UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

WEBIDE 2.0: UM AMBIENTE WEB USANDO GWT-EXT PARA ACOMPANHAMENTO E DESENVOLVIMENTO DE EXERCÍCIOS DE PROGRAMAÇÃO

RAFAEL ADRIANO

BLUMENAU
2010

2010/1-19

RAFAEL ADRIANO

WEBIDE 2.0: UM AMBIENTE WEB USANDO GWT-EXT PARA ACOMPANHAMENTO E DESENVOLVIMENTO DE EXERCÍCIOS DE PROGRAMAÇÃO

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. Adilson Vahldick, Mestre - Orientador

BLUMENAU
2010

2010/1-19

WEBIDE 2.0: UM AMBIENTE WEB USANDO GWT-EXT PARA ACOMPANHAMENTO E DESENVOLVIMENTO DE EXERCÍCIOS DE PROGRAMAÇÃO

Por

RAFAEL ADRIANO

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

Presidente:	Prof. Adilson Vahldick, M.Sc. – Orientador, FURB
Membro:	Prof. Marcel Hugo, M.Eng. – FURB
Membro:	Prof. Fernando dos Santos, M.Sc. – FURB

Blumenau, 6 de julho de 2010

Dedico este trabalho aos meus pais, a todos os amigos, à minha namorada, ao meu orientador e especialmente aqueles que me ajudaram diretamente na realização deste.

AGRADECIMENTOS

A Deus, pelo seu imenso amor e graça.

À minha família, que sempre esteve presente.

À minha namorada, Gabriela, pela paciência e apoio.

Ao meu amigo, Julio Gesser, que meu deu ótimas dicas.

Ao professor Fernando dos Santos, que me apoiou nos testes com sua turma.

Aos meus amigos, pelos empurrões e cobranças.

Ao meu orientador, Adilson Vahldick, por ter acreditado na conclusão deste trabalho.

Para mim, sábio não é aquele que proclama palavras de sabedoria, mas sim aquele que demonstra sabedoria em seus atos.

São Gregório

RESUMO

Este trabalho apresenta o desenvolvimento de um ambiente web para acompanhar e desenvolver exercícios de programação. O ambiente possui uma área administrativa, onde o professor cria exercícios para serem resolvidos pelos alunos. Neste ambiente o aluno resolve o exercício utilizando-se da sintaxe Java e de um editor com recursos *syntax highlighting*, *content assist*, numeração de linhas, execução e depuração do algoritmo. O ambiente também fornece uma área para professores e monitores corrigirem os exercícios dos alunos quando finalizados, podendo enviar comentários de correção e reabrir o exercício finalizado para cada aluno. O ambiente foi desenvolvido com as linguagens Java e JavaScript, e utilizou o *framework* GWT em conjunto com a biblioteca de extensão Ext para facilitar a criação de telas. O GWT também se utiliza de tecnologias como AJAX, JSNI (onde é possível escrever código JavaScript em um fonte Java), e o protocolo RPC (comunicação entre cliente e servidor). Foi usada a arquitetura JPDA para desenvolver os recursos de depuração do ambiente.

Palavras-chave: Ensino. GWT-Ext. JPDA. Aprendizado de programação. Ambiente de programação.

ABSTRACT

This paper presents the development of a web environment to monitor and develop programming exercises. The environment has an administrative area, where the teacher creates exercises to be solved by students. In this environment the student solves the exercise using the Java syntax and an editor with syntax highlighting, content assist, line numbering, running and debugging the algorithm. The environment also provides an area for teachers and monitors to correct exercises when the students finalized them, send comments to fix and reopen the exercise for each student. The environment was developed with Java and JavaScript, and used the GWT framework along with an Ext extension library to facilitate the creation of screens. GWT also uses technologies such as AJAX, JSNI (where you can write JavaScript code in a Java source), and protocol RPC (communication between client and server). Also the JPDA architecture was used to develop the debugging environment.

Key-words: Teaching. GWT-Ext. JPDA. Learning programming. Programming environment.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

AJAX – *Asynchronous Javascript And XML*

CSS – *Cascading Style Sheets*

DAO – *Data Access Object*

GWT – *Google Web Toolkit*

HTML – *HyperText Markup Language*

JDI – *Java Debug Interface*

JDK – *Java Development Kit*

JDWP – *Java Debug Wire Protocol*

JLS – *Java Language Specification*

JPDA – *Java Platform Debugger Architecture*

JRE – *Java Runtime Environment*

JSNI – *Java Script Native Interface*

JSON – *JavaScript Object Notation*

JVM – *Java Virtual Machine*

JVM TI – *Java Virtual Machine Tool Interface*

JVMS – *Java Virtual Machine Specification*

MVC – *Model View Controller*

RMI – *Remote Method Invocation*

RPC – *Remote Procedure Call*

UI – *User Interface*

UML – *Unified Modeling Language*

URL – *Uniform Resource Locator*

VM – *Virtual Machine*

XML - *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

A disciplina de programação é uma das disciplinas chaves na formação do profissional da computação. No entanto ela apresenta grande nível de dificuldade no processo de ensino/aprendizagem. Consequentemente a comunidade acadêmica tem se mostrado preocupada com o desempenho dos alunos nas disciplinas introdutórias de programação (XAVIER, 2004).

Para o sucesso no aprendizado de algoritmos exige-se um forte comprometimento por parte dos alunos em praticar os exercícios fornecidos pelo professor, assim como outras tarefas extras. É requerida também do professor uma atenção personalizada ao aluno para respeitar seu ritmo individual de aprendizado, seja corrigindo vários exercícios como complementando com seus comentários (VAHLDICK; LOHN, 2008).

Para acompanhar os alunos na sua desenvoltura nas disciplinas de programação, hoje se veem necessários softwares para auxiliar professores no progresso individual do aluno, pois segundo Branco e Schuvartz (2007), acadêmicos iniciantes, ao se depararem com a disciplina, sentem-se incapazes de programar, devido ao conjunto de habilidades que a programação exige como capacidade para solucionar problemas, raciocínio lógico, habilidade matemática, capacidade de abstração, entre outras.

Os Sistemas Tutores Inteligentes (STI) podem ajudar os professores neste processo, pois, segundo Moreira (2007), eles conseguem proporcionar o ensino individualizado criando caminhos para o aprendizado de acordo com as necessidades do estudante. Este tipo de sistema consegue automatizar a tarefa do professor ao corrigir as atividades dos alunos e indicar o próximo passo que o aluno deve seguir. O aluno desenvolve o algoritmo, que pode devolver a resposta desejada, mas a sua estrutura pode não ser a melhor para alcançar o resultado. Por consequência, tal tipo de avaliação ainda precisa ser efetuada por pessoas experientes.

Decorrente dos fatos acima expostos, e com intuito de melhorar o processo, foi proposto em Lohn (2008) um ambiente web, onde o aluno pode desenvolver os exercícios, acessando os enunciados no próprio ambiente, e o professor (ou monitor) acompanhar e comentar a resolução neste mesmo ambiente. Além de editar os programas, o professor, monitor e aluno também podem compilá-los, executá-los e depurá-los. Com esta possibilidade, o professor avalia o programa e devolve o resultado da avaliação para o aluno e este ciclo se repete até o aluno alcançar o objetivo do exercício.

O ambiente foi disponibilizado como aplicação web, por facilitar a interação com o usuário através de navegadores, dispensando a instalação do software e deixando mais flexível o local onde o aluno desenvolverá os exercícios.

O trabalho aqui apresentado é uma reimplementação baseando-se nos requisitos funcionais do trabalho de Lohn (2008), com intuito de desenvolver um ambiente com melhor usabilidade e robustez. Para isso a nova implementação usou o *framework*[1] GWT (GOOGLE, 2010a), junto com a biblioteca Ext JS (EXTJS, 2010) para a interface com o usuário e a arquitetura JPDA (SUN MICROSYSTEMS, 2010a) para depuração.

GWT é um *framework* AJAX, com o objetivo de facilitar o desenvolvimento de telas web em JavaScript, onde, através de código Java o compilador do GWT traduz para JavaScript (GOOGLE, 2010a). Além desta facilidade, a biblioteca do Ext JS deixa as telas com estilo de aplicação *desktop*. A interface atual do ambiente não era intuitiva. Além disso, faltavam alguns recursos básicos para um ambiente de programação, como número de linhas, *content assist* e *syntax highlighting*.

JPDA é uma arquitetura de depuração multicamadas que permite aos desenvolvedores de ferramentas criarem facilmente aplicações de depuração, portáveis em várias plataformas e implementações de máquina virtual (SUN MICROSYSTEMS, 2010a). A tecnologia de depuração utilizada pelo trabalho de Lohn (2008) é bastante precária, pois a comunicação era através de `Strings`, onde era preciso implementar um *parser* para obter as informações da depuração, deixando o recurso muito vulnerável a erros e a perda de performance, resultando em falhas neste processo por parte do usuário e limitando as informações apresentadas.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho foi desenvolver um ambiente web em GWT-Ext para que se possa codificar, depurar e executar exercícios de programação cadastrados pelo professor, e acompanhar os alunos individualmente, baseando-se nos mesmos requisitos funcionais de Lohn(2008).

Os objetivos específicos do trabalho são:

a) disponibilizar uma nova interface mais amigável usando o *framework* GWT-Ext;

[1] Segundo Uchôa e Melo (1999), *framework* é um (sub) sistema de software, de um domínio particular, que pode ser ajustado para aplicações individuais.

b) utilizar tecnologia de depuração com a arquitetura JPDA;
c) fornecer recursos de edição como número de linhas, auto complemento de código e *syntax highlighting*.

1.2 ESTRUTURA DO TRABALHO

A estrutura deste trabalho está dividida em capítulos que serão explicados a seguir.

O primeiro capítulo apresenta a contextualização, objetivos e justificativa para o desenvolvimento do trabalho.

O segundo capítulo apresenta a fundamentação teórica do trabalho, problemas quanto ao aprendizado de algoritmos, apresentação do *framework* GWT, da biblioteca EXT JS, da API Gwt-Ext, das ferramentas da plataforma Java, da especificação JPDA e por fim, a apresentação dos trabalhos correlatos.

O terceiro capítulo apresenta os requisitos da ferramenta, bem como sua especificação, implementação e resultados. São mostrados diagramas de casos de uso, diagrama de atividades e diagrama de classes. Também é apresentada a operacionalidade do sistema.

O quarto capítulo apresenta as conclusões, limitações e sugestões para extensões futuras.

2 FUNDAMENTAÇÃO TEÓRICA

A seguir são explanados os principais assuntos relacionados ao desenvolvimento do trabalho: aprendizado de algoritmos, GWT, EXT JS, Gwt-Ext, JPDA, ferramentas da plataforma Java e os trabalhos correlatos.

2.1 APRENDIZADOS DE ALGORITMOS

As disciplinas relacionadas ao ensino de algoritmos e programação de computadores na área de computação e informática em um curso de graduação são disciplinas importantes para alunos que se formarão na área de desenvolvimento de software (SOUTO e DUDUCHI, 2009).

Segundo Gondim, Ambrósio e Costa (2009), ensinar programação básica ao aluno no ensino superior é um dos grandes desafios na área de computação. Inicialmente o aluno tem a necessidade de desenvolver o raciocínio lógico/algorítmico que não possui, além de ter que conhecer uma nova tecnologia e uma linguagem desconhecida.

Conforme Hinterholz Jr. (2009), os pesquisadores da área da educação precisam oferecer alternativas para diminuir a evasão quanto a repetência nessa disciplina, e assim melhorando a qualidade na formação do acadêmico nos cursos de computação em nível de graduação.

Com uma ferramenta que auxiliasse os professores a identificar alunos com dificuldades em raciocínio lógico e resolução de problemas, facilitaria acompanhar os alunos de forma individual com atividades específicas, tendo como resultado a redução de repetência nas disciplinas de programação básica e desistência nos cursos de computação (SOUTO e DUDUCHI, 2009).

2.2 GOOGLE WEB TOOLKIT (GWT)

Criar aplicativos para a web era um processo tedioso e com alta incidência de erros. Os

desenvolvedores podem passar 90% do tempo trabalhando para contornar peculiaridades do navegador. Além disso, a criação, a reutilização e a manutenção de grandes bases de código JavaScript e componentes AJAX pode ser difícil e delicada. O GWT facilita esse processo, permitindo que os desenvolvedores criem rapidamente e mantenham aplicativos *front-end* JavaScript complexos e de alto desempenho na linguagem de programação Java (GOOGLE, 2010a).

2.2.1 Comunicação

O GWT suporta um conjunto de protocolos de transferência, como JSON e XML, mas também tem outro protocolo de comunicação específico, o GWT RPC. Este protocolo trabalha de forma semelhante ao tradicional Java RMI, basta criar uma interface e especificar os métodos remotos que se desejam ser chamados. Quando é chamado um método remoto pelo cliente, o GWT RPC serializa automaticamente os argumentos, invoca o método adequado no servidor e decodifica o valor de retorno para o código cliente. O GWT RPC também pode lidar com hierarquias de classe polimórficas e até mesmo transmitir as exceções pela rede (GOOGLE, 2010a).

2.2.2 *Java Script Native Interface* (JSNI)

Segundo Google (2010a), o GWT disponibiliza classes como `JavaScriptObject`, para que seja possível modelar objetos JavaScript como os tipos do Java, sem memória adicional ou velocidade elevada.

Métodos JSNI são declarados nativamente e contêm código JavaScript em um bloco de comentários especialmente formatados entre o final da lista de parâmetros e o ponto e vírgula. Um bloco de comentário JSNI começa com `/*-{` e termina com `}-*/`. Métodos JSNI são chamados, como qualquer método Java normal. Eles podem ser estáticos ou métodos de instância (GOOGLE, 2010b).

O Quadro 1 apresenta um código exemplo do JSNI, que retorna a posição do cursor do `body` de um `iframe`.

```
public native JavaScriptObject getCaretPosition() /*-{
      winFrame = field.iframe.contentWindow;
      var result = {offset: 0};
      var cursor = select.cursorPos(winFrame.document.body, true);
      if (cursor) {
            result.offset = cursor.offset;
      }
      return result;
}-*/;
```

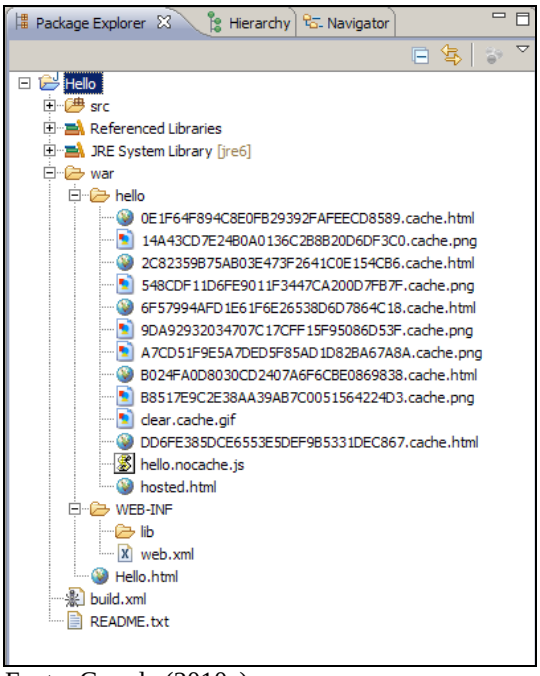
Quadro 1 – Código exemplo do JSNI

2.2.3 Compilador

Segundo Google (2010c), GWT tem um compilador que converte código Java em JavaScript, transformando o código de aplicação Java em um aplicativo JavaScript equivalente.

O compilador GWT suporta a grande maioria da sintaxe da linguagem Java. A biblioteca *runtime* do GWT emula um subconjunto relevante da biblioteca *runtime* do Java. Se uma classe JRE ou método não é suportado, o compilador irá emitir um erro (GOOGLE, 2010c).

A Figura 1 mostra os arquivos gerados após a compilação do compilador GWT de um módulo chamado `Hello`.



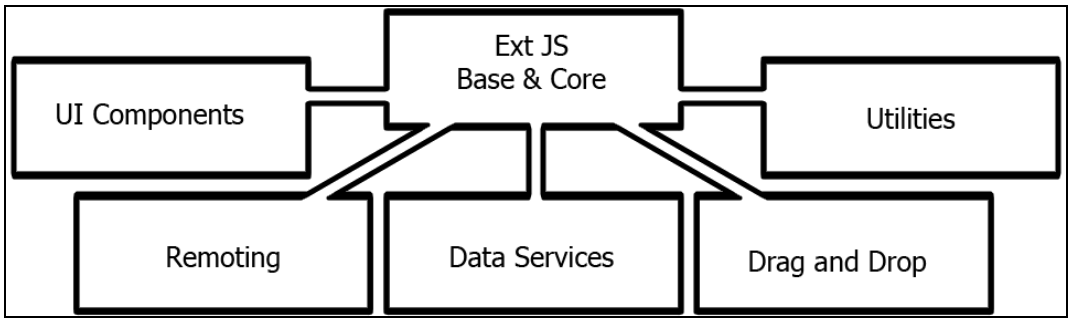
Fonte: Google (2010c).

Figura 1 – Arquivos gerados após a compilação pelo compilador GWT

2.3 EXT JS

Ext JS é uma biblioteca JavaScript para criar aplicações ricas para internet, com alto desempenho, *widgets*[2], UI personalizáveis, bom *design* e modelo de componente extensível, com uma API intuitiva e fácil de usar (EXT JS, 2010).

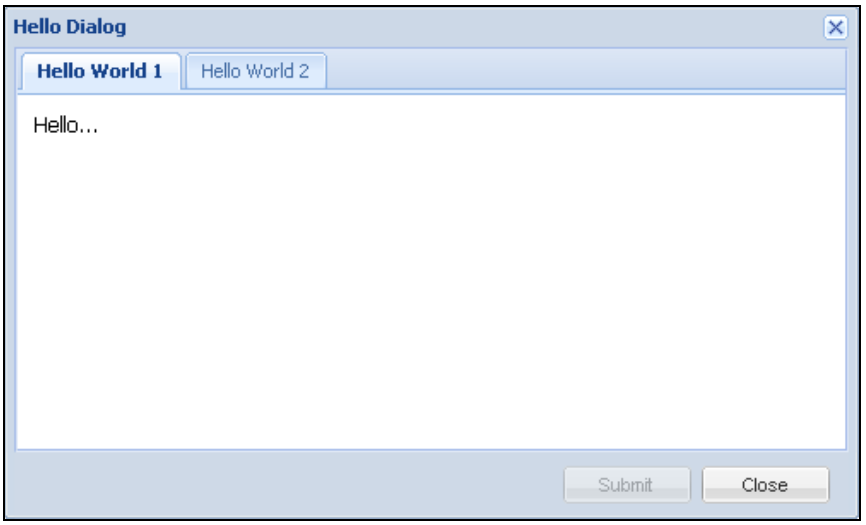
Segundo Garcia Jr. (2010, p. 6), Ext JS é uma estrutura que não fornece somente componentes UI, mas contém também uma série de outras funcionalidades. Estes podem ser divididos em seis grandes áreas: `Core`, `UI Components`, `Data Services`, `Drag and Drop` e utilidades gerais (Figura 2).



Fonte: Garcia Jr. (2010, p. 6).

Figura 2 – Diagrama das seis áreas do Ext JS

Na Figura 3 é mostrado um exemplo de tela gerado pelo Ext JS.



Fonte: Ext JS (2010).

Figura 3 – Tela gerado pelo Ext JS

[2] Segundo Deitel e Deitel (2009), *widgets*, também referenciados como *gadgets*, são miniaplicações projetadas para serem executadas como aplicações individuais ou *add-on* (adicionáveis) em páginas Web.

2.4 GWT-EXT

Gwt-Ext é uma API que ajuda a unir a funcionalidade do GWT e os *widgets* disponíveis na biblioteca Ext JS (JIVAN, 2008).

Segundo Jivan (2008), a configuração do Gwt-Ext é simples, somente precisando adicionar o arquivo `gwtext.jar` no *classpath* do projeto, e colocar a linha `<inherits name='com.gwtext.GwtExt'/>` no XML de configuração do GWT, conforme Quadro 2.

```
<module>

  <inherits name='com.google.gwt.user.User'/>
  <inherits name='com.gwtext.GwtExt'/>

  <entry-point class='com.mycompany.mypackage.client.HelloWorld'/>

</module>
```

Fonte: Jivan (2008).

Quadro 2 –XML de configuração do GWT

Após a configuração do arquivo XML, é preciso referenciar os fontes JavaScript e o estilos CSS da biblioteca Ext JS na página HTML principal, conforme Quadro 3.

```
<html>
  <head>
    ...

    <link rel="stylesheet" href="js/resources/css/ext-all.css"/>
    ...
    <script language="javascript" src="js/adapter/ext/ext-
base.js"></script>
      <script language="javascript" src="js/ext-all.js"></script>
    ...
</html>
```

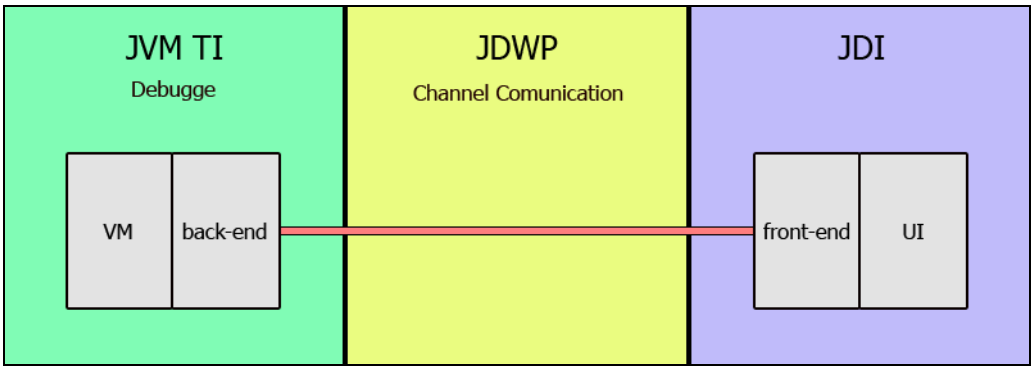
Fonte: Jivan (2008).

Quadro 3 – Página HTML principal

2.5 JAVA PLATFORM DEBUGGER ARCHITECTURE (JPDA)

Nesta seção é apresentada a arquitetura de depuração especificada pela Sun Microsystems. Na seção 2.5.1 é mostrada a camada *Java Virtual Machine Tool Interface* (JVM TI). Na seção 2.5.2 é vista a camada *Java Debug Wire Protocol* (JDWP) da arquitetura e por fim na seção 2.5.3 a última camada da arquitetura, o *Java Debug Interface* (JDI).

Segundo Sun Microsystems (2010a), JPDA é uma arquitetura multicamadas de depuração que permite aos desenvolvedores criarem facilmente aplicativos que utilizam um depurador. Essa arquitetura consiste em três camadas conforme Figura 4. Cada camada é explicada nas seções subsequentes.



