UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

JEduc: reflexão sobre a Linguagem Java na Educação

por

CÁSSIA ALVES PEREGO

Dissertação submetida à avaliação,
como requisito parcial para a obtenção do grau de
Mestre em Ciência da Computação

Profa. Dra. Maria Lúcia Blanck Lisboa
Orientadora

Porto Alegre, dezembro de 2002.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Perego, Cássia Alves
JEduc: reflexão sobre a Linguagem Java na Educação / por Cássia Alves Perego. – Porto Alegre: PPGC da UFRGS, 2002.
90 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR – RS, 2002. Orientadora: Lisbôa, Maria Lúcia Blanck.

1. Java. 2. Ensino de programação. 3. Linguagens de Programação. I. Lisbôa, Maria Lúcia Blanck. II. Título.

Agradecimentos

Embora uma dissertação seja, pela sua finalidade acadêmica, um trabalho individual, há contribuições diversas que não podem e nem devem deixar de ser realçadas. Por essa razão, desejo expressar os meus sinceros agradecimentos.

Primeiramente o amor, a compreensão e, às vezes, o consolo recebido do Paulo, pois ficar por dois anos e meio com uma esposa ausente não é nada fácil... Aos meus pais, por toda a dedicação e incentivo que sempre estiveram presentes, ensinando-me a importância da construção e coerência de meus próprios valores. Aos meus dois irmãos, pelo simples fato deles existirem. Um agradecimento muito especial à minha mãe, que ainda fica exultante com os "feitos" da filha e torce por suas pequenas e sofridas conquistas, que aos olhos dela são sempre grandes vitórias. Minha esperança é que, compensando o tempo e esforço despendidos, algumas experiências vividas durante o mestrado venham por ajudar a mim mesma a identificar maneiras adicionais de enriquecer suas vidas.

Minha seleção, no âmbito acadêmico, começa pela excelência profissional da Profa. Lúcia Lisboa, que conferiu prestígio e valor ao meu trabalho de mestrado. Além de ter dividido o seu conhecimento, também assumiu o papel de amiga, demonstrando bastante abertura de espírito desde a primeira aula na UEL, que logo abriu a porta que rapidamente me encaminharia para o tema tratado nesta dissertação. Foram fundamentais a disponibilidade revelada ao longo deste ano e meio, as críticas e sugestões relevantes feitas durante a orientação. Espero ter retribuído, com a seriedade de meu trabalho, a confiança em mim depositada.

Incluo também a Silvia, por sua generosidade desde o nosso primeiro contacto, pela disponibilidade e amizade então demonstradas. Ao longo destes meses, nas nossas diversas trocas de *e-mail*, ela revelou o exemplo perfeito do espírito de partilha. O nome do Telmo também consta nesta lista seletiva. Foi sorte cruzar com eles nesta etapa de conclusão de mestrado, suas idéias permearam meu trabalho. Sem este apoio na fase de implementação, acho que o ambiente JEduc seria somente um sonho. Muito obrigada, Silvia e Telmo!!

A todos os colegas do CPD e da FIPP que sempre se disponibilizaram a acertos necessários de horário para que eu pudesse cumprir com todas as minhas obrigações profissionais e acadêmicas. Aos colegas do Mestrado pela excelente relação pessoal que criamos e espero que não se perca. Em especial à Aglaê, à Melissa e ao Silvio pela companhia e apoio nos momentos bons e menos bons, e pela amizade.

Quero agradecer à UNOESTE pelo apoio financeiro e à Faculdade de Informática de Presidente Prudente (FIPP) pelas dispensas para as viagens, que foram muito importantes para que eu pudesse realizar meus estudos e obter o grau de Mestre em Ciência da Computação.

Enfim, a todos os professores do Instituto de Informática da UFRGS, pelos conhecimentos transmitidos, pois tive a oportunidade e o privilégio de renová-los e vivenciar novas situações, que se tornaram lições valiosas para toda uma vida.

Sumário

Lista de Abreviaturas

API	Application Program Interface
AWT	Abstract Windowing Toolkit
GUI	Graphical User Interface
HTML	HyperText Markup Language
IDE	Integrated Development Environment
IDECoRe	Integrated Development Environment Core Reuse
JDK	Java Development Kit
JRE	Java Runtime Environment
JVM	Java Virtual Machine
Mac	Macintosh Computer
PC	Personal Computer
SBC	Sociedade Brasileira de Computação

Lista de Figuras

Lista de Tabelas

Resumo

Neste estudo são discutidos alguns aspectos relacionados à escolha da primeira linguagem de programação em currículos de ciência da computação, com interesse especial em Pascal e Java. A primeira linguagem é amplamente adotada para ensinar programação aos novatos, enquanto a segunda está ganhando popularidade como uma linguagem moderna e abrangente, que pode ser usada em muitas disciplinas ao longo de um curso de graduação em computação como ferramenta para ensinar desde recursos básicos de programação até tópicos mais avançados.

