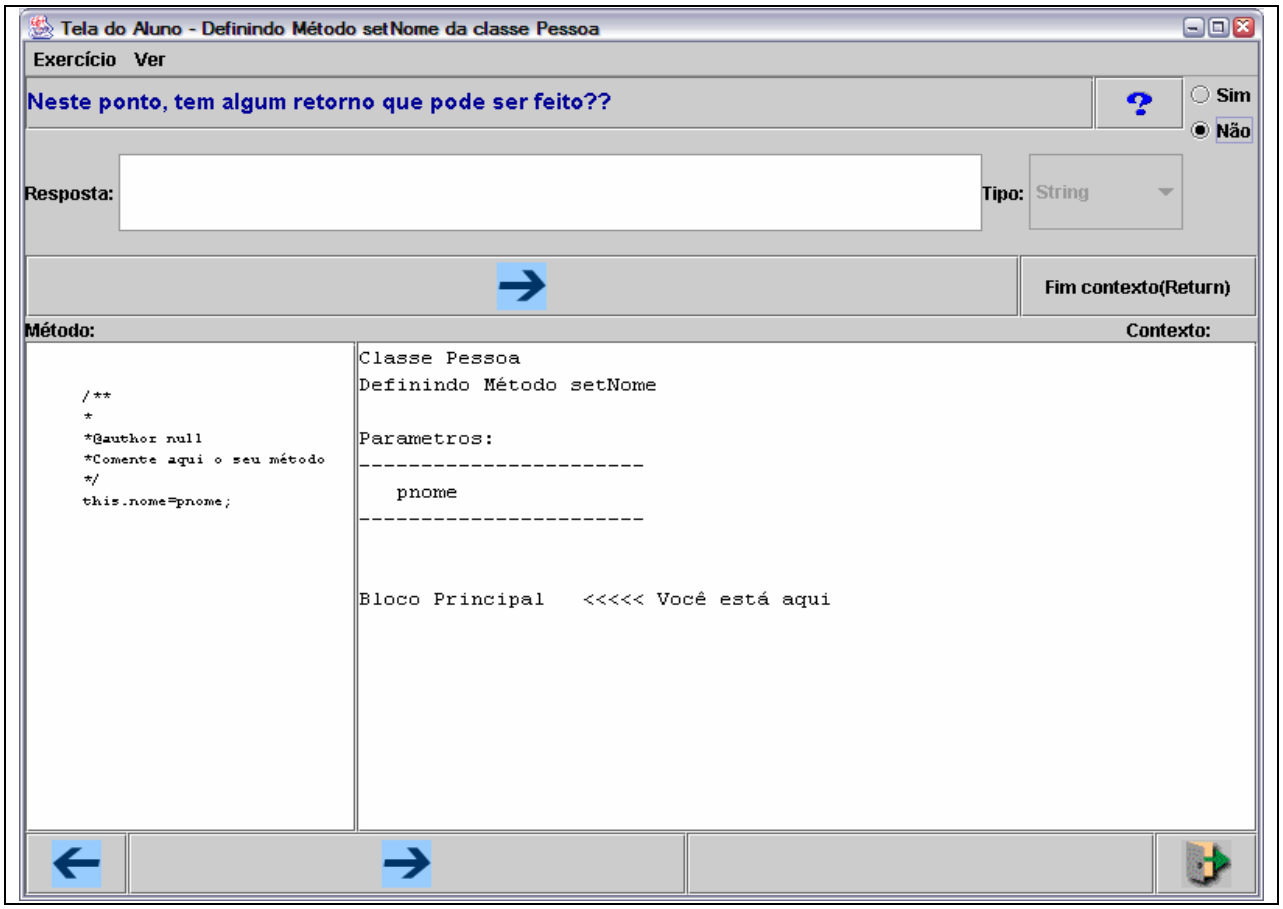


Figura 25: Tela de detalhamento do método

Após o fim do detalhamento do método é fechada essa tela e volta-se para a tela onde se está definindo métodos, para a próxima pergunta referente questão de métodos.

Ao acessar o menu “Ver / Fonte gerado” pode ser verificado o fonte gerado até esse ponto na linguagem JAVA. A figura 26 mostra a tela de fonte gerado depois de identificado os atributos e definido o método setNome da classe Pessoa.