Fonte: Sun Microsystems (2010a).

Figura 4 – Estrutura das camadas compostas pelo JPDA

2.5.1 *Java Virtual Machine Tool Interface* (JVM TI)

Segundo Sun Microsystems (2010a), JVM TI é uma interface de programação nativa, que define serviços como, requisição de informação (por exemplo, o estado atual do *stackframe*[3]), ações (por exemplo, definir um *breakpoint*[4]), e notificação (por exemplo, quando um *breakpoint* foi atingido). A interface permite que qualquer implementação de uma VM possa ser facilmente plugada na arquitetura de depuração.

Segundo Sun Microsystems (2010a), o *back-end*[5] é responsável pela comunicação de requisições entre o *front-end*[6] e a VM, e também pela comunicação de respostas das requisições (incluindo os eventos desejados) da VM para o *front-end*. O *back-end* se comunica com o *front-end* através de um canal de comunicação utilizando o JDWP. Além disso, o *back-end* se comunica com a VM usando o JVM TI.

[3] *Stackframe* é a pilha de chamadas de subrotinas em um programa de computador durante uma depuração (CALL, 2010).
[4] *Breakpoint* é um ponto de interrupção em um processo de computador durante uma depuração (BREAKPOINT, 2010).
[5] *Back-end* é um termo generalizado que se refere à etapa final de um processo (FRONT-END, 2010).
[6] *Front-end* é um termo generalizado que se refere à etapa inicial de um processo (FRONT-END, 2010).

2.5.2 *Java Debug Wire Protocol* (JDWP)

Essa camada define o formato das informações e requisições de transferência entre o processo depurado e o depurador *front-end*. Ela não define o mecanismo de transporte (*socket*, serial, memória compartilhada, etc). A especificação do protocolo permite que o processo depurado e o depurador *front-end* executem em implementações de VMs separadas e/ou em plataformas separadas. E também permite que o *front-end* possa ser escrito em outra linguagem diferente do Java (SUN MICROSYSTEMS, 2010a).

2.5.3 *Java Debug Interface* (JDI)

Segundo Sun Microsystems (2010a) a camada JDI define informações e requisições em código de alto nível. A interface é 100% Java e é implementada pelo *front-end*. Enquanto implementações de depuradores poderiam utilizar diretamente o JDWP ou JVM TI, esta interface facilita muito a integração do depurador com o ambiente de desenvolvimento.

O *front-end* implementa a interface JDI e abstrai as informações de baixo nível que vem da camada JDWP, oferecendo informações mais alto nível (SUN MICROSYSTEMS, 2010a).

A UI não é especificada, deixando que desenvolvedores dos ambientes de depuração proporcionem um valor acrescentado na ferramenta (SUN MICROSYSTEMS, 2010a).

2.6 FERRAMENTAS DA PLATAFORMA JAVA

Nesta seção serão apresentadas as ferramentas que acompanham a plataforma Java. Na seção 2.6.1 será apresentada a ferramenta responsável por compilar fontes Java, e na seção 2.6.2 será mostrada a ferramenta responsável por executar fontes Java compilados.

2.6.1 Compilação

Segundo Sun Microsystems (2010b) o compilador oficial para linguagem de programação Java é o `javac`. Ele é um programa executável que funciona através de linhas de comando.

Como entrada, o compilador `javac` somente aceita código definido pela *Java Language Specification* (JLS). Depois de compilar, é gerado um ou mais arquivos *bytecodes*. O arquivo *bytecode* tem uma estrutura definida pela *Java Virtual Machine Specification* (JVMS), e possui um conjunto de instruções que podem ser interpretadas pela JVM (SUN MICROSYSTEMS, 2010b).

Na linha de comando, é possível incluir argumentos, para executar determinadas ações quando o código fonte for compilado. Dentre esses argumentos, somente serão citados três, que serão usados para o desenvolvimento deste trabalho.

No Quadro 4 é apresentada a linha de comando para compilar com os argumentos que serão usados neste trabalho.

```
javac –g –cp json.jar; WebIde.java Exercicio.java
```

Quadro 4 – Exemplo da linha de comando usada pelo compilador Javac

Segundo Sun Microsystems (2010b), o argumento `"-g"` é necessário para gerar informações que serão usadas posteriormente pelo processo de depuração. O argumento `"-cp"` é usado para indicar todas as classes e bibliotecas necessárias para que a compilação execute com sucesso. Os argumentos `"WebIde.java"` e `"Exercicio.java"` são as classes que serão compiladas pelo comando.

2.6.2 Execução

A execução de código fonte Java, é feita a partir da linha de comando através do programa executável `java`, que segundo Sun Microsystems (2010b), faz isso a partir do ambiente *runtime* Java, invocando o método `main` da classe passada como parâmetro. O método declarado deve ser de visibilidade pública, estática, não deve retornar nenhum valor e deve aceitar como argumento um *array* de String, conforme o Quadro 5.

```
public static void main(String[] args) {...}
```

Quadro 5 – Método que será invocado quando executado um programa Java

A ferramenta `java` executa o arquivo *bytecode* gerado na compilação, interpretando o código e gerando instruções para o processador (SUN MICROSYSTEMS, 2010a).

No Quadro 6 é mostrada a linha de comando usada para executar um código fonte compilado na ferramenta `java`,

```
java –cp C:\Exercicio;json.jar Exercicio
```

Quadro 6 – Exemplo da linha de comando usado pela ferramenta Java

2.7 TRABALHOS CORRELATOS

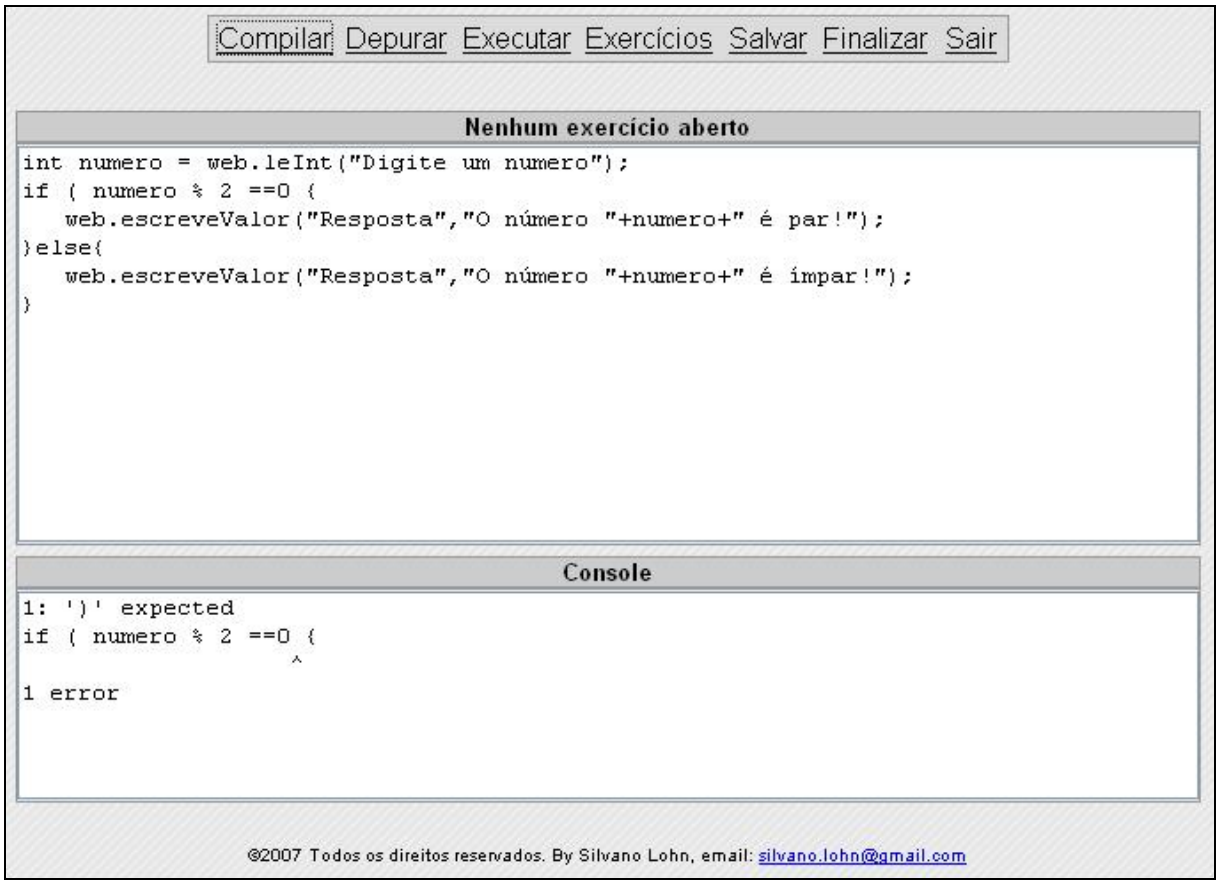
Existem alguns sistemas semelhantes ao trabalho proposto. Dentre eles foram escolhidos três cujas características enquadram-se nos principais objetivos deste trabalho. Foram selecionados os seguintes projetos: “Ambiente na Web para Execução e Depuração de Programas com Sintaxe Java” (LOHN, 2008), “Ferramenta Computacional de Apoio ao Processo de Ensino-Aprendizagem dos Fundamentos de Programação de Computadores” (BRANCO; SCHUVARTZ, 2007) e “Tepequém: uma nova Ferramenta para o Ensino de Algoritmos nos Cursos Superiores em Computação” (HINTERHOLZ JR, 2009).

2.7.1 Ambiente na Web para Execução e Depuração de Programas com Sintaxe Java

O objetivo do trabalho de Lohn (2008) é fazer com que os professores ou monitores tenham um acompanhamento individualizado e personalizado de cada aluno na disciplina introdutória de programação. Para isso, desenvolveu-se um ambiente web para codificação de algoritmos na linguagem Java. Este ambiente oferece:

a) edição de código Java;
b) compilação, execução e depuração do código;
c) sistema para visualização de exercícios e comentários do professor ou monitor sobre os mesmos.

A Figura 5 apresenta a tela de edição do ambiente, o qual foi desenvolvido com *servlets* e AJAX.



Fonte: Lohn (2008, p. 72).

Figura 5 – Ambiente de programação

2.7.2 Ferramenta Computacional de Apoio ao Processo de Ensino-Aprendizagem dos Fundamentos de Programação de Computadores

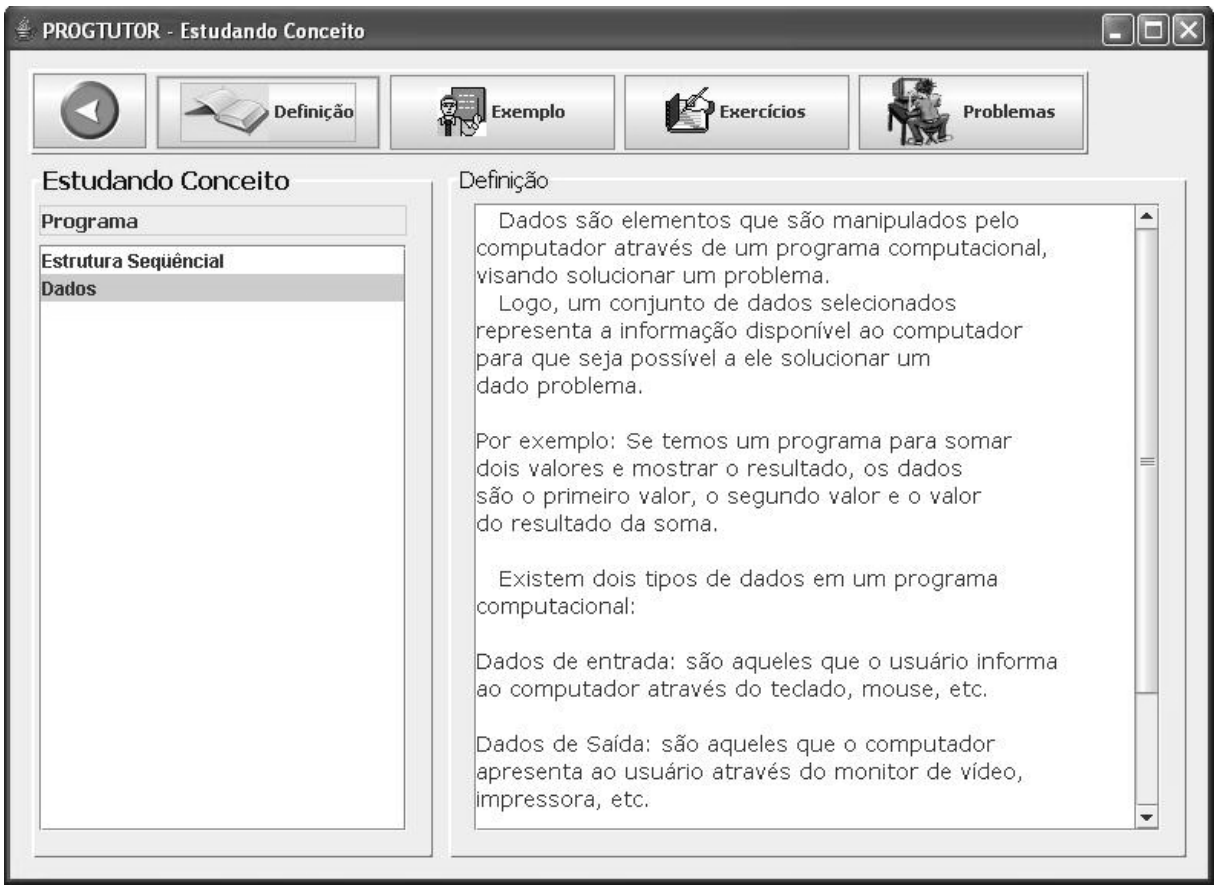
A ferramenta de Branco e Schuvartz (2007) tem por objetivo servir de apoio ao processo de ensino-aprendizagem dos fundamentos de programação de computadores, utilizando-se de técnicas de Inteligência Artificial (IA), é dividida em três módulos, sendo eles:

a) módulo domínio: possui o conteúdo relacionado aos fundamentos de programação de computadores e é representado pelas entidades `Conceitos`, `Passos`, `Situações`, `Imagens`, `Exercícios` e `Alternativas` presentes no modelo conceitual da base de dados do Sistemas Tutores Inteligentes (STI);
b) módulo aluno: visa armazenar informações sobre os usuários, permitindo que eles sejam identificados pelo sistema e recebam tratamento adequado de acordo com seu nível de conhecimento. Este módulo está fortemente ligado ao Módulo

Domínio e, em conjunto com ele, serve de base para que o Módulo Tutor possa determinar qual estratégia de ensino aplicar a cada estudante;

c) módulo tutor: aplica a estratégia de ensino mais adequada a cada usuário do STI. Para tanto, este módulo deve trabalhar em conjunto com os demais módulos, `Aluno` e `Domínio`, que retêm informações úteis sobre os usuários e sobre o conteúdo que está sendo ensinado pelo sistema respectivamente.

Utiliza-se de uma interface (Figura 6) para que o aluno possa acessar os conceitos e exemplos, corrigindo exercícios e problemas, verificando o nível de conhecimento de cada estudante e direcionando-o pelos conteúdos durante o processo de ensino-aprendizagem de acordo com seu desempenho.



Fonte: Branco e Schuvartz (2007).

Figura 6 – Ambiente da ferramenta

2.7.3 Um Ambiente para Ensino de Programação com *Feedback* Automático de Exercícios

O trabalho de Moreira e Favero (2009) tem por objetivo estudar técnicas de avaliação automática de algoritmos que possibilitam um retorno imediato ao estudante auxiliando o professor na correção e avaliação de exercícios de programação.

Segundo Moreira e Favero (2009) foram utilizadas duas técnicas, sendo elas:

a) regressão linear múltipla: essa técnica pode predizer um conceito numérico a partir de diversos indicadores que medem diferentes aspectos dos objetos sendo avaliados;
b) n-gramas: que consiste em extrair cadeias de caracteres (n-grama) de uma cadeia maior. Cada n-grama pode assumir um valor como unigrama, bigrama ou trigrama. Combinando medidas de similaridade vetorial, é possível comparar até textos livres. E comparando duas cadeias de caracteres, pode-se medir quantos n-gramas elas compartilham, determinando quanto dos dois textos são similares.

No estudo de caso de Moreira e Favero (2009), foram usadas duas abordagens para a predição da nota do aluno: a complexidade do programa, utilizando a técnica de regressão linear múltiplas com métricas de engenharia de software. E a medida de similaridade estrutural, utilizando a técnica de regressão linear múltiplas com n-gramas como indicadores.

3 DESENVOLVIMENTO

Neste capítulo são apresentados os requisitos, a especificação e a implementação do trabalho. Também são mostrados diagramas de atividades, diagramas de classe e o modelo de entidades e relacionamentos. Para finalizar é apresentada uma validação do trabalho utilizando códigos fonte de exemplos, com os resultados obtidos e uma comparação com os trabalhos correlatos.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Analisando a proposta do ambiente web, foram resgatados do trabalho de Lohn (2008) os seguintes Requisitos Funcionais (RF):

a) permitir que o programa seja carregado, editado e salvo (RF);
b) fazer a execução do programa (RF);
c) fazer a depuração do programa de modo passo-a-passo informando a linha e o valor das variáveis em execução (RF);
d) fazer a compilação do programa informando quando houver os erros de compilação (RF);
e) fazer um ambiente administrativo onde seja possível criar e corrigir os exercícios (RF);
f) permitir que os usuários cadastrem-se no ambiente (RF);

Os novos Requisitos Funcionais e Requisitos Não Funcionais (RNF) são:

a) permitir o auto-complemento de código (RF);
b) fazer o *syntax highlighting* no código (RF);
c) apresentar numeração de linhas (RF);
d) utilizar a arquitetura JPDA para a depuração (RNF);
e) ser implementado utilizando o *framework* GWT-Ext (RNF);
f) permitir programação monobloco sem divisão de métodos (RNF);
g) ser compatível com os principais navegadores comerciais (RNF).

3.2 ESPECIFICAÇÃO

A especificação do sistema foi feita através de diagramas UML e MER, tendo como ferramenta de auxílio o Enterprise Architect e MySQL Workbench.

3.2.1 Diagrama de Casos de Uso

Os diagramas de casos de uso fornecem um modo de descrever a visão externa do sistema e suas interações com o mundo exterior, representando uma visão de alto nível de funcionalidade intencional mediante o recebimento de um tipo de requisição de usuário (FURLAN, 1998, p. 169).

Para Furlan (1998, p. 171), cada cenário é uma história única de realização, relaciona uma sequência de interações entre atores e o sistema com as decisões definidas.

A Figura 7 apresenta o diagrama de casos de uso do sistema.

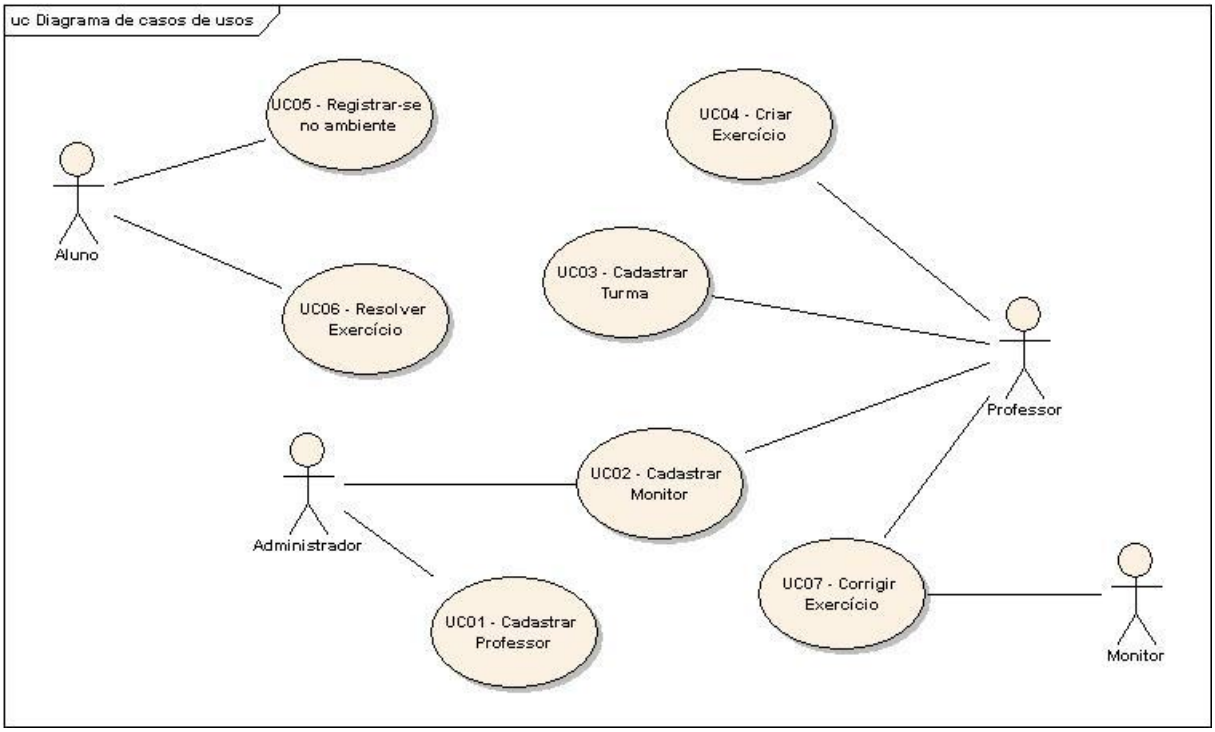


Figura 7 – Diagrama de casos de uso

No Quadro 7 é apresentado o cenário para o caso de uso cadastrar professor.

UC01 – Cadastrar Professor

1. **Descrição:** O administrador cadastra um professor no sistema.
   - **1.1. Atores:** Administrador (Principal).
     Professor (Secundário).
   - **1.2. Precondições:** O administrador deve estar autenticado no sistema.
   - **1.3. Fluxo de eventos:**
     - 1.3.1. Fluxo Básico – Cadastrar Professor.
       a. O administrador clica em Usuários.
       b. O sistema abre a tela com todos os usuários cadastrados.
       c. O administrador clica em Novo Professor/Monitor.
       d. O sistema abre a tela de cadastro.
       e. O administrador preenche os campos nome, usuário, senha, confirmar senha, tipo e email.
       f. O administrador clica em Salvar.
       g. O sistema registra o professor no banco de dados.
   - **1.4. Pós-Condição:** O professor está registrado no sistema.

Quadro 7 – Cenário do caso de uso cadastrar professor

No Quadro 8 é apresentado o cenário para o caso de uso cadastrar monitor.