Embora vários problemas quanto ao ensino de Java, como a primeira linguagem de programação, possam ser apontados, consideramos que Java é uma boa escolha, visto que (a) oferece apoio a importantes questões conceituais e tecnológicas e, (b) é possível contornar algumas complexidades da linguagem e da plataforma Java para torná-las mais adequadas à alunos iniciantes. Além disso, considerando a grande popularidade de Pascal nos currículos de cursos de computação, uma eventual adoção de Java conduz à outro problema: a falta de professores aptos a lecionar programação orientada a objetos. Sugerimos que este problema de migração de Pascal para Java seja enfrentado através de simplificação do ambiente de desenvolvimento de programas, uso de um pacote com classes que facilitam a entrada e saída, e o desenvolvimento de um catálogo comparativo de programas implementados em ambas as linguagens.

Neste estudo também é apresentado o JEduc, um IDE muito simples com o objetivo de dar suporte ao ensino da linguagem de programação orientada a objetos Java aos novatos. Oferece componentes desenvolvidos em Java que integram edição, compilação e execução de programas Java. Além das funcionalidades comuns a um IDE, JEduc foi desenvolvido para agir como uma ferramenta pedagógica: simplifica a maioria das mensagens do compilador e erros da JRE, permite a inserção de esqueletos de comandos, e incorpora pacotes especiais para esconder alguns detalhes sintáticos e semânticos indesejáveis.

Palavras-chave: linguagem de programação Java, ensino de orientação a objetos, introdução à programação, ambiente de apoio ao ensino, JEduc

TITLE: "JEDUC: REFLECTION ABOUT THE JAVA LANGUAGE IN EDUCATION"

Abstract

In this study some aspects related to the choice of the first programming language in computer science curricula are discussed, with special interest in Pascal and Java. The first language has been widely adopted to teach programming to novices, while the second is gaining popularity as a modern language and wide-ranging, that can be used in many disciplines throughout an undergraduate computer science program as a tool to teach since basic programming skills to more advanced computer science topics.

Although some problems related to the adoption of Java as the first programming language can be pointed, we argue that Java is a good choice because: (a) it supports important conceptual and technological issues, and (b) it is possible to bypass some complexities of Java language and platform in order to make them more suitable to beginners. Besides this, and considering the great popularity of Pascal in computer science curricula, if happens to turn from Pascal to Java we have to face another problem: the lack of skilled lecturers to teach object-oriented programming. We suggest to address this problem by simplifying the first approach to Java by means of a simple IDE, the use of a package with classes that facilitate input and output, and also by making available to Pascal lecturers a comparative catalogue of programs implemented in both languages to softer the migration from Pascal to Java.

We also present JEduc, a very simple IDE intended to support the teaching of Java object-oriented programming language to beginners. It offers Java written components which integrate editing, compiling and execution of Java programs. Besides the common IDE functionalities, JEduc was developed to act as a pedagogical tool: it simplifies most of the compiler and JRE error messages, it permits the insertion of command skeletons, and it incorporates special packages to hide some undesirable syntactic and semantic details.

Keywords: Java programming language, object-oriented teaching, introduction to programming, teaching support environment, JEduc

1 Introdução

A visão clássica de ensino, ou seja, aquela em que o professor exerce o papel ativo de provedor de todo o conhecimento, e o aluno o papel passivo de recebê-lo, está se transformando rapidamente sob o peso das novas tecnologias. O novo modelo é centrado no aluno, no qual ele passa a ter um papel muito mais ativo e autônomo na busca do aprendizado e do conhecimento, não apenas na área de computação. No entanto, o ensino de programação, objeto deste estudo, ainda se encontra bastante vinculado às linguagens de programação em uso há mais de 25 anos, projetadas antes do advento de diversas tecnologias hoje populares. Com a Internet, particularmente, têm surgido novos paradigmas de intercâmbio de informação, e suas surpreendentes possibilidades estão capturando a imaginação e interesse do mercado e dos educadores ao redor do mundo, levando-os a repensar a natureza do ensino e da aprendizagem de programação.