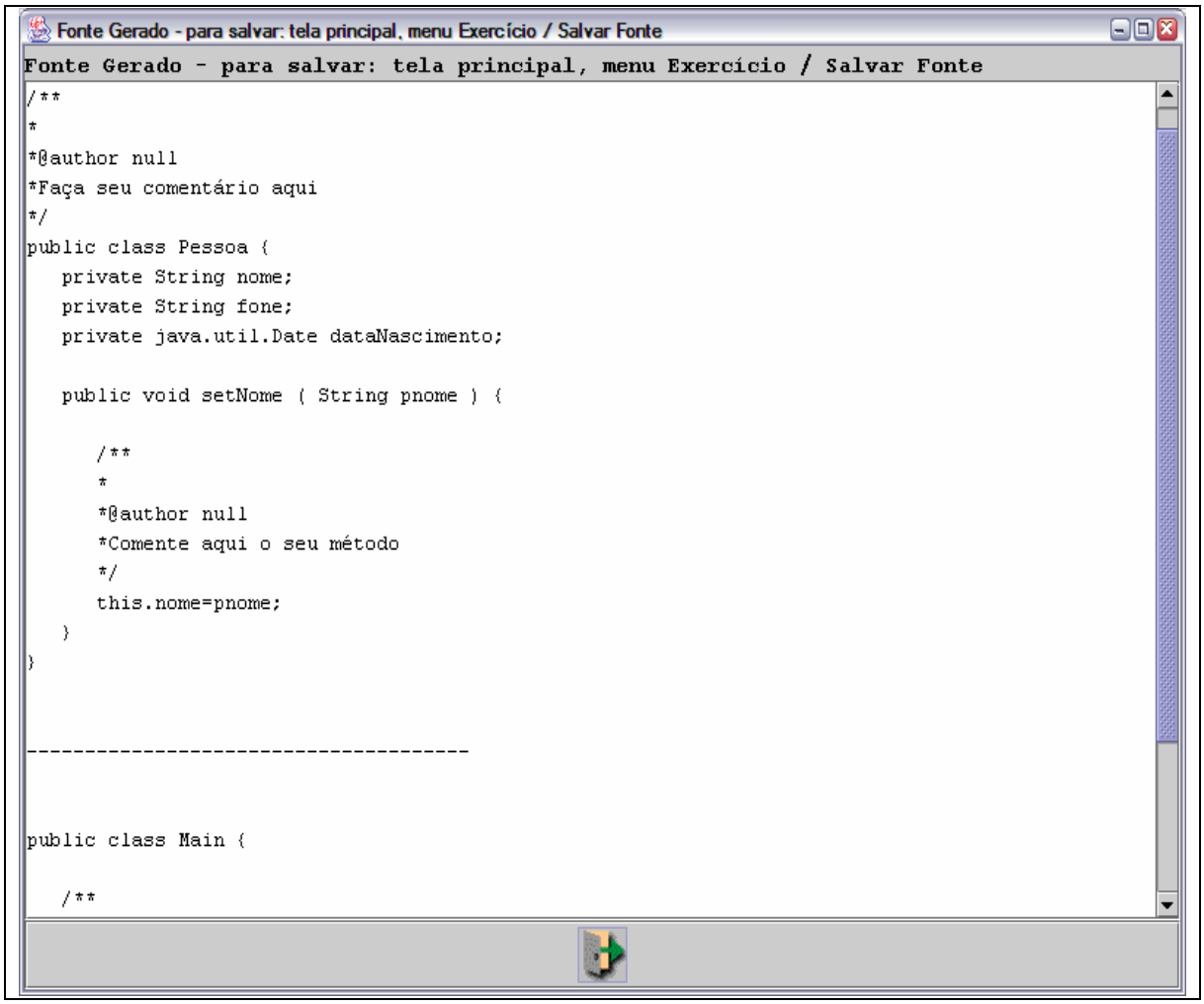


Figura 26: Visualizando fonte gerado até aqui

Foram definidos também os métodos getNome, com retorno String, setFone (retorno void) e getFone (retorno String), setDataNascimento (retorno void) e getDataNascimento (retorno Date). Uma classe Main foi criada para execução do método “public static void Main (String [] args)”. O nome do autor está como nulo, isto porque o sistema não foi

acessado pela tela inicial onde se pede o nome do aluno e o nome do projeto. O autor ficaria com o nome digitado nessa tela e a classe principal, ao invés de Main ficaria com o nome do projeto.