UC02 – Cadastrar Monitor

1. **Descrição:** O professor cadastra um monitor no sistema.
   - **1.1. Atores:** Professor (Principal).
     Monitor (Secundário).
   - **1.2. Precondições:** O professor deve estar autenticado no sistema.
   - **1.3. Fluxo de eventos:**
     - 1.3.1. Fluxo Básico – Cadastrar Monitor.
       a. O professor clica em Usuários.
       b. O sistema abre a tela de com todos os usuários cadastrados.
       c. O professor clica em Novo Monitor.
       d. O sistema abre a tela de cadastro.
       e. O professor preenche os campos nome, usuário, senha, confirmar senha, tipo e email.
       f. O professor clica em Salvar.
       g. O sistema registra o monitor no banco de dados.
   - **1.4. Pós-Condição:** O monitor está registrado no sistema.

Quadro 8 – Cenário do caso de uso cadastrar monitor

No Quadro 9 é apresentado o cenário para o caso de uso cadastrar turma.

UC03 – Cadastrar Turma

1. **Descrição:** O professor cadastra uma turma no sistema.
   - **1.1. Atores:** Professor (Principal).
   - **1.2. Precondições:** O professor deve estar autenticado no sistema.
   - **1.3. Fluxo de eventos:**
     - 1.3.1. Fluxo Básico – Cadastrar Turma.
       a. O professor clica em Nova Turma.
       b. O sistema abre a tela de cadastro.
       c. O professor preenche os campos nome, professor e monitor.
       d. O professor clica em Salvar.
       e. O sistema registra a turma no banco de dados.
   - **1.4. Pós-Condição:** A turma está registrada no sistema.

Quadro 9 – Cenário do caso de uso cadastrar turma

No Quadro 10 é apresentado o cenário para o caso de uso criar exercício.

UC04 – Criar Exercício

1. **Descrição:** O professor cria um exercício para a turma no sistema.
   - **1.1. Atores:** Professor (Principal).
   - **1.2. Precondições:** O professor deve estar autenticado no sistema.
     Deve ter pelo menos uma turma cadastrada.
   - **1.3. Fluxo de eventos:**
     - 1.3.1. Fluxo Básico – Criar Exercício.
       a. O professor clica em Exercícios.
       b. O sistema abre a tela com todos os exercícios cadastrados.
       c. O professor clica em Novo.
       d. O sistema abre a tela de cadastro.
       e. O professor preenche os campos título, linguagem, turma e descrição.
       f. O professor clica em Salvar.
       g. O sistema registra o exercício no banco de dados.
       h. O sistema atribui o exercício para todos os alunos que pertencem a turma.
   - **1.4. Pós-Condição:** O exercício está registrado no sistema.

Quadro 10 – Cenário do caso de uso criar exercício

No Quadro 11 é apresentado o cenário para o caso de uso registrar-se no ambiente.

UC05 – Registrar-se no ambiente

1. **Descrição:** O aluno registra-se no ambiente.
   - **1.1. Atores:** Aluno (Principal).
   - **1.2. Precondições:** Deve ter pelo menos uma turma cadastrada.
   - **1.3. Fluxo de eventos:**
     - 1.3.1. Fluxo Básico – Registrar-se no ambiente.
       a. O aluno clica em Sou Novo.
       b. O sistema abre a tela de cadastro.
       c. O aluno preenche os campos nome, turma, usuário, senha, confirmar senha e email.
       d. O aluno clica em salvar.
       e. O sistema registra o aluno no banco de dados.
       f. O sistema atribui todos os exercícios não finalizados da turma para o aluno.
   - **1.4. Pós-Condição:** O aluno está registrado no sistema.

Quadro 11 – Cenário do caso de uso registrar-se no ambiente

No Quadro 12 é apresentado o cenário para o caso de uso resolver exercício.

UC06 – Resolver Exercício

1. **Descrição:** O aluno deseja resolver o exercício.
   - **1.1. Atores:** Aluno (Principal).
   - **1.2. Precondições:** O aluno deve estar autenticado no sistema.
   - **1.3. Fluxo de eventos:**
     - 1.3.1. Fluxo Básico – Resolver Exercício.
       a. O sistema mostra uma árvore com os exercícios finalizados e não finalizados.
       b. O aluno escolhe um exercício para resolver clicando no nodo da árvore onde o exercício se encontra.
       c. O sistema abre o editor.
       d. O aluno desenvolve a solução do exercício.
       e. O aluno finaliza o exercício clicando em Submeter.
       f. O sistema altera o estado do exercício do aluno para finalizado.
     - 1.3.2. Fluxos Alternativos
       a. Executar o programa
          i. O aluno clica em Executar.
          ii. O sistema compila o programa.
          iii. O sistema inicia a execução do programa.
          iv. O sistema abre a tela de execução.
          v. O aluno realiza a entrada de dados caso necessitar.
          vi. O sistema apresenta a saída de dados caso existir.
          vii. O sistema termina a execução do programa.
       b. Depurar o programa
          i. O aluno clica em Depurar.
          ii. O sistema compila o programa.
          iii. O sistema inicia a depuração do programa.
          iv. O sistema abre a tela de depuração.
          v. O aluno acompanha passo-a-passo a execução do programa clicando em Executar Linha.
          vi. O aluno realiza a entrada de dados caso necessitar.
          vii. O sistema apresenta a saída de dados caso existir.
          viii. O sistema apresenta os valores das variáveis usadas em seu programa.
          ix. O aluno termina a depuração do programa.
       c. Salvar o programa.
          i. O aluno clica em Salvar.
          ii. O sistema salva o programa no banco de dados.
   - **1.2. Pós-Condição:** O exercício terá seu estado como finalizado.

Quadro 12 – Cenário do caso de uso resolver exercício

No Quadro 13 é apresentado o cenário para o caso de uso corrigir exercício.

UC07 – Corrigir Exercício

1. **Descrição:** O professor ou monitor querem corrigir o exercício finalizado do aluno. Caso a solução do exercício do aluno esteja incorreta, o professor ou monitor podem escrever um comentário ao aluno descrevendo os erros cometidos ou dar dicas sobre a solução, e reabrir o exercício para o aluno corrigir.
   - **1.1. Atores:** Professor (Principal).
     Monitor (Principal).
   - **1.2. Precondições:** O professor ou monitor devem estar autenticado no sistema.
     Deve ter pelo menos um exercício finalizado pelo aluno em alguma turma do professor ou monitor.
   - **1.3. Fluxo de eventos:**
     - 1.3.1. Fluxo Básico – Corrigir Exercício.
       a. O sistema mostra uma árvore com os exercícios finalizados pelos alunos.
       b. O professor ou monitor escolhem um exercício para corrigir, clicando no nodo onde o exercício se encontra.
       c. O sistema abre o exercício no editor.
       d. O professor ou monitor podem corrigir erros de lógica ou compilação.
       e. O professor ou monitor executam o programa do aluno para testá-lo.
     - 1.3.2. Fluxos Alternativos
       a. Enviar Comentário
          i. No item c do fluxo básico, caso a solução do exercício não esteja correta, o professor ou monitor clica em Comentários do Exercício.
          ii. O sistema abre a tela de cadastro de Comentário.
          iii. O professor ou monitor preenche o campo Novo Comentário.
          iv. O professor ou monitor clica em Salvar.
          v. O sistema registra o comentário para o exercício do aluno.
       b. Reabrir Exercício
          i. No item c do fluxo básico, caso a solução do exercício não esteja correta, o professor ou monitor clica em Reabrir o exercício.
          ii. O sistema altera o estado do exercício para não finalizado.
   - **1.4. Pós-Condições:** Exercício do aluno estará corrigido.
     Foi registrado um comentário no exercício do aluno.

Quadro 13 – Cenário do caso de uso corrigir exercício

3.2.2 Diagrama de atividades

Segundo Furlan (1998, p. 221), um diagrama de atividade permite escolher a ordem pela qual as coisas devem ser feitas, isto é, indica meramente as regras essenciais de sequência que necessitam ser seguidas.

A Figura 8 apresenta o diagrama de atividades para indicar a sequência de utilização

do ambiente pelo professor. O professor começa entrando no sistema, onde ele pode optar pelas opções:

a) criar uma turma;
b) criar um monitor;
c) criar um exercício;
d) visualizar os exercícios finalizados pelos alunos;
e) ou fechar o sistema.

No caso de criar uma turma, um monitor ou um exercício, será aberto uma janela onde o professor irá preencher os dados, e após isso, o sistema irá salvar no banco de dados. No caso de criar um exercício, após salvar no banco de dados, o sistema irá distribuir o exercício criado para todos os alunos da turma.

Na opção para visualizar os exercícios finalizados pelo aluno, o professor irá expandir a árvore de turmas, onde constam os alunos e seus exercícios finalizados. Caso existir um exercício finalizado, o professor abre o exercício, e terá as alternativas para executar o programa, depurar ou corrigi-lo.

Se ele executar ou depurar o programa do aluno, o sistema irá gerar os arquivos fontes no disco do servidor e compilá-los. Caso ocorra erro na compilação, a execução ou depuração é abortada. Se não ocorrer erro de compilação, o sistema executa o processo do programa e gerencia o processo até ele ser finalizado com erro ou com sucesso.

Caso o professor corrigir o exercício do aluno, e se estiver com algum erro de lógica, o professor poderá criar um comentário para o aluno. Neste caso será aberta uma janela para preencher os dados do comentário, e depois o sistema salvará no banco de dados. Criando ou não o comentário, o professor pode reabrir o exercício para o aluno, onde o sistema altera o estado do exercício para não finalizado e envia um email para o aluno informando que seu exercício foi reaberto.

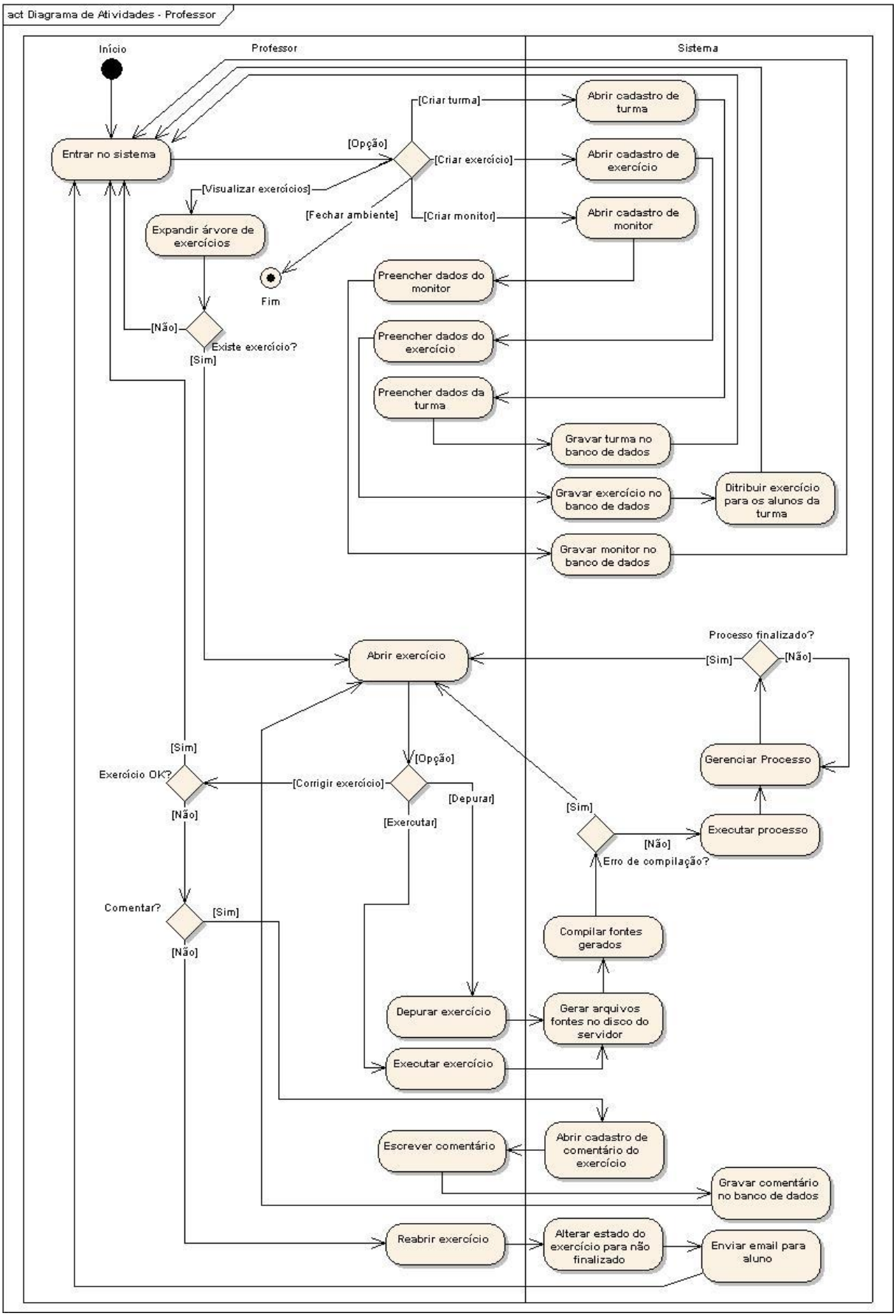


Figura 8 – Diagrama de atividades do professor

A Figura 9 apresenta o diagrama de atividades onde o monitor utiliza o ambiente. Diferentemente do professor, o monitor não poderá criar turma, monitor ou exercício. Somente poderá executar, depurar e corrigir o exercício do aluno, igual ao processo do professor.

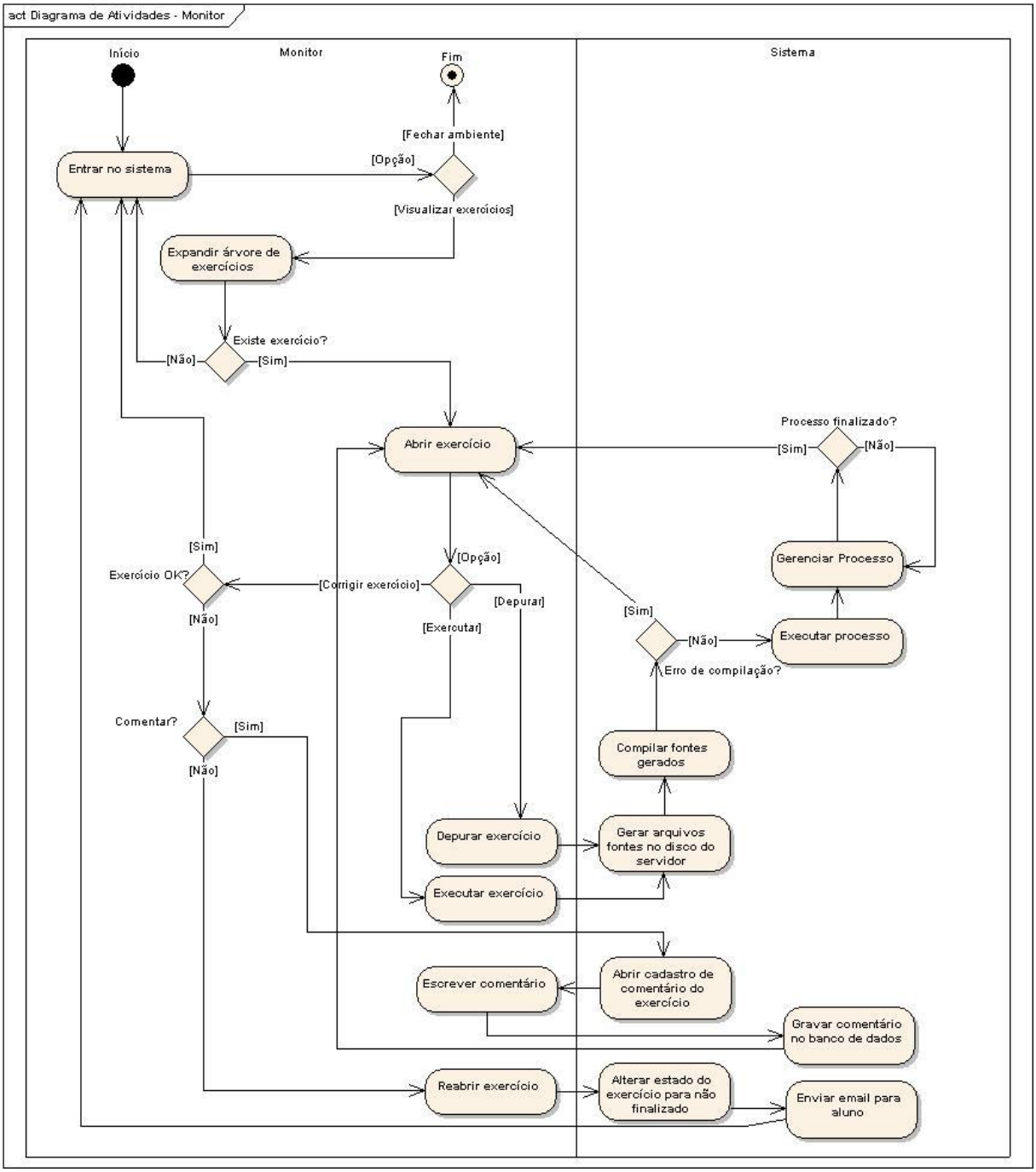


Figura 9 – Diagrama de atividades do monitor

A Figura 10 apresenta o diagrama de atividades onde o aluno utiliza o ambiente. O aluno é obrigado a registrar-se no sistema para utilizar o ambiente. Se ele não estiver registrado, poderá abrir o formulário de cadastro do aluno, e preencher os campos nome, usuário, senha, turma e email. Depois de efetuar o registro, o aluno poderá entrar no ambiente.

No ambiente o aluno tem duas escolhas, ele poderá sair do ambiente ou visualizar seus exercícios abertos expandindo a árvore de exercícios. Se existir um exercício em aberto, ele abre o exercício para desenvolver o programa com a intenção de resolver o exercício. Durante o desenvolvimento do programa, o aluno poderá executar, depurar, atualizar sintaxe (*syntax highlighting*), executar o assistente de conteúdo (*content assist*), salvar ou finalizar o exercício.

Enquanto o aluno estiver escrevendo o código fonte do seu programa, a qualquer momento pode atualizar sintaxe, para que o *syntax highlighting* do seu código seja atualizado. E também poderá chamar o *content assist*, que trará os possíveis complementos de código.

Na execução ou depuração do programa, o sistema gera os arquivos no servidor, para que possam ser compilados, e depois de compilados, os arquivos são executados. Mas se na compilação, houver algum erro, a mensagem de erro é retornada para o usuário, e a execução ou depuração é abortada.

Se não houver erros de compilação, na execução, é aberto uma janela onde o aluno poderá fazer entrada de dados quando o programa solicitar e também ver a saída de dados do seu programa. O processo é finalizado quando o usuário fecha a janela ou o programa chega ao seu fim.

Na depuração, também é aberta uma janela, onde é possível visualizar o código fonte, a linha focada que será executada, as variáveis que o programa esta usando e a parte para fazer a entrada e saída de dados quando necessário. O aluno poderá escolher em executar linha por linha, ou continuar o processo até que chegue ao seu final. O processo é finalizado quando o aluno fecha a janela ou o programa chega ao seu fim.

Se o aluno não terminou o exercício, ele pode salvar o exercício para continuar em outro momento, ou se terminou, finalizá-lo para o professor ou monitor corrigirem. Após finalizá-lo, o sistema envia um email para o professor informando que o aluno finalizou o exercício.

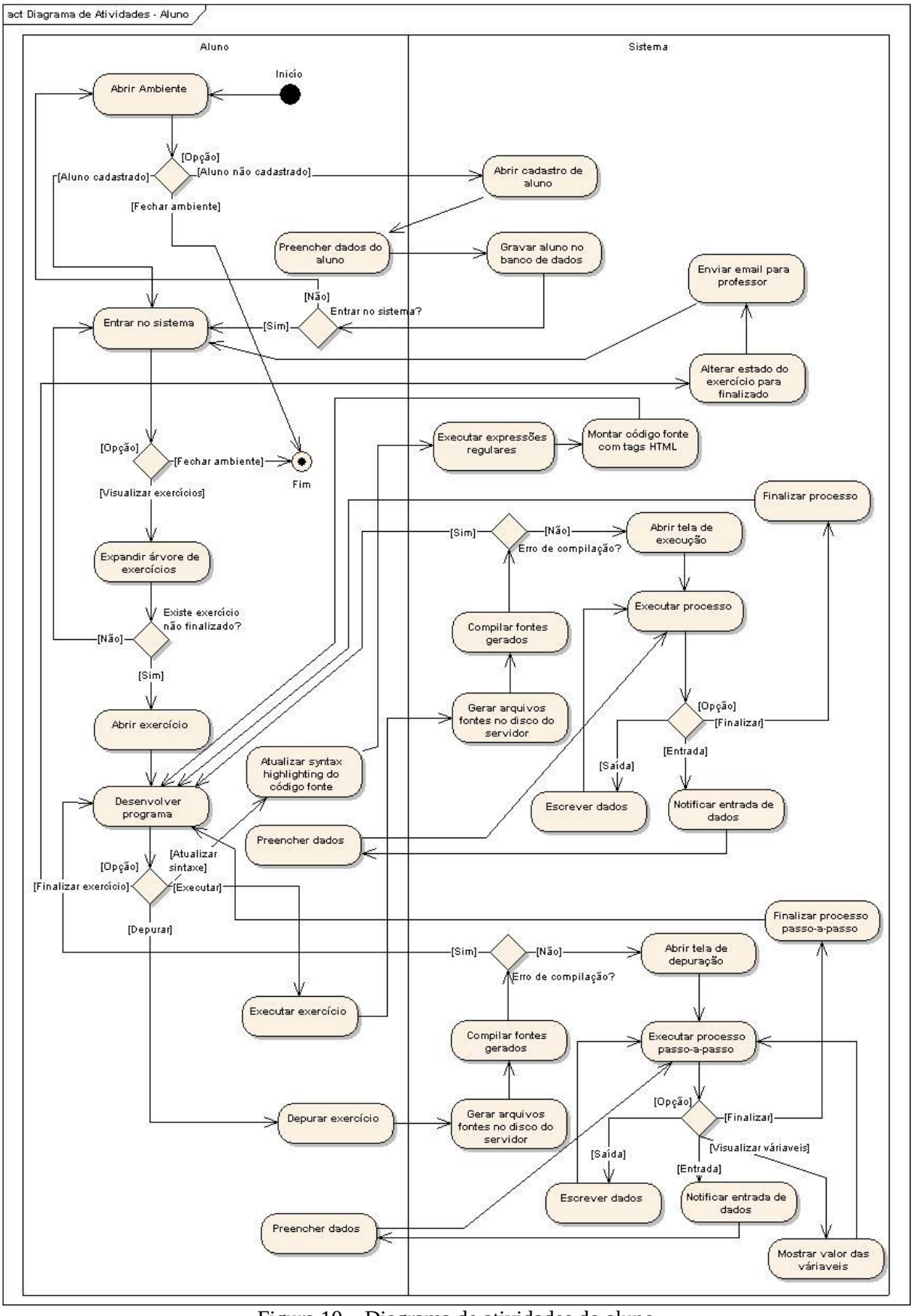


Figura 10 – Diagrama de atividades do aluno

3.2.3 Diagrama de classes

Para Furlan (1998, p. 91), diagrama de classe trata-se de uma estrutura lógica estática em uma superfície de duas dimensões mostrando uma coleção de elementos declarativos de modelo, como classes, tipos e seus respectivos conteúdos e relações.