Neste capítulo estão abordados, de forma sintética, os aspectos que motivaram e deram origem a todo este estudo, bem como os objetivos a serem atingidos e a forma como o texto desta dissertação de mestrado está organizado.

1.1 Motivação

Na tentativa de resolver, pelo menos em parte, os problemas relacionados à constante evolução tecnológica, e, com o objetivo de facilitar e aprimorar o aprendizado, os pesquisadores estão começando a perceber a necessidade de buscar uma alternativa para o modelo tradicional do ensino de programação nos diversos cursos de computação.

Acredita-se que a abordagem de programação orientada a objetos é bastante apropriada para ser ensinada em disciplinas introdutórias de programação por apoiar-se em conceitos já bastante amadurecidos no meio acadêmico e na comunidade de programação, em seu sentido amplo [KOL 99]. Em particular, a linguagem Java vem apresentando notável penetração e oferece, além de uma implementação orientada a objetos apoiada em uma ampla biblioteca de classes executáveis em várias plataformas, a possibilidade da gratuidade de ferramentas básicas de desenvolvimento: compilador, ambiente de execução e depurador, entre outros.

Em especial, é importante notar que esta linguagem de uso geral pode ser usada em diferentes contextos de programação, e, se o aluno dominar a linguagem poderá aprofundar-se em outras particularidades para usá-la em disciplinas de temas específicos, como programação visual, redes, sistemas distribuídos, tolerância à falhas, engenharia de *software*, etc. Disto decorre que esta linguagem pode ser usada ao longo de todo um curso de graduação.

Ainda poucos são os esforços no desenvolvimento de *softwares* que possam ser utilizados no ensino de tópicos relacionados com a área da computação, considerando-se a alta demanda de profissionais bem qualificados. Existem inúmeros trabalhos e projetos que utilizam a filosofia de software livre para o ensino das mais diversas áreas do conhecimento [FRE 2001] [FRE 2002] [BRO 2002]. Embora várias disciplinas que fazem parte de um curso de graduação em computação utilizem programação, este estudo dará atenção às disciplinas introdutórias de programação. Existe uma grande

preocupação dos professores dessas disciplinas com a adoção de uma linguagem de programação que satisfaça tanto aspectos teóricos e didáticos quanto técnicos, e que também seja de fácil aprendizado para o aluno.

A linguagem Java enfatiza idéias de flexibilidade, dinamismo e segurança. A flexibilidade pode ser notada pelos diferentes tipos de aplicações que podem ser desenvolvidas, tais como aplicações centralizadas, concorrentes, distribuídas e aplicações cliente-servidor, executáveis em navegadores na Internet. O dinamismo deve-se à sua filosofia de implementação, na qual o núcleo básico da linguagem é oferecido na forma de um pequeno conjunto de tipos e comandos, sendo as demais construções oferecidas através de bibliotecas de componentes, permitindo que a expressividade da linguagem seja facilmente ampliada pela adição de novas bibliotecas. O dinamismo também se expressa pela carga dinâmica de componentes em sua Máquina Virtual de Execução - JVM, independente da plataforma de hardware. A segurança traduz-se pela obrigatoriedade de inspeção de código carregado pela JVM, automaticamente realizada pelos seus carregadores de classes, os quais definem e implementam diversos níveis de normas de segurança.

Além disso, incentiva-se seu uso como linguagem inicial por possuir uma especificação única, aberta, clara e precisa [BRU 2002a], complementada por vasta documentação de suas classes e dos pacotes padrão. A agregação de Java com um grupo de tecnologias relacionadas a ela, tem estendido substancialmente as capacidades de programação em domínios específicos, como por exemplo, aplicações móveis.

Porém, são encontrados vários problemas referentes ao ensino de Java como a primeira linguagem de programação. Esses problemas podem ser reduzidos através do ocultamento de algumas complexidades de Java, proporcionando assim um ambiente de desenvolvimento de programas condizente com o de uma linguagem que deverá servir de apoio ao ensino de programação. A proposta alvo deste estudo é a criação um IDE (*Integrated Development Environment*) munido de um conjunto de pacotes de classes "simplificadoras" que auxiliem o estudante a escrever programas rapidamente, sem exposição a estas complexidades. Para o desenvolvimento destas classes foi usada como base de referência a linguagem Pascal. Com esta base de referência, procura-se um modelo para amenizar a sintaxe de Java, principalmente quanto à entrada e saída, bem como ilustrar as semelhanças da solução de problemas (implementação estruturada de métodos) e as diferenças (organização procedimental x organização de classes) entre ambas.