Após informados todos os métodos dessa classe e informado que não há mais métodos, a pergunta novamente é feita em torno de classes. A classe a ser identificada é a classe Fisica. Foi identificada a classe Fisica com o atributo CPF (tipo int) e método getCpf (retorno int) e método setCpf (retorno vazio). O mesmo procedimento foi feito para a classe Juridica identificando o atributo CNPJ com seus métodos set e get.

O contexto no final ficou como representado na figura 27.

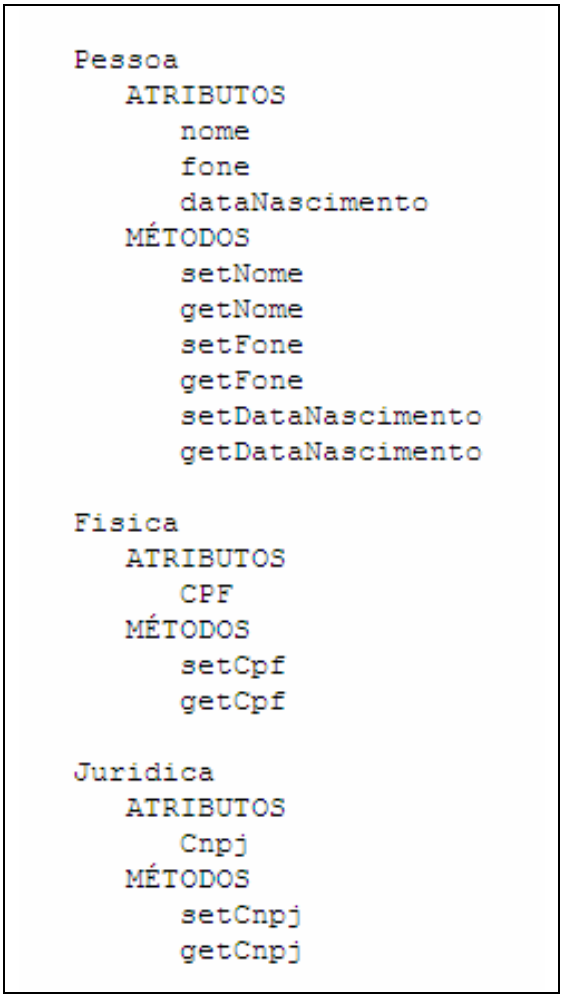


Figura 27: Contexto final na tela do aluno

No final foi salvo o fonte através do menu “salvar fonte”. O arquivo fonte final da resolução, referente a classe Pessoa, salvo está no quadro 2.

```
/**
*
*@author null
*Faça seu comentário aqui
*/
public class Pessoa {
   private String nome;
   private String fone;
   private java.util.Date dataNascimento;

   public void setNome ( String pnome ) {

      /**
      *
      *@author null
      *Comente aqui o seu método
      */
      this.nome=pnome;
   }
   public String getNome (  ) {

      /**
      *
      *@author null
      *Comente aqui o seu método
      */
      return this.nome;
   }
   public void setFone ( String pfone ) {

      /**
                                              *
      *@author null
      *Comente aqui o seu método
      */
      this.fone=pfone;
   }
   public String getFone (  ) {

      /**
      *
      *@author null
      *Comente aqui o seu método
      */
      return this.fone;
   }
   public void setDataNascimento ( java.util.Date pDataNascimento ) {

      /**
      *
      *@author null
      *Comente aqui o seu método
      */
      this.dataNascimento=pDataNascimento;
   }
   public java.util.Date getDataNascimento (  ) {

      /**
      *
      *@author null
      *Comente aqui o seu método
      */
      return this.dataNascimento;
   }
}


------------------------------------
```