Na Figura 11 são apresentadas as classes da camada de modelo e controle do usuário, e o Quadro 14 apresenta a responsabilidade de cada classe.

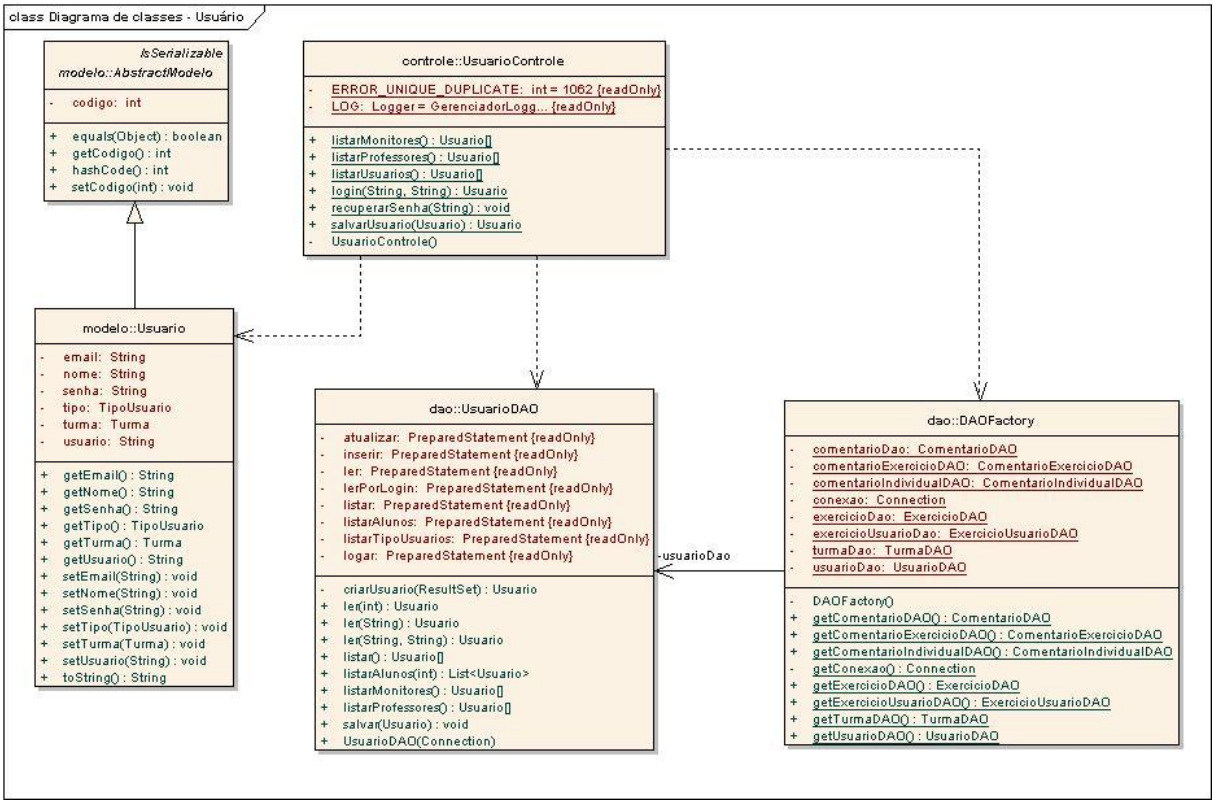


Figura 11 – Classes de controle e modelo do usuário

Classe	Descrição
`AbstractModelo`	Classe utilizada como base nas classes da camada de modelo, encapsulando o valor do código e implementando métodos básicos como `equals` e `hashcode`.
`Usuario`	Classe utilizada para encapsular dados de um usuário em relação ao usuário.
`UsuarioControle`	Classe utilizada para realizar a integração entre a camada de modelo/controle com a camada de visão.
`DAOFactory`	Classe utilizada para encapsular a conexão com o banco de dados e construir/fornecer as classes DAO.
`UsuarioDAO`	Classe utilizada para realizar ações no banco de dados como `select`, `delete`, `insert` e `update` referente a tabela de usuários.

Quadro 14 – Descrição das classes de controle e modelo do usuário

Na Figura 12 são apresentadas as classes da camada de modelo e controle da turma, e o Quadro 15 apresenta a responsabilidade de cada classe.

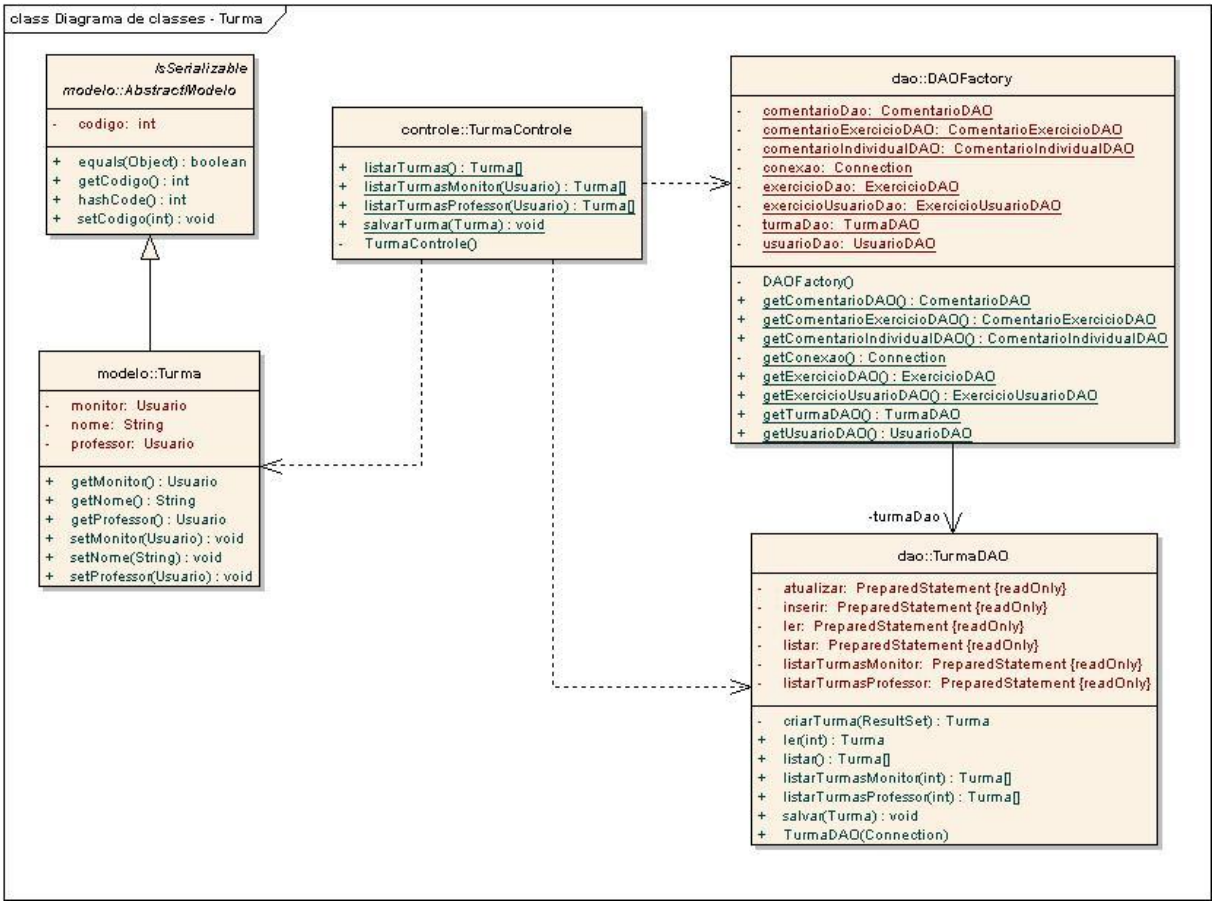


Figura 12 – Classes de controle e modelo da turma

Classe	Descrição
`Turma`	Classe utilizada para encapsular dados de uma turma.
`TurmaControle`	Classe utilizada para realizar a integração entre a camada de modelo/controle com a camada de visão em relação a turma.
`TurmaDAO`	Classe utilizada para realizar ações no banco de dados como `select`, `delete`, `insert` e `update` referente a tabela de turmas.

Quadro 15 – Descrição das classes de controle e modelo da turma

Na Figura 13 são apresentadas as classes da camada de modelo e controle do exercício, e o Quadro 16 apresenta a responsabilidade de cada classe.

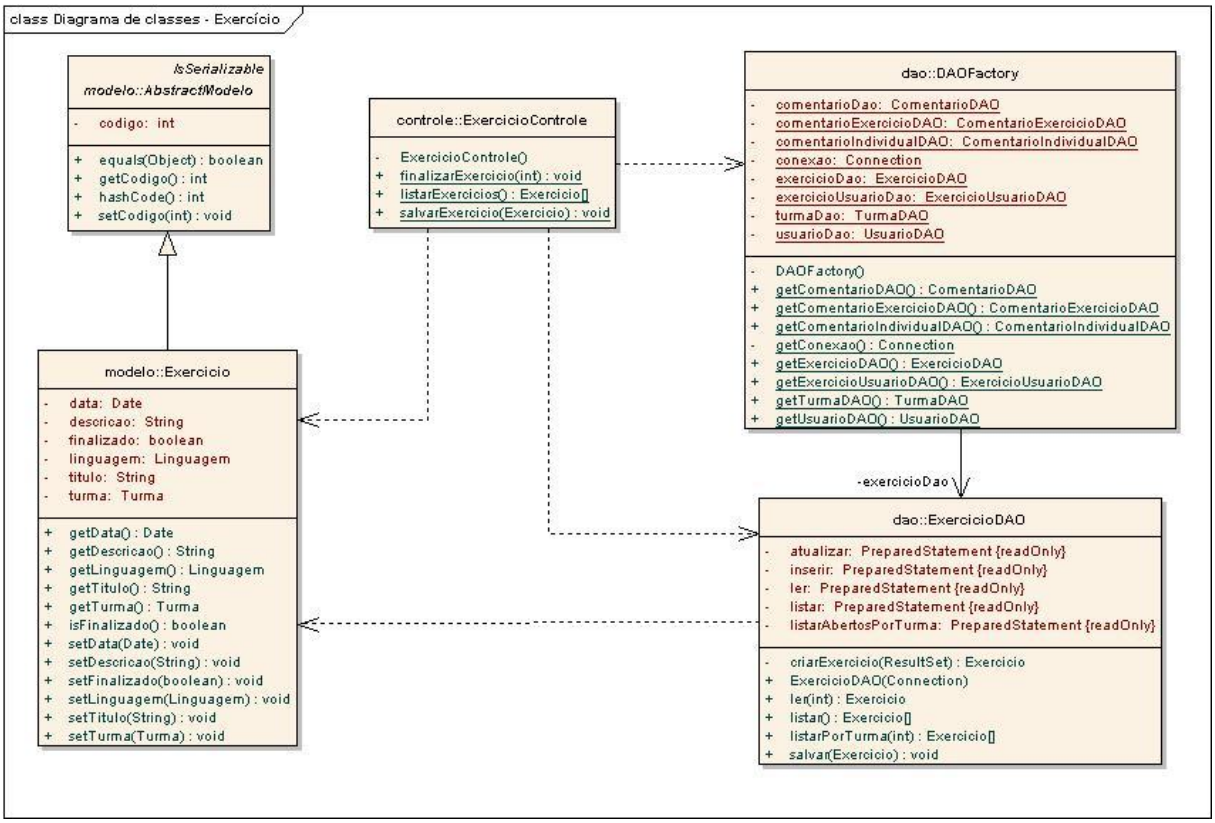


Figura 13 – Classes de controle e modelo do exercício

Classe	Descrição
`Exercicio`	Classe utilizada para encapsular dados de um exercício.
`ExercicioControle`	Classe utilizada para realizar a integração entre a camada de modelo/controle com a camada de visão em relação ao exercício.
`ExercicioDAO`	Classe utilizada para realizar ações no banco de dados como `select`, `delete`, `insert` e `update` referente a tabela de exercícios.

Quadro 16 – Descrição das classes de controle e modelo do exercício

Na Figura 14 são apresentadas as classes da camada de modelo e controle do comentário, e o Quadro 17 apresenta a responsabilidade de cada classe.

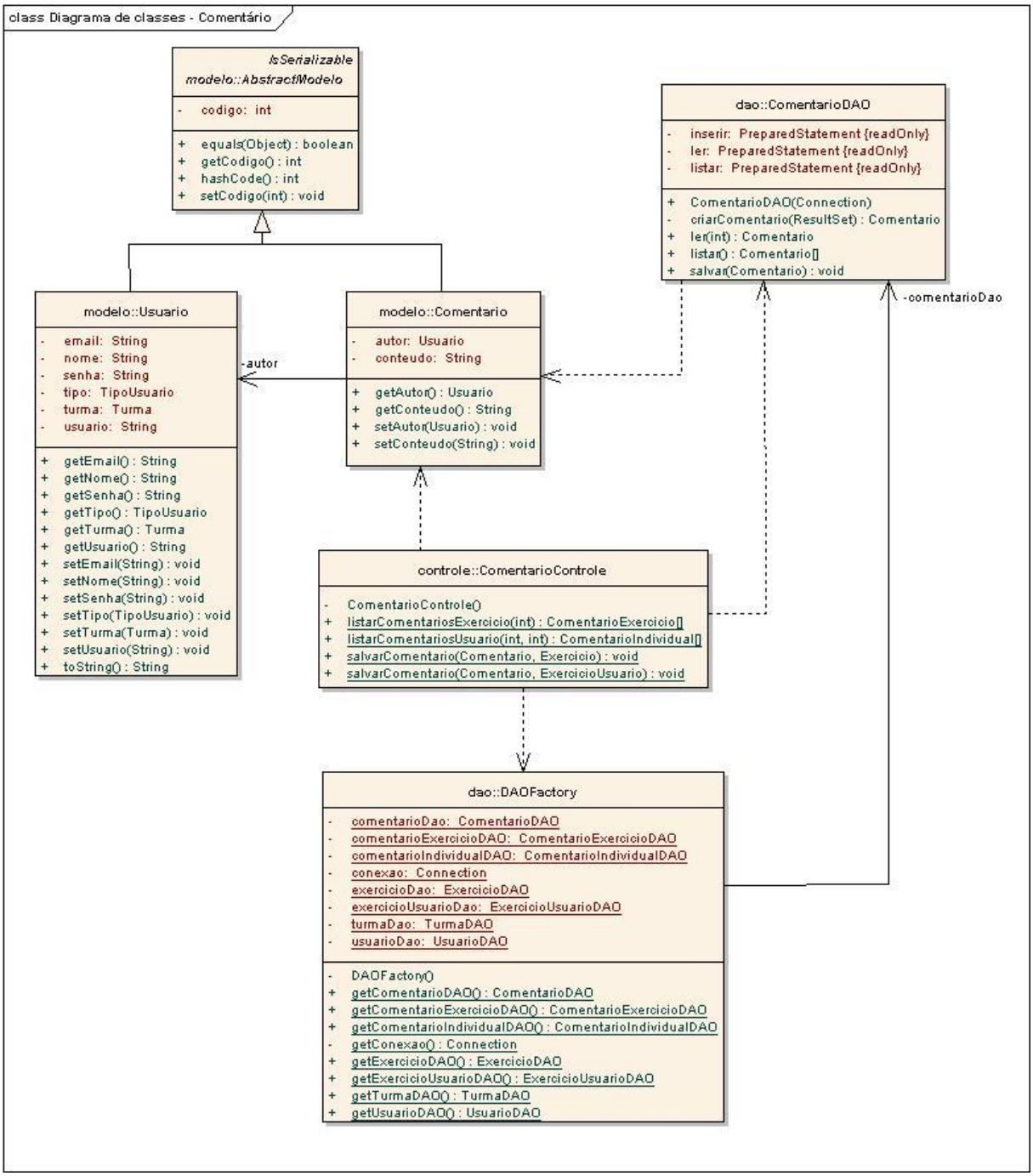


Figura 14 – Classes de controle e modelo do comentário

Classe	Descrição
`Comentario`	Classe utilizada para encapsular dados de um comentário.
`ComentarioControle`	Classe utilizada para realizar a integração entre a camada de modelo/controle com a camada de visão em relação ao comentário.
`ComentarioDAO`	Classe utilizada para realizar ações no banco de dados como `select`, `delete`, `insert` e `update` referente a tabela de comentários.

Quadro 17 – Descrição das classes de controle e modelo do comentário

Na Figura 15 são apresentadas as classes da camada de modelo e controle do exercício do usuário, e o Quadro 18 apresenta a responsabilidade de cada classe.

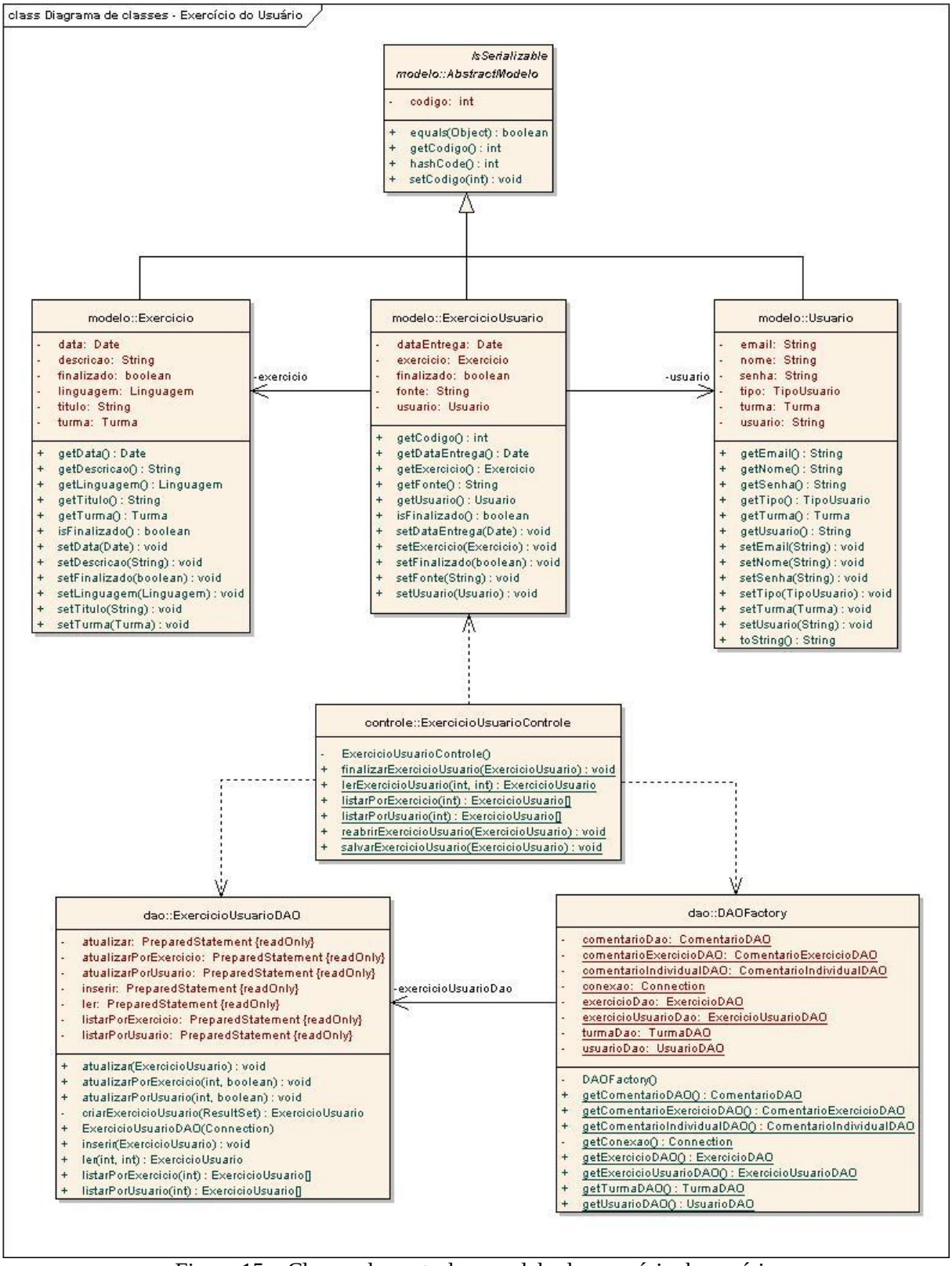


Figura 15 – Classes de controle e modelo do exercício do usuário

Classe	Descrição
`Exercicio`	Classe utilizada para encapsular dados de um exercício.
`ExercicioUsuario`	Classe utilizada para encapsular dados de um exercício do usuário.
`ExercicioUsuarioControle`	Classe utilizada para realizar a integração entre a camada de modelo/controle com a camada de visão em relação ao exercício do usuário.
`ExercicioUsuarioDAO`	Classe utilizada para realizar ações no banco de dados como `select`, `delete`, `insert` e `update` referente a tabela de exercícios do usuário.

Quadro 18 – Descrição das classes de controle e modelo do exercício do usuário

Na Figura 16 são apresentadas as classes da camada de modelo e controle do comentário do exercício, e o Quadro 19 apresenta a responsabilidade de cada classe.

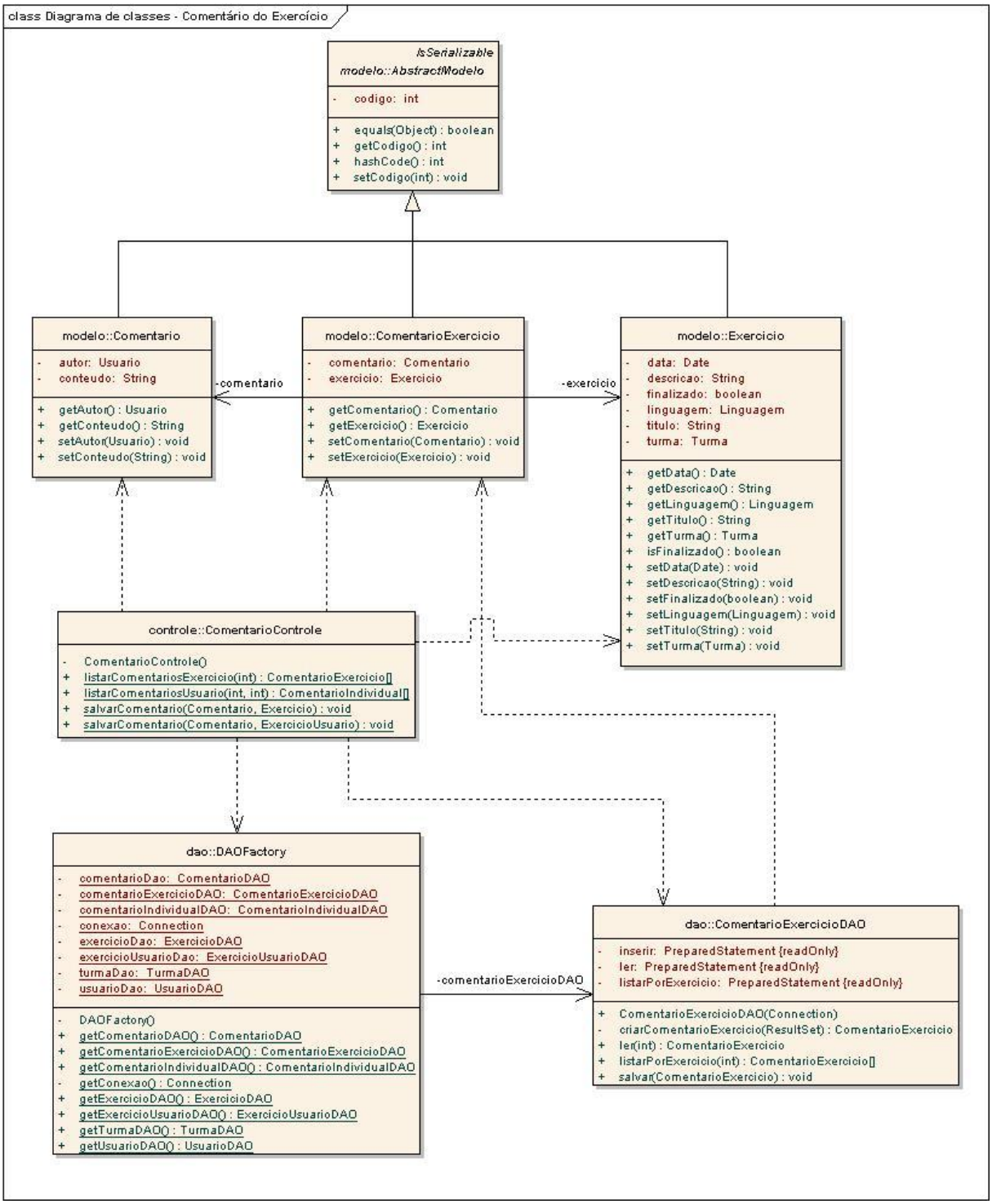


Figura 16 – Classes de controle e modelo do comentário do exercício

Classe	Descrição
`ComentarioExercicio`	Classe utilizada para encapsular dados de um comentário do exercício.
`ComentarioControle`	Classe utilizada para realizar a integração entre a camada de modelo/controle com a camada de visão em relação ao comentário do exercício.
`ComentarioExercicioDAO`	Classe utilizada para realizar ações no banco de dados como `select`, `delete`, `insert` e `update` referente a tabela de comentários do exercício.

Quadro 19 – Descrição das classes de controle e modelo do comentário do exercício

Na Figura 17 são apresentadas as classes da camada de modelo e controle do comentário individual e o Quadro 20 é apresentada a responsabilidade de cada classe.

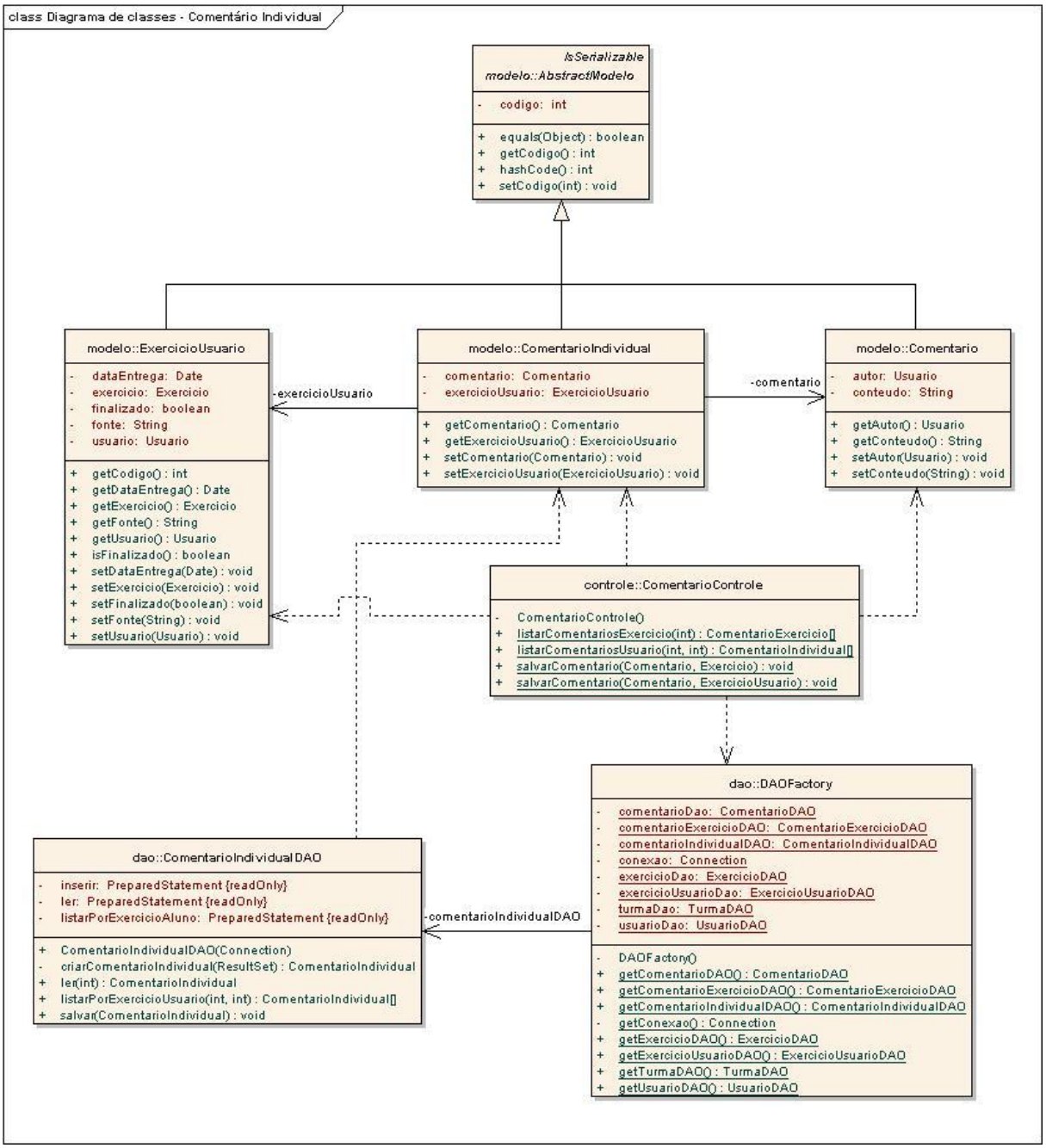


Figura 17 – Classes de controle e modelo do comentário individual

Classe	Descrição
`ComentarioIndividual`	Classe utilizada para encapsular dados de um comentário do usuário.
`ComentarioIndividualDAO`	Classe utilizada para realizar ações no banco de dados como `select`, `delete`, `insert` e `update` referente a tabela de comentários do usuário.

Quadro 20 – Descrição das classes de controle e modelo do comentário individual

Na Figura 18 são apresentadas as classes usadas na implementação do depurador de código Java da camada JDI, e no Quadro 21 são apresentadas as descrições de cada classe.

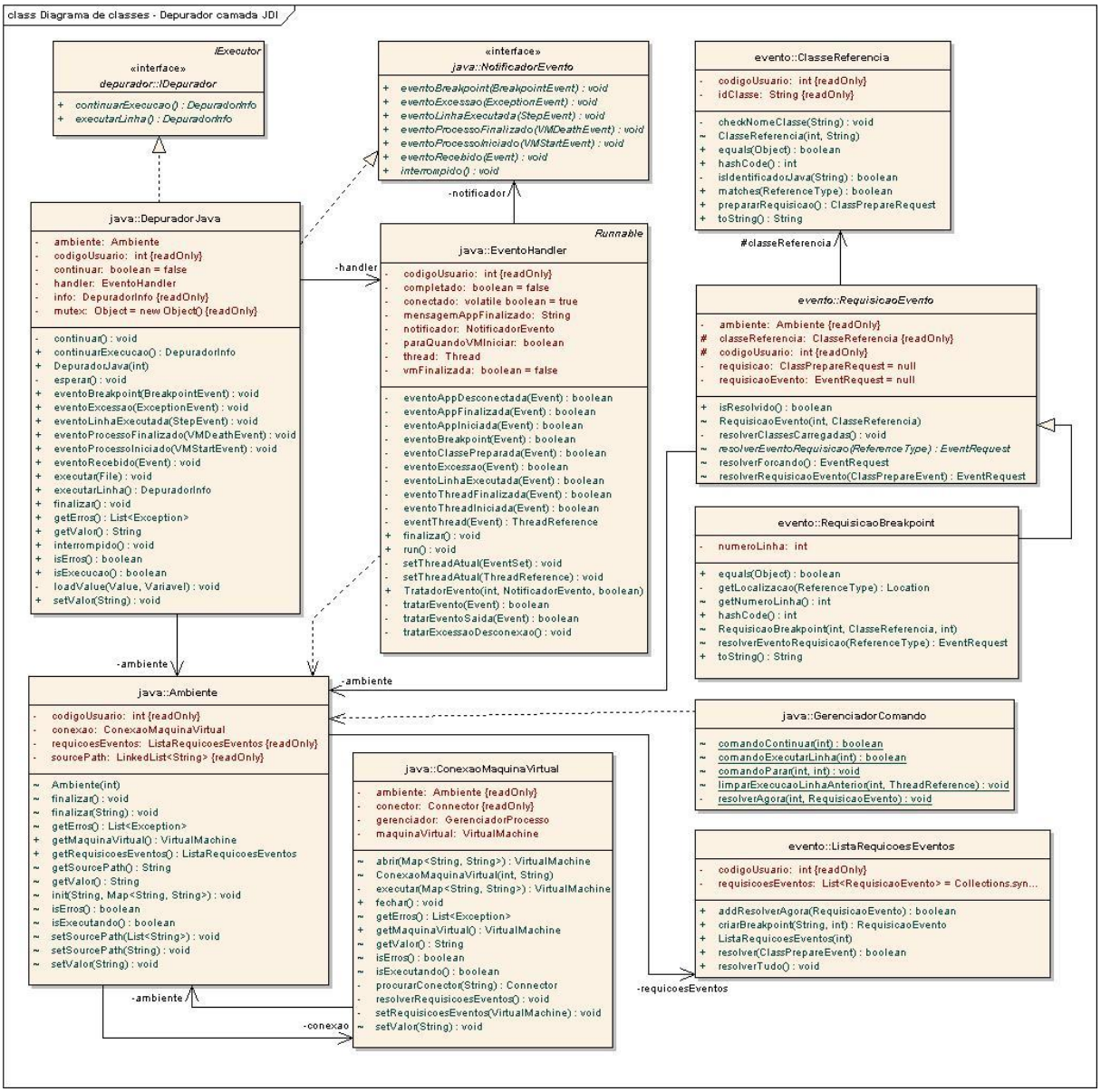


Figura 18 - Classes utilizadas no depurador Java

Classe	Descrição
`IDepurador`	Interface para a implementação de um depurador de alguma linguagem. Usada para comunicação entre cliente e servidor.
`DepuradorJava`	Implementação da interface `IDepurador` e `NotificadorEvento`. Responsável por receber as notificações de eventos que ocorrem no processo de depuração, e os tratar.
`NoticadorEvento`	Interface para implementação do notificador de eventos que ocorrem no processo de depuração.
`EventoHandler`	Responsável por manipular os eventos provenientes da fila da máquina virtual e os enviar para o notificador.
`Ambiente`	Responsável por encapsular a conexão da máquina virtual e a lista de requisições.
`ConexaoMaquinaVirtual`	Responsável por conectar-se na máquina virtual e encapsular o processo criado para a depuração.
`RequisicaoEvento`	Responsável por tratar a classe da requisição. Por exemplo, a classe onde o *breakpoint* será localizado.
`ClasseReferencia`	Responsável por identificar a classe de uma requisição.
`RequicaoBreakpoint`	Responsável por tratar a linha onde breakpoint esta localizado, pois a linha pode ser inválida. Uma linha é inválida quando ela esta em branco ou existe uma declaração de variável local sem inicialização.
`ListaRequisicoesEventos`	Responsável por guardar as requisições para a máquina virtual.
`GerenciadorComando`	Responsável por criar requisições para a máquina virtual. As requisições implementadas são, requisição de breakpoint, requisição de execução de uma linha e continuar execução até o final do processo ou até o próximo breakpoint.

Quadro 21 - Descrição das classes utilizadas no depurador de código Java

Na Figura 19 é apresentada a classe da biblioteca de entrada e saída de dados, e no Quadro 22 são apresentadas as descrições dos métodos.

Figura 19 - Classe utilizada para entrada e saída de dados durante a execução e depuração

Método	**Retorno**	**Parâmetros**
`WebIde.escreverValor`	-	String, String
`WebIde.escreverValor`	-	String, char
`WebIde.escreverValor`	-	String, byte
`WebIde.escreverValor`	-	String, int
`WebIde.escreverValor`	-	String, long
`WebIde.escreverValor`	-	String, float
`WebIde.escreverValor`	-	String, double
`WebIde.escreverValor`	-	String, boolean
`WebIde.lerBoolean`	boolean	String
`WebIde.lerByte`	byte	String
`WebIde.lerChar`	char	String
`WebIde.lerFloat`	float	String
`WebIde.lerInt`	int	String
`WebIde.lerLong`	long	String
`WebIde.lerString`	String	String
`WebIde.lerOpcao`	String	String, Object[]

Quadro 22 - Descrição dos métodos da classe WebIde que fornece entrada e saída de dados

3.2.4 Modelo de entidades e relacionamentos

Segundo Mecenas e Oliveira (2005, p.34 e 36), entidade pode ser definida como um

conjunto de elementos (objetos) de que o negócio do usuário precisa para funcionar adequadamente. E relacionamentos são as associações entre os objetos, que interagem em decorrência de uma razão relevante do negócio.

A Figura 20 apresenta o modelo de entidade-relacionamento utilizado no sistema.

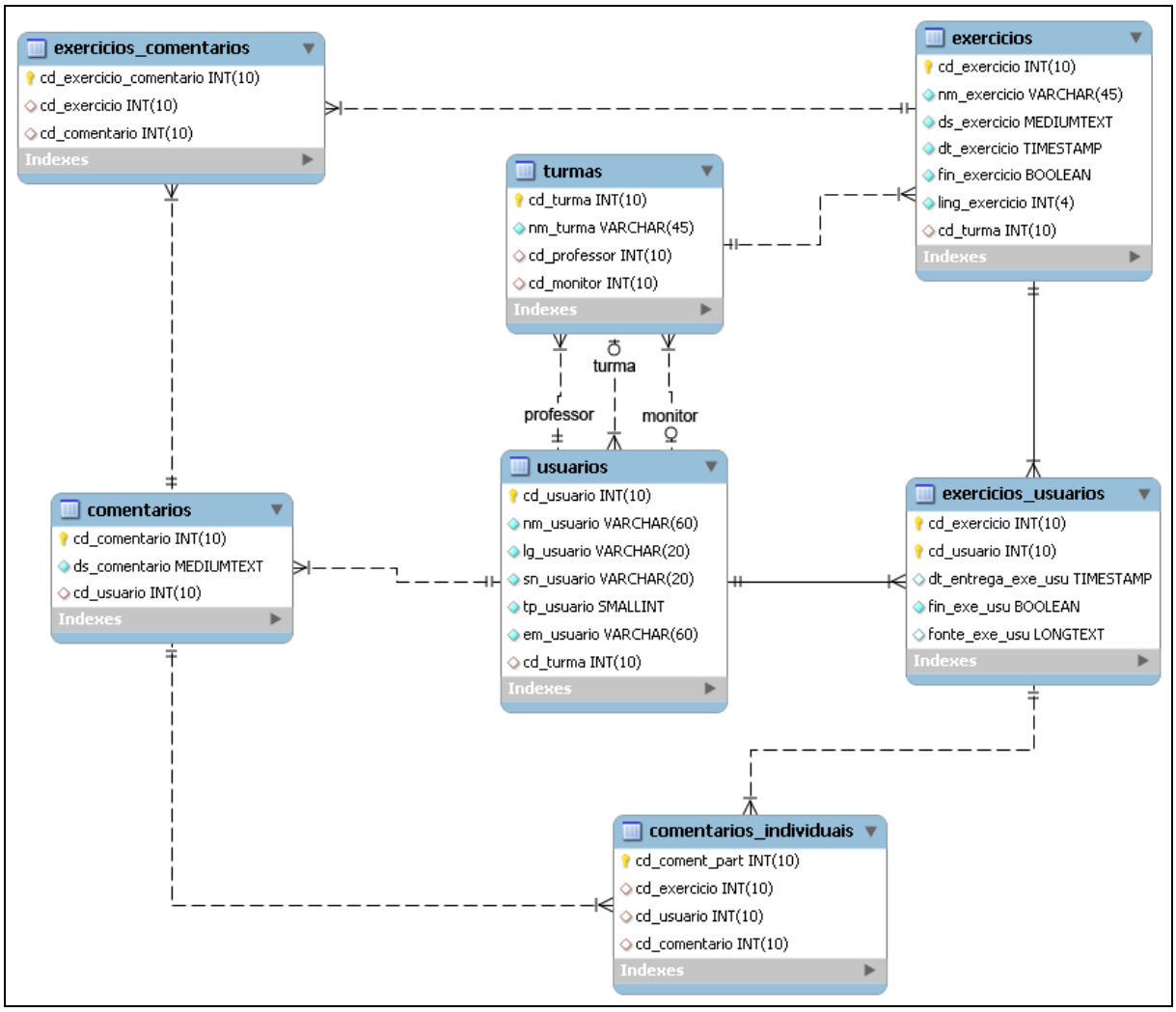


Figura 20 – Modelo de entidades e relacionamentos

No Quadro 23 é apresentado o dicionário de dados do modelo entidades e relacionamentos deste trabalho.

Tabela usuarios	
cd_usuario	Código do usuário no sistema
nm_usuario	Nome do usuário
lg_usuario	Login do usuário
sn_usuario	Senha do usuário
tp_usuario	Tipo de usuário (professor, monitor ou aluno)
em_usuario	Email do usuário
cd_turma	Código da turma que o usuário participa
Tabela comentarios	
cd_comentario	Código do comentário no sistema
ds_comentario	Texto do comentário
cd_usuario	Código do usuário que enviou o comentário
Tabela turmas	
cd_turma	Código da turma no sistema
nm_turma	Nome da turma
cd_professor	Código do professor da turma
cd_monitor	Código do monitor da turma
Tabela exercicios	
cd_exercicio	Código do exercício no sistema
nm_exercicio	Nome do exercício
ds_exercicio	Descrição do exercício (enunciado)
dt_exercicio	Data que o exercício foi criado
fin_exercicio	Indicador de exercício finalizado pelo professor
ling_exercicio	Linguagem de programação usada no exercício
cd_turma	Código da turma que recebeu o exercício
Tabela exercicios_comentarios	
cd_exercicio_comentario	Chave primária utilizada como índice na tabela
cd_exercicio	Código do exercício
cd_comentario	Código do comentário
Tabela comentarios_individuais	
cd_coment_part	Chave primária utilizada como índice na tabela
cd_exercicio	Código do exercício
cd_usuario	Código do usuário
cd_comentario	Código do comentário
Tabela exercicios_usuarios	
cd_exercicio	Código do exercício

cd_usuario	Código do usuário
dt_entrega_exe	Data da finalização do exercício
fin_exe_usu	Indicador de exercício finalizado pelo aluno/professor
fonte_exe_usu	Código fonte do usuário

Quadro 23 – Dicionário de dados

3.3 IMPLEMENTAÇÃO

Para o desenvolvimento da aplicação foi utilizada a linguagem de programação Java JDK 1.6 através da IDE Eclipse 3.5, o *framework* utilizado foi o GWT 1.7 integrado com a biblioteca Ext JS 2.0.2, e para servidor de aplicação foi utilizado o Apache-Tomcat 6.0 e o banco de dados MySQL 5.5.

3.3.1 Técnicas e ferramentas utilizadas

Nesta seção serão apresentadas quatro seções. Na seção 3.3.1.1 é apresentado a implementação da entrada e saída de dados do ambiente. Na seção 3.3.1.2 é mostrado como foi desenvolvida a depuração utilizando JPDA. Na seção 3.3.1.3 é visto a implementação do *Syntax Highlighting*. E por último, na seção 3.3.1.4 é apresentado a implementação do *Content Assist*.

3.3.1.1 Entrada e saída de dados

Conforme mostrado na Figura 19 e Quadro 22, foi desenvolvida uma biblioteca de entrada e saída de dados para ser usado na implementação do código do usuário. Ela fornece a possibilidade de escrever e entrar com dados na execução ou depuração do programa. A biblioteca é composta pela classe Java `WebIde`.

Todos os métodos têm como primeiro parâmetro uma String, que representa a identificação do dado na tela para o usuário final. Os métodos de escrita têm como segundo parâmetro o tipo do dado que será apresentado na tela. E entre os métodos de leitura, o

`lerOpcao` é o único que tem um segundo parâmetro, pois ele fornece valores já pré definidos pelo usuário.

No Quadro 24 é apresentado um código exemplo usando a biblioteca de entrada e saída de dados.

```
String nome = WebIde.lerString("Digite seu nome");
int idade = WebIde.lerInt("Digite sua idade");
String sexo = WebIde.lerOpcao("Escolha seu sexo", new String[] {"M", "F"});
WebIde.escreverValor("Nome", nome);
WebIde.escreverValor("Idade", idade);
WebIde.escreverValor("Sexo", sexo);
```

Quadro 24 – Código exemplo usando a biblioteca de entrada e saída de dados

A Figura 21 mostra a execução do código exemplo utilizando a biblioteca de entrada e saída de dados.