Quadro 2: Parte do fonte referente a classe Pessoa

O quadro 3 representa o restante do código fonte gerado.

```
/**
*@author null
*Faça seu comentário aqui
*/
public class Fisica {
   private int CPF;

   public void setCpf ( int pCpf ) {
      /**
      *@author null
      *Comente aqui o seu método
      */
      this.CPF=pCpf;
   }
   public int getCpf (  ) {
      /**
      *@author null
      *Comente aqui o seu método
      */
      return this.CPF;
   }
}
-------------------------------------
/**
*@author null
*Faça seu comentário aqui
*/
public class Juridica {
   private int Cnpj;

   public void setCnpj ( int pCnpj ) {
      /**
      *@author null
      *Comente aqui o seu método
      */
      this.Cnpj=pCnpj;
   }
   public int getCnpj (  ) {
      /**
      *@author null
      *Comente aqui o seu método
      */
      return this.Cnpj;
   }
}
-------------------------------------
public class Main {
   /**
   *@author null
   *
   */
   public static void main (String[] args) {
   }
}
```

Quadro 3: Fonte gerado

5.2 RESULTADOS E DISCUSSÃO

Este trabalho atingiu o objetivo de fazer uma extensão do trabalho de Gubler (2002), e contemplar a implementação de um simples problema de OO. O protótipo apresentado teve um grande diferencial dos demais trabalhos citados no item 5 (Trabalhos Correlatos), com a implementação da tela do professor, onde agora se permite que o professor faça o cadastro de um exercício para a solução posterior do aluno.

Apesar de não terem sido realizados testes com alunos, pode-se concluir que o protótipo se mostrou de simples utilização, mas é indicado para alunos que estão sendo apresentados aos conceitos de OO, devido a existência das várias perguntas que podem acabar tirando o interesse de quem tem um conhecimento um pouco maior.

O protótipo atendeu o objetivo da geração de código fonte em JAVA e a forma que foi concebido permite a implementação de geração de código fonte em novas linguagens de forma relativamente simples.

Não foi contemplada a geração de diagramas de casos de uso, diagrama de contexto e diagrama de classe. A justificativa para este fato é que inicialmente se esperava que o resultado do projeto de pesquisa “implementação do tutor de algoritmos versão 2.0” (Mattos 2003) Implementação do tutor de algoritmos versão 2.0. projeto pipe.) desenvolvido na FURB pudesse ser agregado ao contexto do projeto atual. Como isto não foi possível em função da incompatibilidade de modelos de implementação, foi necessário o desenvolvimento do módulo de algoritmos o que impôs um atraso importante no cronograma original.

Há algumas definições na orientação a objetos que este sistema não abrange, como a herança, associação, agregação, etc. A implementação desses conceitos implica na complementação da árvore de navegação, o que não ocorreu em função do comprometimento do cronograma não foram contemplados.

6 CONCLUSÕES

A proposta apresentada por Mattos (2000) estendida por Gubler (2002) são interessantes no sentido de auxílio do aluno em uma disciplina de difícil compreensão que é a disciplina de introdução a programação, principalmente pelo fato dos protótipos interagirem com o aluno buscando o resultado.

O objetivo principal do trabalho que era de se desenvolver um protótipo nos moldes do protótipo desenvolvido em Gubler (2002) e que atendesse agora também a orientação a objetos foi alcançado. O protótipo desenvolvido mostrou-se capaz de guiar o aluno na resolução de um exercício proposto. O protótipo também é capaz de gerar código fonte em JAVA, atingindo assim esse objetivo.

Foram realizados vários testes, ainda sem a participação de alunos, identificando que os resultados foram conforme os desejados, com geração de código fonte na linguagem de programação JAVA, inclusive com possibilidade de comentários no formato JAVADOC, mas ainda com limitações. A existência apenas do comando **Se** como seleção e **Enquanto** como repetição, também são limitações do protótipo. O protótipo somente permite possibilidade de se definir apenas os tipos primitivos do JAVA e a classe String.