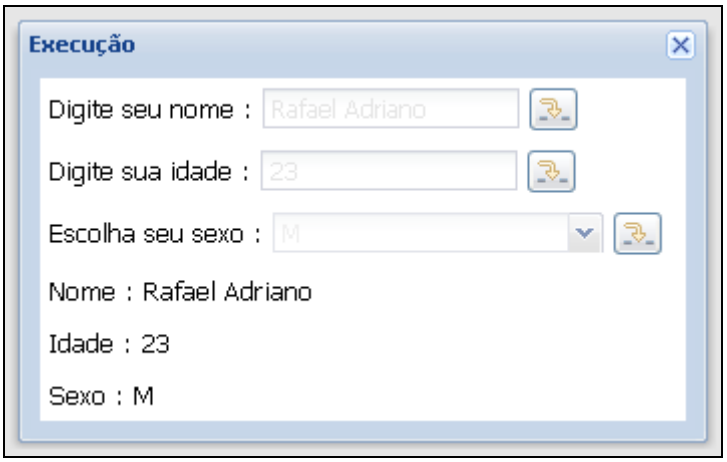


Figura 21 – Execução do código utilizando a biblioteca de entrada e saída de dados

3.3.1.2 Depuração usando JPDA

O desenvolvimento do depurador de código Java foi feito utilizando arquitetura JPDA, especificada pela SUN MICROSYSTEMS. Dentro da JDK 1.6, encontra-se o tools.jar. Este arquivo contém a implementação da especificação para linguagem Java. Ele já oferece uma API para trabalhar na camada JDI, onde os desenvolvedores podem implementar o controle de depuração de código Java, de acordo com sua necessidade.

O método `eventoLinhaExecutada` da classe `DepuradorJava` apresentado no Quadro 25, trata as informações das variáveis, como seu valor, seu tipo, se é primitivo, *array* ou um objeto complexo. Também se pode perceber que existe uma alta abstração das informações

vindas pelo evento, não é preciso implementar um *parser* para obtê-las, caso fosse executada diretamente do `jdb`[7], através da linha de comando.

```
public void eventoLinhaExecutada(StepEvent se) {
	Thread.yield();
	int subLinha = GerenciadorFonte.subLinha(se.location().lineNumber());
	info.setLinha(subLinha);
	try {
		// Se não saiu do main, continua
		StackFrame frame = null;
		for (StackFrame stackFrame : se.thread().frames()) {
			frame = stackFrame;
		}
		try {
			List<LocalVariable> variables = frame.visibleVariables();
			List<Variavel> vars = new ArrayList<Variavel>();
			for (LocalVariable localVariable : variables) {
				Variavel var = new Variavel();
				Value value = frame.getValue(localVariable);
				var.setNome(localVariable.name());
				if (value != null) {
					var.setValor(value.toString());
					// Verifica se não tem subvariaveis
					loadValue(value, var);
				}
				vars.add(var);
			}
			info.setVariaveis(vars);
		} catch (AbsentInformationException e) {
			// Se não tiver nenhuma váriavel
		}
	} catch (IncompatibleThreadStateException e) {
		LOG.error("Erro ao executar linha: ", e);
	}
	LOG.debug("Linha " + subLinha + " parada.");
	continuar();
}

private void loadValue(Value value, Variavel variavel) {
	variavel.setIcon("local_variable.gif");
	Type type = value.type();
	if (type instanceof ReferenceType) {

		List<Variavel> variaveis = variavel.getVariaveis();
		if (variaveis == null) {
			variaveis = new ArrayList<Variavel>();
		}
		variavel.setVariaveis(variaveis);

		if (type instanceof ArrayTypeImpl) {
			List<Value> values = ((ArrayReferenceImpl)
value).getValues();
			int index = 0;
			// Cria as variaveis para os valores dessa variavel
			for (Value val : values) {
				Variavel var = new Variavel();
				var.setNome("[" + index + "]");
```

[7] `jdb` é um depurador de fontes Java e sua execução é realizada através de linhas de comando (SUN MICROSYSTEMS, 2010b).

```
                        if (val != null) {
                              var.setValor(val.toString());
                              loadValue(val, var);
                        }
                        var.setIcon("index_array.gif");
                        variaveis.add(var);
                        ++index;
                  }
            } else if (type instanceof ClassTypeImpl) {
                  ObjectReferenceImpl objRef = (ObjectReferenceImpl) value;
                  Map<Field, Value> values =
objRef.getValues(objRef.referenceType().allFields());
                  Set<Field> keySet = values.keySet();
                  for (Field field : keySet) {
                        Variavel var = new Variavel();
                        Value val = values.get(field);
                        var.setNome(field.name());
                        if (val != null) {
                              var.setValor(val.toString());
                        }
                        var.setIcon("private_field.gif");
                        variaveis.add(var);
                  }
            }
      }
}
```

Quadro 25 – Definição dos métodos para tratar o evento de execução de uma linha

O método `executar` apresentado no Quadro 26, contido na interface `IExecutor` e implementado pela classe `DepuradorJava`, é o passo inicial para se criar o processo de depuração. Ele cria o ambiente, o notificador e invoca a conexão com a máquina virtual, que disponibiliza o processo para o ambiente.

```
@Override
public void executar(File arquivo) throws ExecucaoException {
      LOG.info("Iniciando processo de depuração.");
      AmbienteManager.criarAmbiente(codigoUsuario);
      ambiente = AmbienteManager.getAmbiente(codigoUsuario);

      Map<String, String> args = new HashMap<String, String>();

      String pasta = arquivo.getParentFile().getAbsolutePath();
      String jar = pasta + File.separator + "json.jar";

      args.put("main", "Exercicio");
      args.put("options", String.format("-classpath %s%s%s%s", pasta,
File.pathSeparator, jar, File.pathSeparator));

      ambiente.init("com.sun.jdi.CommandLineLaunch", args);

      handler = new EventoHandler(codigoUsuario, this, false);
}
```

Quadro 26 – Definação do método que inicia o processo de depuração

Os métodos `executarLinha` e `continuarProcesso` apresentados no Quadro 27, são as ligações da interação do usuário sobre o processo de depuração, onde são enviados requisições para executar uma linha ou continuar o processo até o final.

```
@Override
public DepuradorInfo continuarExecucao() {
	boolean executou =
GerenciadorComando.comandoContinuar(codigoUsuario);
	if (executou) {
		esperar();
	}
	return info;
}

@Override
public DepuradorInfo executarLinha() {
	LOG.debug("Executando linha");
	try {
		boolean executou =
GerenciadorComando.comandoExecutarLinha(codigoUsuario);
		if (executou) {
			esperar();
			LOG.debug("Linha executada");
		} else {
			LOG.warn("Não esperou executar a linha.");
		}
	} catch (Exception e) {
		LOG.error("Erro ao executar linha: ", e);
	}
	return info;
}
```

Quadro 27 – Definição dos métodos que enviam requisições para máquina virtual

Na Figura 22 é apresentada a tela de depuração de um código Java e as suas variáveis na aplicação.

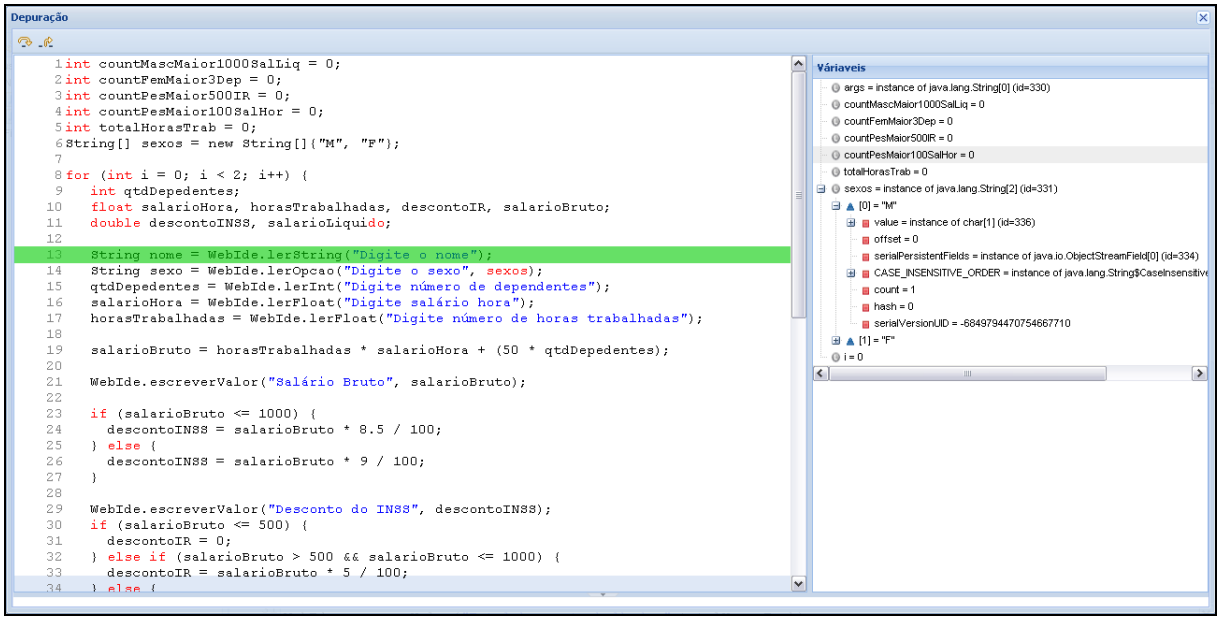


Figura 22 – Tela de depuração de um código Java na aplicação

3.3.1.3 *Syntax Highlighting*

No desenvolvimento do *syntax highlighting* foram utilizadas expressões regulares para encontrar as palavras reservadas, valores literais e comentário de código. O Java oferece uma API para realizar a execução de expressões regulares. Que encontra-se nas classes `Pattern` e `Matcher`. Ela compila em tempo de execução uma expressão, podendo usar essa expressão compilada várias vezes, sem perder desempenho. No Quadro 28 são apresentadas as expressões regulares usadas na implementação do *syntax highlighting*.

Expressão	Descrição
`//([^\n\r])*(\n|\r|\r\n)?`	Responsável por detectar um comentário de linha.
`"([^"\\\n\r]|(\\([ntbrf\\'"]|[0-7]([0-7])?|[0-3][0-7][0-7]))|(\\u[0-9a-fA-F][0-9A-Fa-f][0-9A-Fa-f][0-9A-Fa-f]))*"`	Responsável por detectar um valor literal do tipo String.
`'([^'\\\n\r]|(\\([ntbrf\\'"]|[0-7]([0-7])?|[0-3][0-7][0-7]))|(\\u[0-9a-fA-F][0-9A-Fa-f][0-9A-Fa-f][0-9A-Fa-f]))*'`	Responsável por detectar um valor literal do tipo char.
`synchronized|implements|instanceof|interface|protected|transient|strictfp|abstract|volatile|continue|default|boolean|extends|finally|package|private|public|return|static|switch|import|double|throws|assert|native|short|super|break|catch|class|const|throw|false|final|float|while|char|byte|case|else|enum|null|this|void|goto|true|long|try|for|new|int|if|do`	Responsável por detectar as palavras reservadas.

Quadro 28 – Expressões regulares usadas na implementação do syntax highlighting

A instância da classe `Pattern` que é gerada após a compilação, é *thread safe*. Desta forma não é necessário preocupar-se com a concorrência de *threads*. Com isso a compilação da expressão é feita somente uma vez, e é guardado em uma variável estática. O Quadro 29 mostra trecho de código da compilação das expressões.

```
// Expressões regulares
comentarioBloco = Pattern.compile(comentarioBloco);
comentarioLinha = Pattern.compile(comentarioLinha);
valoresLiterais = new Pattern[valoresLiterais.length];

// Valores literais
for (int i = 0; i < valoresLiterais.length; i++) {
      valoresLiterais[i] = Pattern.compile(valoresLiterais[i]);
}

// Palavras reservadas
StringBuilder bfPalavrasReservadas = new StringBuilder();
for (int i = 0; i < palavrasReservadas.length; i++) {
      bfPalavrasReservadas.append(palavrasReservadas[i]);
      bfPalavrasReservadas.append("|");
}
bfPalavrasReservadas.setLength(bfPalavrasReservadas.length() - 1);
palavrasReservadas = Pattern.compile(bfPalavrasReservadas.toString());
```

Quadro 29 – Compilação das expressões regulares usando a classe Pattern

Na Figura 23 é apresentado um código de fonte Java no editor da aplicação com o *syntax highlighting* aplicado.

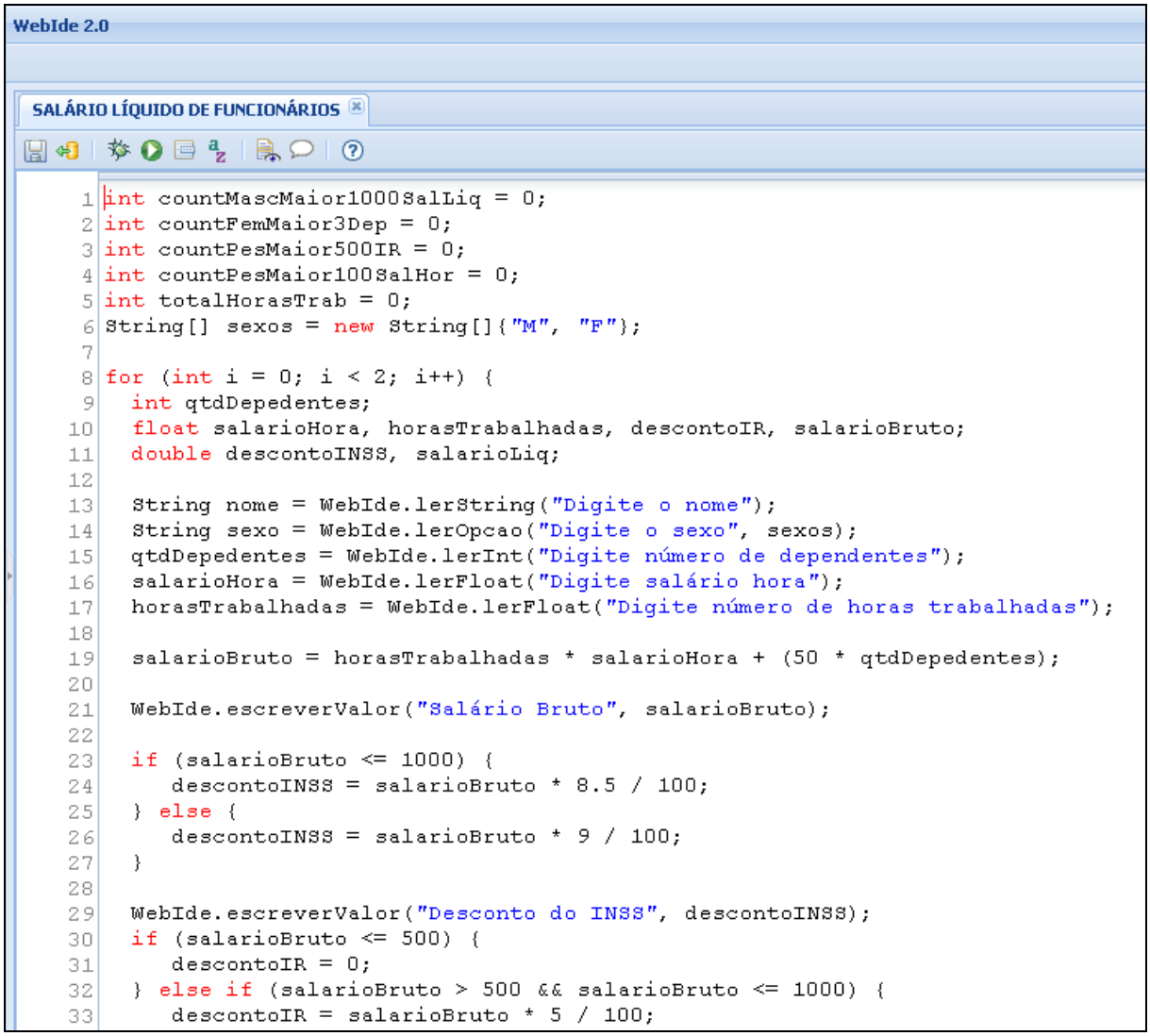


Figura 23 – Resultado do *syntax highlighting* de um código Java

3.3.1.4 *Content Assist*

O *content assist* de código foi desenvolvido usando a biblioteca javaparser, um projeto *opensource* criado por Gesser (2010). A implementação consiste em fazer a análise do código Java, percorrer a árvore de expressões procurando possíveis variáveis, métodos e classes que possam ser usadas em algum ponto do código. No Quadro 30 é mostrado o código do *parser*. Nota-se que é usada uma estratégia de injeção de código *dummy*[8] para que não ocorra erro léxico no *parser*.

```
@Override
public Complementar[] bind(String input, int coluna) throws Exception {
	List<Complementar> complementares = Collections.emptyList();
	input = input.replaceAll("\r\n", "");
	if (coluna < input.length()) {
		StringBuilder sb = new StringBuilder(input);
		sb.replace(coluna, sb.length()-1, DUMMY_TEMP + "();");
		CompilationUnit compUnit = innerParser(sb.toString());
		if (compUnit != null) {
			VisitadorJavaBind visitador = new VisitadorJavaBind();
			compUnit.accept(visitador, null);
			complementares = visitador.getComplementares();
		}
	} else if (coluna == input.length()) {
		CompilationUnit compUnit = innerParser(input + DUMMY_TEMP +
"();");
		if (compUnit != null) {
			VisitadorJavaBind visitador = new VisitadorJavaBind();
			compUnit.accept(visitador, null);
			complementares = visitador.getComplementares();
		}
	}
	return complementares.toArray(new
Complementar[complementares.size()]);
}
```

Quadro 30 – Parser do código Java e procura de complementos na árvore de expressões

Na Figura 24 é apresentada o resultado no sistema do *content assist*, que foi gerado a partir da classe `WebIde`, onde são mostrados todos os métodos possíveis da classe.

[8] *dummy* vem da língua inglesa, cujo sua tradução para língua portuguesa é falso, ou seja, no contexto do trabalho é um código falso.

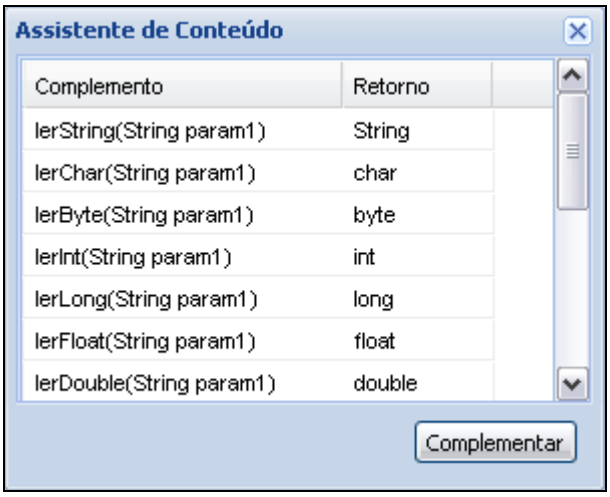


Figura 24 – Resultado do *content assist* no sistema

3.4 OPERACIONALIDADE DO AMBIENTE

Nesta seção é apresentada a operacionalidade da aplicação Na seção 3.4.1 será apresentada a criação de um exercício pelo professor. A seção 3.4.2 será mostrada o registro do aluno e seu login. A seção 3.4.3 será apresentada a resolução do exercício pelo aluno. A seção 3.4.4 será mostrada a correção do exercício pelo monitor.

3.4.1 Criando um exercício

Nesta seção é mostrada a criação de um exercício pelo professor, referente ao `UC04 – Criar Exercício`. Na Figura 25 é apresentado o ambiente onde o professor entrará para criar o exercício.

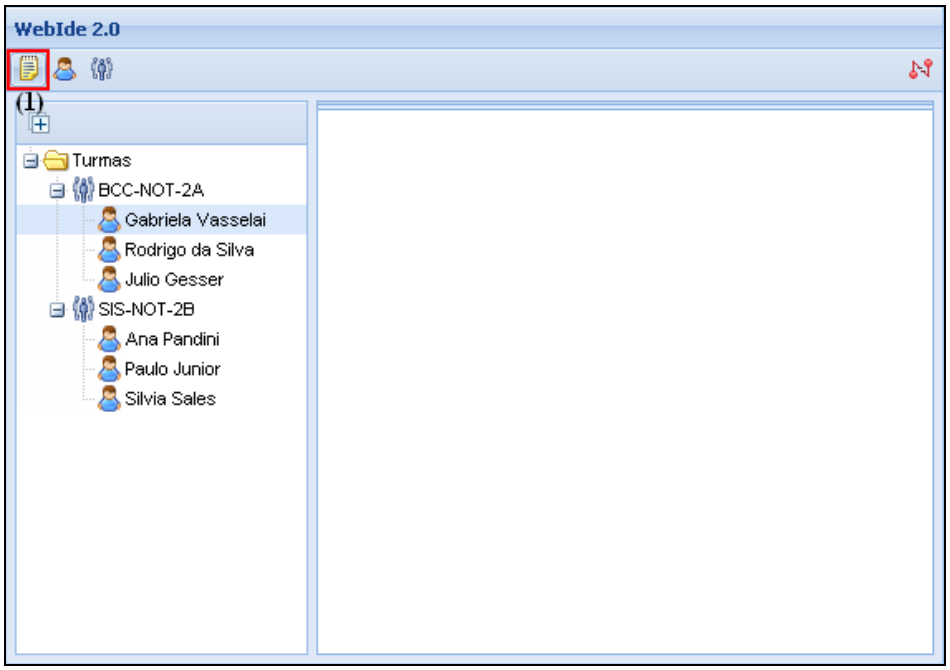


Figura 25 – Ambiente WebIde 2.0

Quando o professor clica no botão `Exercícios` (1), abre-se uma tabela que lista todos os exercícios já cadastrados, conforme a Figura 26.

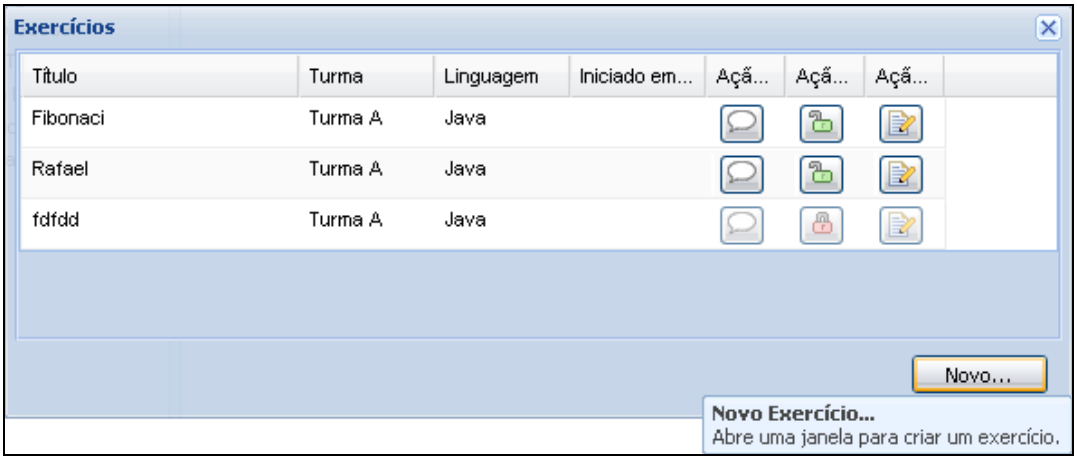


Figura 26 – Tela com todos os exercícios cadastrados no sistema

Quando o professor clicar no botão `Novo...`, irá abrir um formulário para preencher todos os dados necessários para criação de um exercício, visto na Figura 27.

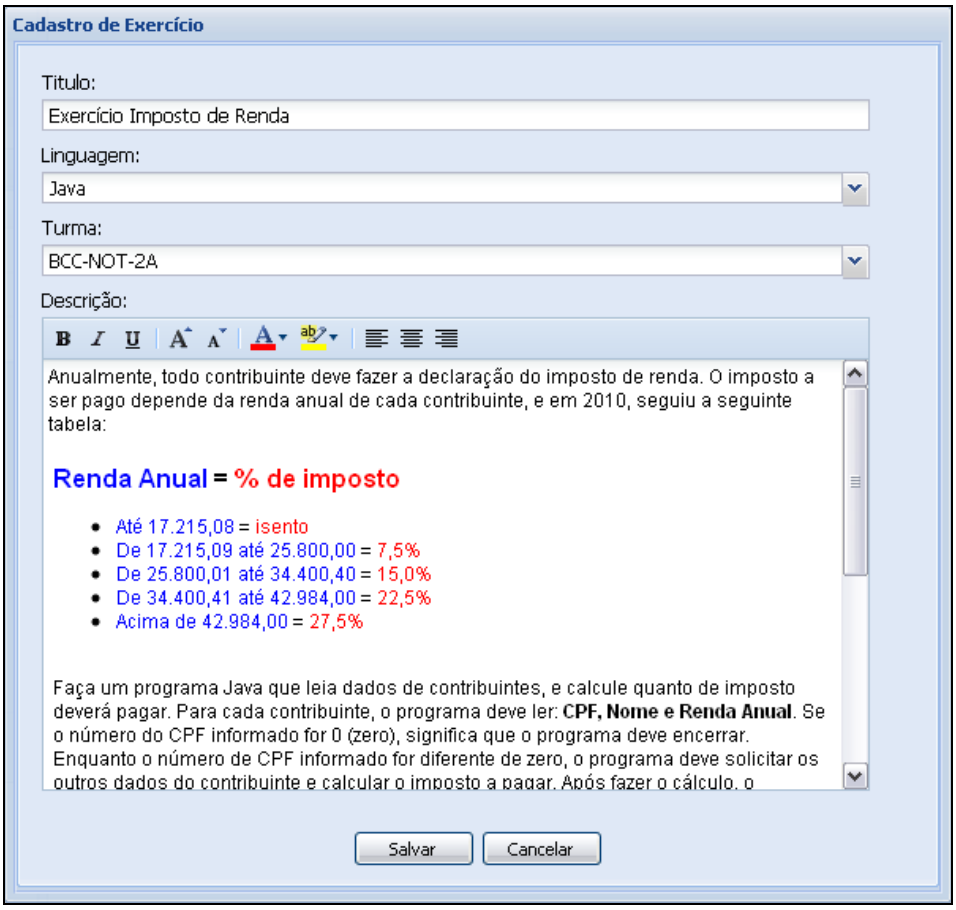


Figura 27 – Formulário de dados do exercício

Ao final da criação do exercício o professor clica no botão `Salvar`. O exercício será gravado no banco de dados e distribuído para todos os alunos da turma. A Figura 28 mostra a tabela de exercícios onde pode-se verificar que o exercício foi criado com sucesso. Depois de criado, o professor pode alterar o exercício, finalizá-lo ou fazer um comentário para turma sobre o exercício.

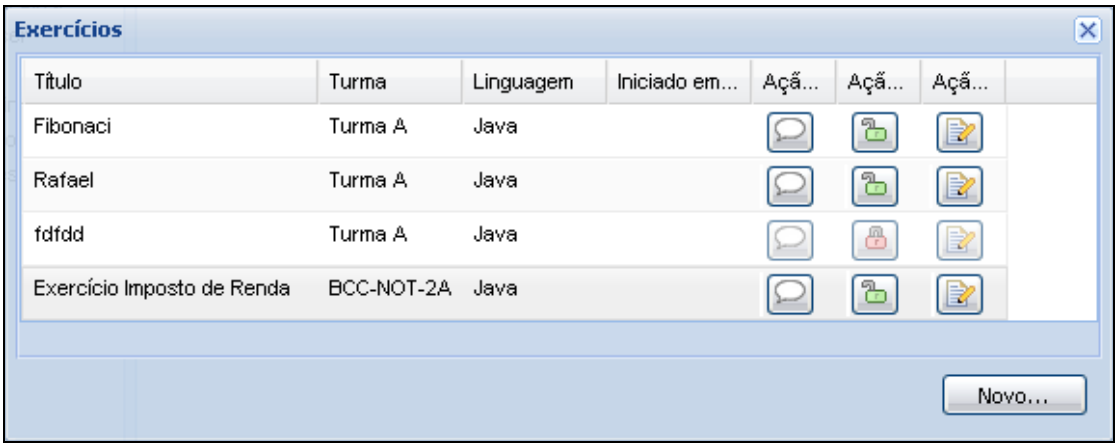


Figura 28 – Lista de todos os exercícios cadastrados no sistema

3.4.2 Registro do aluno e seu *login*

Nesta seção será apresentado o registro de um aluno no sistema, referente ao `UC05 - Registrar-se no ambiente`, e o seu *login*. Na Figura 29 é mostrada a tela inicial do sistema, onde o aluno irá clicar no botão `Sou Novo`.

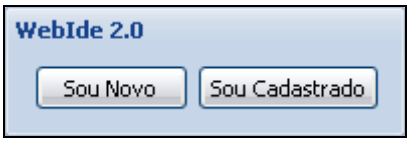


Figura 29 – Tela inicial do sistema

Após clicar no botão, o sistema mostrará a tela de cadastro de um aluno (Figura 30), onde o aluno preencherá os campos necessários para seu registro.

Figura 30 – Tela de cadastro do aluno

Ao clicar no botão `Salvar`, o aluno é gravado no banco de dados. Agora que o aluno está registrado no sistema, será mostrado o seu login. Na tela inicial, conforme mostrado na Figura 29 o aluno irá clicar no botão `Sou Cadastrado`, e o sistema mostrará a tela de login de um usuário, conforme Figura 31.

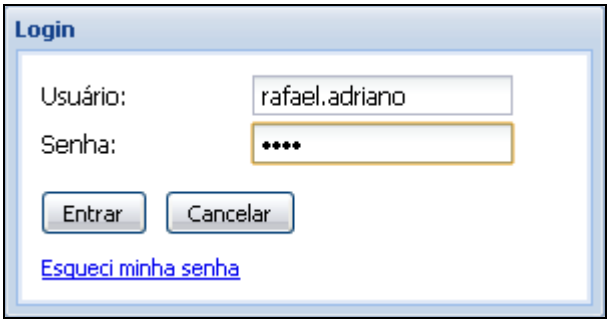


Figura 31 – Tela de *login* de um usuário

Nesta tela o aluno preencherá os dados necessários para o *login* e clicará no botão `Entrar`. Se os dados do usuário estiverem corretos, o sistema entrará com o aluno. Caso contrário é mostrado uma mensagem de *login* inválido.

3.4.3 Resolvendo um exercício

Nesta seção será visto a resolução do exercício pelo o aluno, referente ao `UC06 - Resolver Exercício`. Na Figura 32 são mostrados todos os exercícios abertos e fechados do aluno. Na lateral direita é o espaço onde é desenvolvido o exercício e na esquerda é a área onde é mostrada a árvore de exercícios.

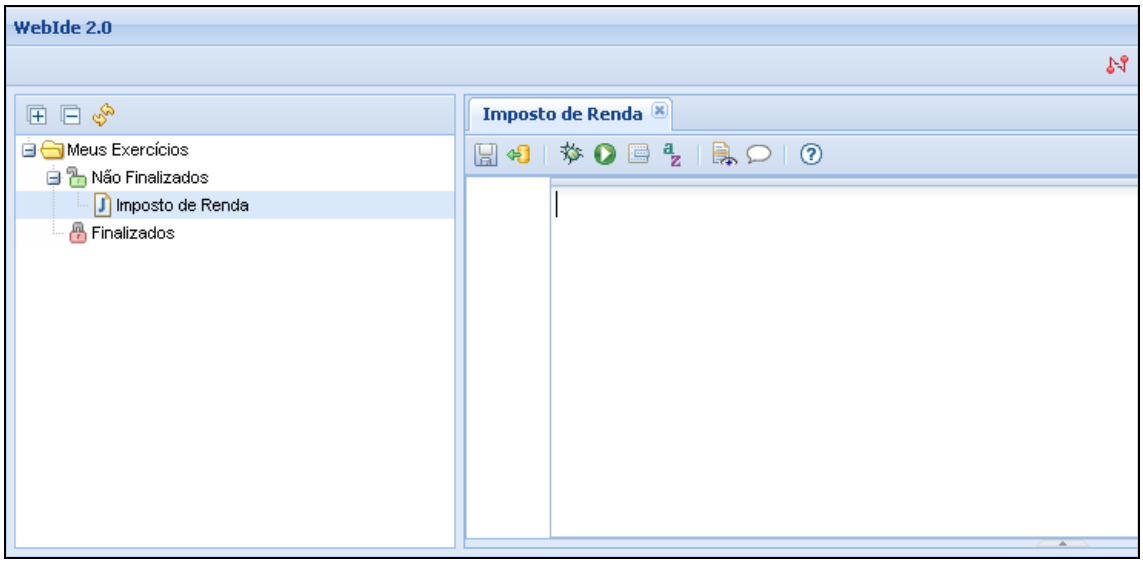


Figura 32 – Ambiente onde o aluno resolverá o exercício

Nesse ambiente encontram-se os botões salvar (1), finalizar e submeter (2), compilar e depurar (3), compilar e executar (4), assistente de conteúdo (5), atualizar sintaxe (6), enunciado do exercício (7), comentários do exercício (8) e ajuda (9), conforme a Figura 33.

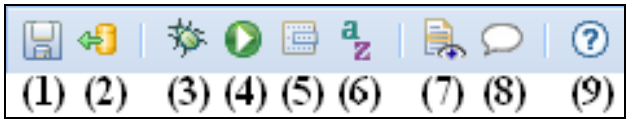


Figura 33 – Barra de ferramentas do editor

Na Figura 34 é apresentado o editor de código, e também o console mostrando um erro de compilação de um programa.

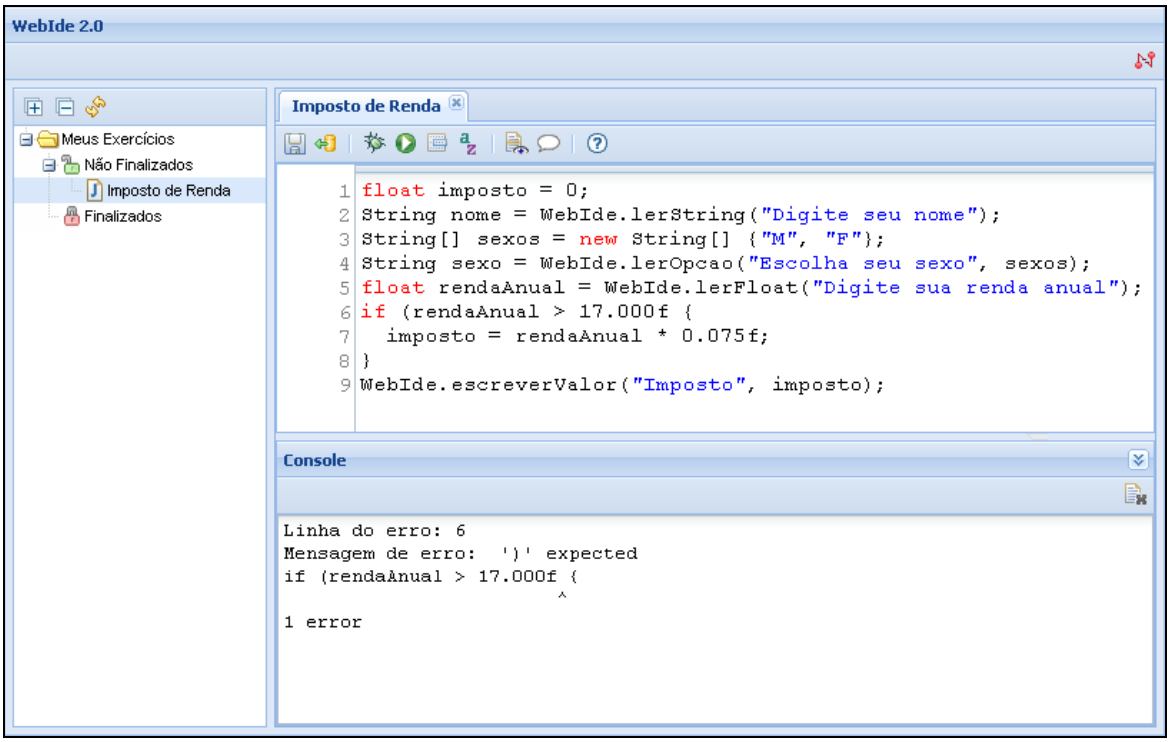


Figura 34 – Compilação do programa com erros

Os erros de compilação exibidos no console são formados pelo número da linha onde o erro ocorreu, mensagem do erro, seguido da linha de código onde o erro se encontra. A mensagem de erro é gerada pelo `javac` e mostrado no console da aplicação. Na Figura 35 é mostrado o programa que está codificado na Figura 34 sem erros de compilação, executando todos os passos programados. Na execução é digitado o nome (Passo 01), depois escolhido o sexo (Passo 02) e por último digitado a renda e exibido o resultado do programa (Passo 03).

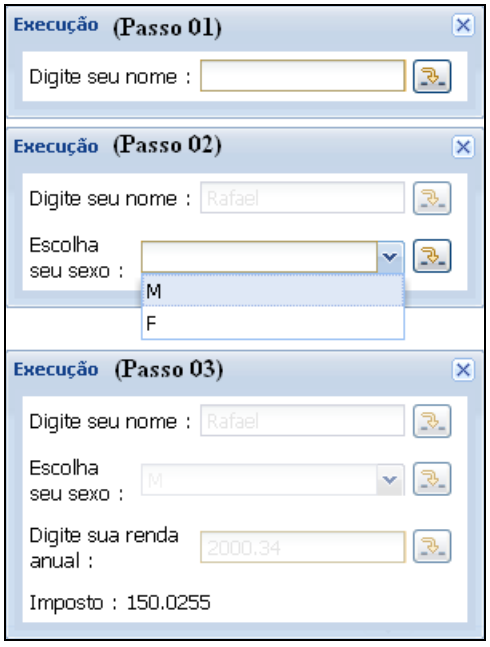


Figura 35 – Execução do programa

Na Figura 36 é mostrado o momento que o aluno depura o programa, onde é possível visualizar as variáveis e seus valores, entrar com dados e ver as saídas de dados. O botão para executar linha (1), é onde o aluno pode executar o programa passo-a-passo, e no botão ao lado é o botão para continuar processo (2), que faz com que o programa execute até o final, sem parar em mais nenhuma linha.

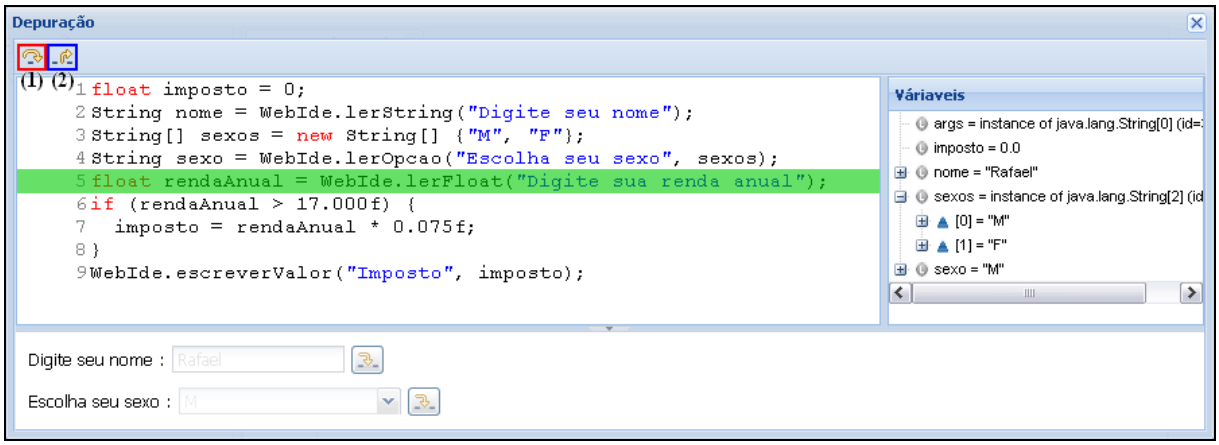


Figura 36 – Depuração do programa

3.4.4 Correção do exercício

Nesta seção é mostrada a correção de um exercício, referente ao `UC07 - Corrigir Exercício`. Na Figura 37 é apresentando o ambiente para o monitor. Nota-se que os botões para criar turmas (2) e usuários (1) do tipo monitor ou professor estão desabilitados. Isso porque o monitor somente pode comentar no exercício e reabri-lo para o aluno. Também são mostradas as turmas que o monitor pertence e os alunos de cada turma com seus respectivos exercícios finalizados.

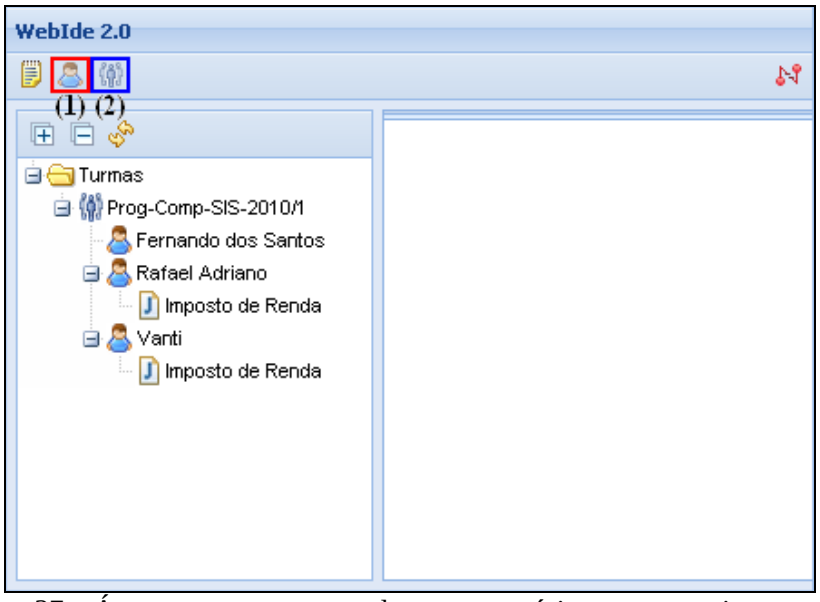


Figura 37 – Árvore com as turmas, alunos e exercícios que o monitor pertence

Após a correção do exercício, se o exercício estiver incorreto, o monitor enviará um comentário individual para o aluno, como mostra a Figura 38.

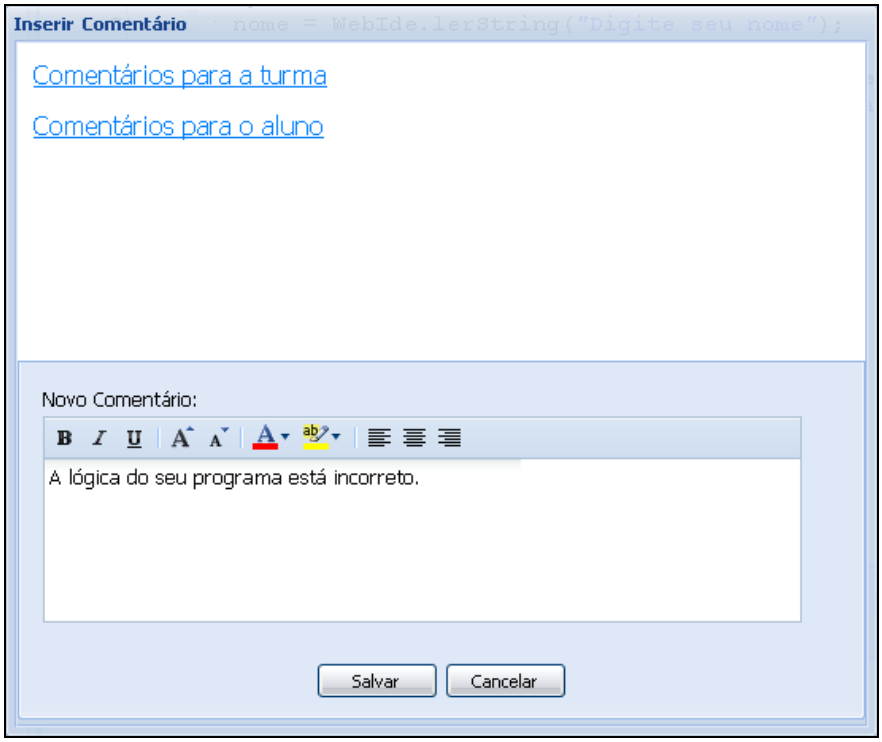


Figura 38 – Janela para cadastrar comentário individual

Depois que o monitor criar um comentário para o aluno, ele pode reabrir o exercício, para que o aluno possa corrigir.

Na Figura 39 apresenta os comentários que serão visto pelo aluno.

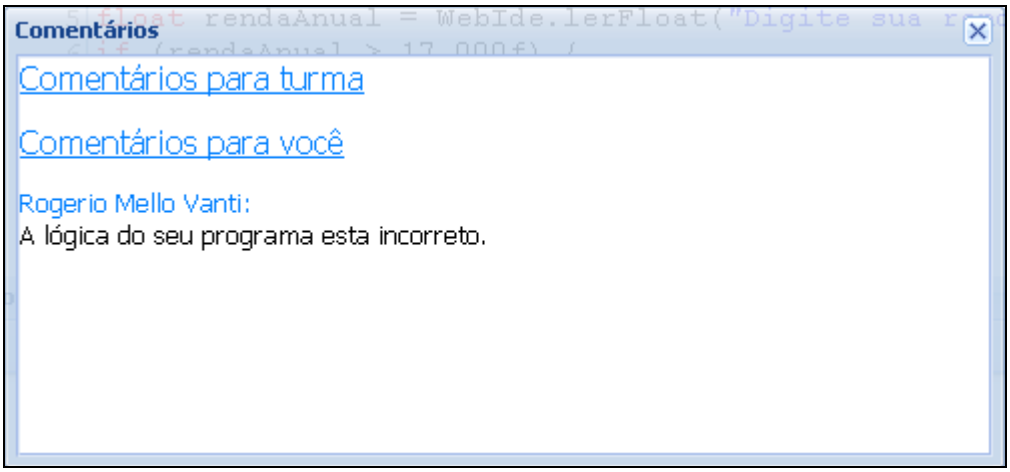


Figura 39 – Janela para visualizar comentários

Na janela de comentários é apresentado o autor e o comentário, separados por comentários da turma e comentários individuais.