6.1 EXTENSÕES

Como possíveis extensões a esse trabalho sugere-se:

a) a possibilidade de geração de diagramas de casos de uso, diagramas de classe e de contexto;
b) a implementação das funcionalidades referente a dificuldade do problema utilizando os itens orientação e tipo de problema incluídos na tela do professor (Figura 10);
c) a implementação de todos os tipos de comandos possíveis no JAVA, como o switch, for, etc.
d) a possibilidade de se definir tipos como tipos de classes. Como exemplo definir: “java.lang.Integer inteiro;”;
e) a possibilidade de se colocar comentários de código automaticamente nos métodos, atributos, definindo melhor o contexto e objetivos, possibilitando ao aluno a edição desses comentários de código;

f) a possibilidade de extensão de outras linguagens na geração de código fonte, assim como otimização desse código;
g) contemplar os demais conceitos de OO, como a possibilidade de chamada de métodos, herança, etc.

REFERÊNCIAS BIBLIOGRÁFICAS

BIZZOTO, Carlos Eduardo Negrão. **O Aprendiz:** ambiente extensível para o aprendizado distribuído. 2003. 123 f. Dissertação (Pós-Graduação em Engenharia de Produção) – Universidade Federal de Santa Catarina, Florianópolis.

CAMPOS, Gilda H. B. de. **Metodologia para avaliação da qualidade de software educacional:** Diretrizes para desenvolvedores e usuários. Rio de Janeiro: Publicações Técnicas, 1994.

COELHO, Cláudio Ulysses F.; HAGUENAUER, Cristina. **As tecnologias da informação e da comunicação e sua influência na mudança do perfil e da postura do professor.** Rio de Janeiro, 2004. Disponível em <http://www.ricesu.com.br/colabora/n6/artigos/n_6/id01.php>. Acesso em: 01 nov. 2004.

COX, Brad J. **Programação orientada para objeto.** São Paulo: Makron, McGraw-Hill, 1991.

DEITEL, H. M.; DEITEL, P. J. **Java, como programar.** Porto Alegre: Bookman, 2003.

FOWLER, Martin; SCOTT, Kendal. **UML essencial:** um breve guia para a linguagem-padrão de modelagem de objetos. Porto Alegre: Bookman, 2000.

GALHARDO, Mariane Fogaça; ZAINA, Luciana Aparecida Martinez. Simulação para Ensino de Conceitos da Orientação a Objetos. In: XI Seminários de Computação – SEMINCO, 2004, Blumenau. **Anais...** Blumenau, Universidade Regional de Blumenau – FURB, 2004.

GUBLER, André Iraldo; **Protótipo de um sistema especialista para auxiliar no ensino de algoritmos.** 2002. 66 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

KLOTZ, Gilson; **Protótipo de um sistema de apoio à escrita de redações.** 2002. 64 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

MATTOS, Mauro Marcelo. **Metodologia de desenvolvimento de sistemas orientados a objetos**. Março, 2003. Notas de aula.

_____. Ferramenta case para apoio a construção de abstrações em lógica de programação. In: II Simpósio de Informática do Planalto Médio – SIPM'2000, 5., 2000, Passo Fundo. **Anais...** Passo Fundo, Universidade de Passo Fundo, 2000.

MONTENEGRO, Fernando; PACHECO, Roberto. **Orientação a objetos em C++.** Rio de Janeiro: Ciência Moderna Ltda., 1994.

RICESU - REDES DE INSTITUIÇÕES CATÓLICAS DE ENSINO SUPERIOR. **Ramificações de Pesquisa da Inteligência Artificial**. Rio de Janeiro, [2003]. Disponível em: <http://www.ricesu.com.br/colabora/n8/artigos/n_8/id03c.htm>. Acesso em: 01 nov. 2004.

SILVA, Ildeu Moreira da. **Assimilação e acomodação dos conceitos de orientação a objetos:** um estudo de caso na PRODABEL. 2000. 160 f. Dissertação (Mestrado em Administração Pública) – Escola de Governo de Minas Gerais, Fundação João Pinheiro, Belo Horizonte.

SILVA, Carlos Alberto da. **Informática na educação.** 2000. 45 f. Monografia (Pós-graduação em tecnologias em desenvolvimento de sistemas) – Informática na educação, Universidade Regional de Blumenau, Canoinhas.

VALENTE, José Armando. **Diferentes uso do computador na educação.** Rio de Janeiro, [2003]. Disponível em: <http://www.educacaopublica.rj.gov.br/biblioteca/educacao/educ27b.htm>. Acesso em: 27 out. 2004.