3.5 VALIDAÇÃO

O sistema WebIde 2.0 foi implantado no servidor `campeche` da FURB. Entre os dias dezoito e vinte de maio de dois mil e dez, foi aplicado um exercício valendo dois pontos na recuperação de uma prova, para a turma noturna de Programação de Computadores de BCC ministrado pelo professor Fernando dos Santos. O exercício foi extraclasse, e os alunos puderam usar o sistema em qualquer computador, dentro ou fora das dependências da FURB.

Esta validação teve o intuito de testar a compatibilidade dos navegadores com o sistema, a usabilidade do sistema, o funcionamento com vários usuários simultâneos e a compilação, execução e depuração dos programas criados pelos alunos.

No dia vinte de maio de dois mil e dez, na parte da manhã, foi constatado que a aplicação tinha travado. Então foi acionado o administrador do servidor para ver o que tinha ocorrido, e verificou-se que o processo de algum aluno tinha ficado travado, fazendo que o sistema não iniciasse mais nenhum processo de execução ou depuração. Então o servidor `campeche` foi reiniciado e o sistema voltou a funcionar. Devido ao problema ocorrido, o autor e o professor Fernando dos Santos chegaram à conclusão que era preciso prorrogar o prazo da entrega do exercício e deixar outra alternativa de ambiente para os alunos fazerem o exercício, caso o WebIde 2.0 travasse novamente. O prazo da entrega do exercício foi adiado para o dia vinte e quatro de maio de dois mil e dez. No dia vinte e um de maio foi encontrado o

problema no WebIde 2.0, que era uma recursividade infinita no código de leitura das variáveis durante uma depuração, a recursividade existia para pegar todos as subvariáveis das variáveis, e quando vinha uma variável que era uma classe, e essa classe tinha um atributo que era ela mesma, o problema ocorria. Para corrigir, foi decidido retirar a recursividade, e carregar somente a primeira hierarquia de variáveis de uma classe. Após a correção foi atualizado no servidor `campeche`.

Quando algum aluno finalizava o exercício, o professor ou monitor abria o exercício para corrigí-lo. Se encontrasse algum problema na resolução do exercício, era apontado o problema para o aluno através do comentário individual, dizendo o porque do erro, e se necessário, escrevia alguma dica para resolver o problema. No dia vinte e quatro de maio de dois mil e dez, as vinte e duas horas, foi finalizado o exercício pelo professor Fernando.

No Quadro 31 é apresentado o enunciado do exercício aplicado.

```
     Criar um programa que calcule o salário líquido de funcionários. Para
cada funcionário o programa deve ler as seguintes informações: nome,
sexo(M ou F), quantidade de horas trabalhadas(pode ser informado horas
fracionadas (ex: 4.5)), salário por hora(o salário hora pode conter
centavos), número de dependentes.
     Seu programa deve ler estas informações para exatamente 10
funcionários, usando uma repetição for.
     Para cada funcionário lido, o programa deve calcular e mostrar (não é
necessário criar métodos): salário bruto, desconto do instituto nacional
de seguridade social (INSS), desconto do imposto de renda (IR), salário
líquido.
     Ao final, o programa deve exibir um relatório, com as seguintes
informações: quantas pessoas do sexo 'M' possuem salário líquido maior que
R$ 1000.00, quantas pessoas do sexo 'F' possuem mais que 3 dependentes,
quantas pessoas tem o desconto do IR maior que R$ 500.00, quantas pessoas
tem o salário hora maior que R$ 100.00, total de horas trabalhadas das 10
pessoas e qual o nome da pessoa com o maior salário líquido.
FÓRMULAS PARA CÁLCULO:
Salário bruto
horas trabalhadas * salário hora + (50 * número de dependentes)

Desconto INSS
salário bruto <= 1000   INSS=salário bruto * 8.5/100
salário bruto > 1000    INSS=salário bruto * 9/100

Desconto IR
salário bruto <= 500           IR=0
salário bruto > 500 e <= 1000  IR=salário bruto*5/100
salário bruto > 1000           IR=salário bruto*7/100
```

Quadro 31 – Enunciado do exercício aplicado

A turma era composta por quarenta e dois alunos, e desses vinte e três alunos fizeram o exercício. Dessa maneira foi realizada a validação do ambiente WebIde 2.0.

As conclusões do experimento são discutidas na próxima seção.

3.6 RESULTADOS E DISCUSSÃO

Os alunos não tiveram contato com o sistema antes do exercício e qualquer explicação de como usá-lo. Durante a validação nenhum aluno que usou o sistema fez algum tipo de questionamento sobre como abrir o exercício, ou como executar ou depurar, entre outras funcionalidades. Com isso acredita-se que a usabilidade está boa. Somente uma aluna ficou em dúvida se o botão de `Submeter` era para finalizar o exercício, com isso foi alterado a descrição do botão para `Submeter` e `Finalizar`.

Quanto à robustez pode-se concluir que não foi boa, pois houve alguns problemas com editor de código. Por exemplo, às vezes ao salvar o código fonte, apareciam *tags* HTML no meio do código. Outro problema era a numeração de linhas que não estava acompanhando o *scroll* do editor, confundindo, em alguns casos, os alunos quando tinha algum erro de compilação. Ambos os erros foram corrigidos após a fase de validação.

Não foi diagnosticado nenhum problema relacionado ao multiusuário. Problemas com compilação e execução não foram relatados, somente ocorreu um problema na depuração. Os navegadores como Internet Explorer, Firefox, Chrome e Safari são compatíveis com o sistema.

Um ponto importante encontrado durante os exercícios foi a dificuldade da interpretação da mensagem de erro de compilação do exercício pelo aluno. Muitos achavam que era erro do sistema e enviavam e-mails para o autor desse trabalho que respondia para os alunos explicando o que seria o erro. Para amenizar o problema, o professor Fernando dos Santos pediu para o autor adicionar na ajuda do sistema, os erros mais comuns de compilação que estavam ocorrendo entre os alunos, explicando o porquê o erro ocorria e como poderia corrigí-lo, conforme Figura 40.

Figura 40 – Tela da ajuda do sistema

Também foi melhorada a mensagem de erro em si, que é mostrado no Quadro 32 comparando a mensagem antiga com a nova.

Antigo Formato da Mensagem	Novo Formato da Mensagem
2: ')' expected if (rendaAnual > 17.000f { ^ 1 error	Linha do erro: 2 Mensagem do erro: ')' expected if (rendaAnual > 17.000f { ^ 1 error

Quadro 32 – Comparação com o antigo e novo formato da mensagem

Outro ponto era o sistema não notificar para o aluno quando o exercício era reaberto pelo professor ou monitor. Então, para avisá-los durante o processo de experimento o autor enviou e-mails avisando que o exercício tinha sido reaberto. Mais tarde foi implementado um notificador, que envia um e-mail para o aluno automaticamente quando seu exercício é reaberto.

No Quadro 33 são apresentados os resultados das ações que os usuários tomavam

durante o período de exercícios. Cada ação gerava um *log* com informações necessárias sobre esta ação e gravado em uma tabela no banco de dados do sistema.

Descrição	Resultado
1. Quantidade de alunos que usaram o ambiente	23 alunos
2. Média de tempo gasto pelos alunos para resolver e finalizar o exercício	81 horas
3. O maior tempo gasto por um aluno para resolver e finalizar o exercício	165 horas
4. O menor tempo gasto por um aluno para resolver e finalizar o exercício	1 hora
5. Quantidade de chamadas para abrir a ajuda de contexto	289
6. Quantidade de chamadas para execução	1465 execuções
7. Quantidade de chamadas para depuração	190 depurações
8. Quantidade de reabertura de exercícios	12 reaberturas
9. Quantidade de vezes que os exercícios foram salvos	673
10. Quantidade de vezes que foi visualizado o enunciado do exercício	394

Quadro 33 – Resultado das ações tomadas pelos alunos durante o período de exercícios

A média de tempo gasto (2) foi obtida a partir da primeira data que visitou o exercício até o momento da última finalização por tempo corrido, ou seja, em média os alunos levaram três dias para resolver o exercício. O maior (3) e o menor (4) tempo mostram os limites do fechamento do exercício, onde o maior tempo para esse fechamento foi de sete dias.

Em relação a utilização da ajuda (5), pode-se concluir que é de suma importância ter uma ajuda no sistema para auxiliar o aluno.

As quantidades de execuções (6) e depurações (7) tiveram uma grande diferença entre seus valores, podendo supor que os alunos usaram o comando executar para testar a compilação de seus programas, já que ambos os comandos compilavam antes de iniciar seus processos. Os alunos também usaram de forma considerável a depuração, em média oito depurações por aluno.

Os exercícios reabertos (8), em média foram zero ponto cinco por aluno, o que não é um valor muito alto, chegando a conclusão que cinquenta por cento dos alunos resolveu o exercício com êxito sem precisar reabrir.

A média de exercícios salvos (9) pelos alunos foi de vinte e nove. E da abertura do enunciado (10) foi de dezessete, supondo que o aluno teve dificuldades de interpretar o

enunciado do exercício.

Em relação às notas que os alunos tiraram usando o ambiente, o resultado foi muito bom, pois a média das notas foi de um ponto cinco, dito que na validação do exercício valia no máximo dois.

No Quadro 34 é apresentada uma comparação em alguns aspectos técnicos entre este trabalho e o trabalho correlato de Lohn (2008).

Comparação		
	WebIde 1.0	WebIde 2.0
Compilação de programas	X	X
Execução de programas	X	X
Depuração de programas	X	X
Alteração de variáveis em tempo de depuração	X	
Programação em Java	X	X
Ambiente integrado para criação, resolução e correção de exercícios	X	X
Suporte a *syntax highlighting*		X
Suporte a *content assist*		X
Interface rica de fácil usabilidade		X

Quadro 34 – Comparação as duas versões do WebIde

O uso da arquitetura JPDA para o depurador do sistema foi muito importante para extensões futuras. Somente usando o `jdb`, era preciso fazer *parse* das informações recebidas, com isso, seria difícil a criação de novos recursos para depurador, podendo gerar muitos erros e um código de manutenção complexa.

Inicialmente no desenvolvimento do *syntax highlighting*, foi usado JavaScript utilizando-se do JSNI. Mas houve muitos problemas com compatibilidade dos navegadores, o que fez com que esse caminho fosse abortado. Então foi utilizado um componente fornecido pelo EXT JS. Porém a API do Gwt-Ext tinha problemas ao integrar este componente, por exemplo, não funciona o *listener* de sincronizações do conteúdo de um `textarea` para um `iframe`, por este motivo o *syntax highlighting* funciona somente de forma manual.

O trabalho correlato de Branco e Schuvartz (2007) visa ensinar fundamentos de programação de computadores aos alunos usando técnicas de IA indicando o caminho mais apropriado para o nível do aluno. É divido em três módulos, sendo que o modulo tutor é onde

são apresentados os conceitos, exemplos, exercícios para o aluno através de interfaces. A ferramenta também corrige o exercício do aluno e verifica o nível de conhecimento para direciona-lo pelos conteúdos durante o ensino-aprendizagem. Enquanto, o presente trabalho, tem exercícios criados pelo professor, mas é preciso da intervenção do professor ou monitor para correção do exercício, onde eles indicam qual caminho o aluno deve seguir, dizendo onde ele errou através dos comentários individualizados.

O trabalho correlato de Moreira e Favero (2009) visa automatizar *feedbacks* imediatos de exercícios feitos por alunos, sem precisar da intervenção do professor, utilizando técnicas de similaridade, ou seja, o professor cria a resolução correta do exercício, e a ferramenta faz cálculos para saber a porcentagem de similaridade que as resoluções têm entre si. Esse tipo de técnica, o presente trabalho não possui, precisando ainda de um professor ou monitor para fazer esse papel.

4 CONCLUSÕES

O desenvolvimento deste ambiente web teve como objetivos desenvolver um ambiente baseando-se nos requisitos funcionais do trabalho de Lohn (2008), melhorando a usabilidade e robustez, a depuração com uma arquitetura que pudesse oferecer mais segurança e facilidade na implementação e oferecendo suportes básicos no editor de código fonte, como numeração de linhas, *syntax highlighting* e *content assist*.

O ambiente é voltado para o auxílio do aprendizado da programação, facilitando para o professor ou monitor acompanhar individualmente cada aluno, e também agilizando as correções dos exercícios e entregando o resultado mais rápido para o aluno.

Para o aluno, o ambiente facilita sua mobilidade, pois não é preciso instalar a ferramenta na máquina em que vai trabalhar, bastando acesso à internet. Com isso, abre-se a possibilidade de fazerem exercícios extraclasses, mesmo sem o professor ou monitor não estarem juntos.

Os objetivos tiveram as seguintes conclusões:

a) a criação de uma interface mais amigável usando o *framework* Gwt-Ext foi atingida com êxito;
b) o uso da arquitetura JPDA no processo de depuração também foi atingida com êxito;
c) os recursos de numeração de linhas e *syntax highlighting* foram atingidos com sucesso, somente o recurso de *content assist* tem sua disponibilidade limitada, não funcionando no navegador Internet Explorer.

Mesmo com os problemas que ocorreram durante o período de validação, o ambiente se apresentou estável, de fácil acesso e uso. O sistema teve uma ótima aceitação dos alunos e do professor Fernando, que utilizaram a ferramenta na disciplina de Programação de Computadores.

Os estudos apresentados das bibliotecas, *framework* e arquitetura foram importantes para o desenvolvimento deste trabalho. Não apresentaram nenhum problema ou dificuldade para implementação do sistema. A especificação da arquitetura JPDA era clara e objetiva, o que fez com que o desenvolvimento da depuração atingisse seu funcionamento desejado. Ela abstrai totalmente as informações de uma depuração, sem precisar conhecer a estrutura das informações que o depurador da linguagem oferece.

4.1 EXTENSÕES

Como extensões da ferramenta segurem-se:

a) desenvolver a técnica *Sand Box*[9], para capturar processos que podem travar o servidor;
b) desenvolver enunciado inteligente, onde o professor possa criar regras de programação no exercício, e a aplicação guie o aluno a programar no caminho correto. Um exemplo seria o exercício precisar de uma estrutura *for* em um algum ponto do código para resolver sua resolução;
c) desenvolver um assistente, para tratar erros de compilação, e explicar para o aluno porque esse erro pode ocorrer e como ele pode ser corrigido. Ele seria apto a capturar erros que o compilador gera e procurar em uma base de dados sobre o erro ocorrido;
d) desenvolver mais funcionalidades na depuração, como colocar *breakpoints*, mudar valor da variável, executar expressões ou fazer *drop to frame*[10]. Pois a implementação da arquitetura JPDA para o Java já fornece essas funcionalidades;
e) desenvolver maneiras para o professor e o monitor reutilizarem os seus comentários para problemas comuns encontrados nos exercícios dos alunos;
f) desenvolver funcionalidades para programação colaborativa, como bate papo, versão de código fonte, e melhorar o ambiente para suportar edição do código fonte simultâneo;
g) desenvolver extensão para criação de métodos no exercício;
h) desenvolver o uso do *syntax highlighting* de forma automática;
i) desenvolver uma forma de disponibilizar uma URL para executar o programa desenvolvida por um aluno;
j) desenvolver um mecanismo de correção automática dos exercícios usando as ideias do trabalho de Moreira e Favero (2009).

[9] *Sand Box* é um mecanismo de segurança para separação de programas em execução, frequentemente usado para executar código não testado (SANDBOX, 2010).
[10] Segundo Eclipse (2010), *drop to frame* é um comando para reentrar no *stack frame* selecionado.

REFERÊNCIAS BIBLIOGRÁFICAS

BRANCO, Wilson C.; SCHUVARTZ, Aguinaldo. Ferramenta computacional de apoio ao processo de ensino-aprendizagem dos fundamentos de programação de computadores. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 18., 2007, São Paulo. **Anais...** São Paulo: Sociedade Brasileira de Computação, 2007. Não paginado. 1 CD-ROM.

BREAKPOINT. In: WIKIPEDIA, the free encyclopedia. [S.l.]: Wikimedia Foundation, 2010. Disponível em: <http://en.wikipedia.org/wiki/Breakpoint>. Acesso em: 1 jun. 2010.

CALL stack. In: WIKIPEDIA, the free encyclopedia. [S.l.]: Wikimedia Foundation, 2010. Disponível em: <http://en.wikipedia.org/wiki/Call_stack>. Acesso em: 1 jun. 2010.

FRONT-END e back-end. In: WIKIPEDIA, the free encyclopedia. [S.l.]: Wikimedia Foundation, 2010. Disponível em: <http://pt.wikipedia.org/wiki/Front-end_e_back-end>. Acesso em: 11 jul. 2010.

DEITEL, Paul J.; DEITEL, Harvey M. **Ajax, rich internet applications e desenvolvimento web para programadores**. Tradução Célia Taniwaki e Daniel Vieira. São Paulo: Pearson, 2009.

ECLIPSE. **Eclipse documentation**: java development user guide. [S.l], [2010]. Disponível em: <http://help.eclipse.org/galileo/index.jsp?topic=/org.eclipse.jdt.doc.user/reference/views/debug/ref-droptoframe.htm>. Acesso em: 1 jun. 2010.

EXT JS. **Ext JS**: cross-browser rich internet application framework. [S.l], [2010]. Disponível em: <http://www.extjs.com/products/js/>. Acesso em: 20 maio 2010.

FURLAN, José D. **Modelagem de objetos através da UML**. São Paulo: Makron Books, 1998.

GARCIA JR, Jesus D. **Ext JS in action**. [S.l.]: Manning Publications Co, 2009. Disponível em: <http://www.manning-sandbox.com/ann.jspa?annID=46>. Acesso em: 20 maio 2010.

GESSER, Julio V. **Javaparser**. [S.l], [2010]. Disponível em: <http://code.google.com/p/javaparser/>. Acesso em: 30 maio 2010.

GONDIM, Halley W. A. S.; AMBRÓSIO, Ana P.; COSTA, Fábio M. Uma Experiência no ensino de algoritmos utilizando ambientes visuais de programação 3D. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 29., 2009, Bento Gonçalves. **Anais...** Bento Gonçalves: Sociedade Brasileira de Computação, 2009. Não paginado. 1 CD-ROM.

GOOGLE. **Google web toolkit**: product overview. [S.l], [2010a]. Disponível em: <http://code.google.com/intl/pt-BR/webtoolkit/overview.html>. Acesso em: 20 maio 2010.

_____. **Google web toolkit**: coding basics. [S.l], [2010b]. Disponível em: <http://code.google.com/intl/pt-BR/webtoolkit/doc/1.6/DevGuideCodingBasics.html>. Acesso em: 20 maio 2010.

_____. **Google web toolkit**: compiling & debugging. [S.l], [2010c]. Disponível em: <http://code.google.com/intl/pt-BR/webtoolkit/doc/1.6/DevGuideCompilingAndDebugging.html>. Acesso em: 20 maio 2010.

JIVAN, Sanjiv. **Gwt-Ext**. [S.l], [2008]. Disponível em: <http://www.gwt-ext.com/wiki/index.php?title=Main_Page>. Acesso em: 22 maio 2010.

LOHN, Silvano. **Ambiente na web para execução e depuração de programas com sintaxe Java**. 2008. 88 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

MECENAS, Ivan; OLIVEIRA, Vivianne. **Banco de dados**: do modelo conceitual à implementação física. Rio de Janeiro: Alta Books, 2005.

MOREIRA, Mireille P.; FAVERO, Eloi L. Um ambiente para ensino de programação com feedback automático de exercícios. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 29., 2009, Bento Gonçalves. **Anais...** Bento Gonçalves: Sociedade Brasileira de Computação, 2009. Não paginado. 1 CD-ROM.

HINTERHOLZ JR, Ornélio. Tepequém: uma nova ferramenta para o ensino de algoritmos nos cursos superiores em computação. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 29., 2009, Bento Gonçalves. **Anais...** Bento Gonçalves: Sociedade Brasileira de Computação, 2009. Não paginado. 1 CD-ROM.

MOREIRA, Thiago D. R. G. Sistemas tutores inteligentes. In: OLIVEIRA JR., H. A. (Coord.). **Inteligência computacional:** aplicada à administração, economia e engenharia em Matlab. Thomson Learning, 2007. p. 265-280.

SANDBOX computer security. In: WIKIPEDIA, the free encyclopedia. [S.l.]: Wikimedia Foundation, 2010. Disponível em: <http://en.wikipedia.org/wiki/Sandbox_(computer_security)>. Acesso em: 1 jun. 2010.

SOUTO, Aletéia V. M.; DUDUCHI, Marcelo. Um processo de avaliação baseado em ferramenta computadorizada para o apoio ao ensino de programação de computadores. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 29., 2009, Bento Gonçalves. **Anais...** Bento Gonçalves: Sociedade Brasileira de Computação, 2009. Não paginado. 1 CD-ROM.

SUN MICROSYSTEMS. **Java platform debugger architecture**: architecture. [S.l], [2010a]. Disponível em: <http://java.sun.com/javase/6/docs/technotes/guides/jpda/architecture.html>. Acesso em: 20 maio 2010.

_____. **JDK tools and utilities**. [S.l], [2010b]. Disponível em: <http://java.sun.com/javase/6/docs/technotes/tools/index.html>. Acesso em: 20 maio 2010.

UCHÔ, Elvira M. A.; MELO, Rubens N. Integração de sistemas de banco de dados heterogêneos usando frameworks. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, 14., 1999, Florianópolis. **Anais...** Florianópolis: Sociedade Brasileira de Computação, 1999. Disponível em: <http://www.inf.ufsc.br/sbbd99/anais/SBBD-Completo/32.pdf >. Acesso em: 08 set. 2009.

VAHLDICK, Adilson; LOHN, Silvano. Ambiente web para execução e acompanhamento de exercícios de programação. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 19., 2008, Fortaleza. **Anais...** Fortaleza: Sociedade Brasileira de Computação, 2008. Não paginado. 1 CD-ROM.

XAVIER, Gláucia. M. C. Estudo dos fatores que influenciam a aprendizagem introdutória de programação. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 15., 2004, Manaus. **Anais...** Manaus: Sociedade Brasileira de Computação, 2004. Não paginado. 1 CD-ROM.