Existem duas questões importantes ao redor da utilização de Java (a) o que ensinar? e (b) como ensinar? Há problemas significantes em sua utilização como linguagem de programação introdutória, mas é uma linguagem que está amadurecendo, e com o passar do tempo suas ferramentas e recursos serão melhorados. O objetivo principal em disciplinas introdutórias de programação é mostrar ao aluno as formas possíveis de solucionar problemas com o desenvolvimento programas, e não apenas programar em Java. Este estudo não propõe nenhuma seqüência ou forma de apresentação dos conteúdos, com vistas ao paradigma orientado a objetos (isso fica a critério do professor da disciplina), mas sim identificar pontos de complexidade, propor soluções e implementá-las em um ambiente de desenvolvimento simples.

Visando qualificar e quantificar o universo a ser atingido, no caso, quais e quantos cursos de graduação em informática e afins utilizam Pascal como linguagem introdutória à programação, quais e quantos já adotam Java como linguagem introdutória ou como outra linguagem ensinada no curso, foi conduzida uma pesquisa

via lista de assinantes da Sociedade Brasileira de Computação solicitando estes dados. Os resultados desta pesquisa estão no Anexo 1.

1.2 Objetivos

Estudar o processo inicial do desenvolvimento de programas dentro do ambiente Java, de forma a compreender as vantagens e a aplicabilidade das diversas classes da API (*Application Program Interface*) padrão, e identificar possíveis problemas e soluções na migração de paradigmas e linguagens, neste caso de Pascal para Java. Não faz parte do escopo deste estudo determinar o conteúdo programático e ferramentas específicas com vistas à adoção da linguagem Java nas disciplinas introdutórias de programação, mas sim mostrar que é possível reduzir tanto a complexidade da linguagem quanto das ferramentas – compiladores, ambientes de execução e de desenvolvimento – para auxiliar o ensino introdutório de programação com Java, quando, por fatores particulares, isso se tornar realidade.

A concretização destas propostas é realizada através do ambiente JEduc [PER 2002b] e um pacote de classes cuja idéia central é simplificar, de forma transparente, algumas características da linguagem Java, no intuito de agilizar o processo de desenvolvimento dos primeiros programas orientados a objetos pelos alunos. No entanto, houve também uma preocupação com a capacitação dos professores para a realização desta migração, do paradigma procedimental para o orientado a objetos. Como conseqüência, foi desenvolvida uma série de anotações especialmente destinadas aos professores de Pascal, que apontam divergências, sejam sintáticas ou sejam semânticas, entre Java e Pascal, bem como um catálogo de exemplos de programação. O objetivo é fornecer suporte relacionado aos principais problemas que eles encontrarão na transcrição dos exemplos utilizados com a linguagem Pascal para os exemplos com a respectiva implementação em Java. Não é defendida aqui a idéia que esta é a abordagem mais adequada, mas sim reconhecida a forma de aprendizagem por comparação do novo com o conhecimento já adquirido, enfatizando semelhanças e diferenças conceituais. Ou seja, pode-se reutilizar e adaptar conhecimentos.

1.3 Estrutura do texto

Esta dissertação está organizada em 6 capítulos, são eles: Introdução, Base Conceitual, Ensino Introdutório com o Paradigma Orientado a Objetos e a Linguagem Java, Proposta de Simplificação, JEduc: ambiente dedicado ao ensino de Java e Conclusões, respectivamente, e 2 Anexos.

O capítulo 2 descreve inicialmente os aspectos envolvidos e a maneira como ocorre o ensino introdutório de programação. Descreve os componentes que todo ambiente integrado de desenvolvimento deve possuir. Explica os princípios sob os quais a ferramenta IDECoRe foi idealizada, projetada e implementada. E, finalmente, aborda as características da reflexão computacional, destacando as que foram exploradas neste estudo.

No capítulo 3 são discutidos os aspectos mais relevantes no ensino introdutório de programação considerando o paradigma orientado a objetos, e Java como a linguagem de programação onde os exemplos serão implementados. Planos de ensino que usam Pascal e Java são mostrados, visando possibilitar uma comparação entre o

paradigma procedimental e o orientado a objetos. São consideradas também as soluções propostas e a forma com que elas afetam professores e alunos no ensino deste novo paradigma.

No capítulo 4 são apontadas algumas complexidades encontradas na linguagem Java, entre elas, a entrada e saída de valores via console do sistema. É apresentado um pacote de classes para facilitar, ou até mesmo ocultar, o ensino destas complexidades à alunos novatos. A ajuda disponível aos professores também está amplamente descrita.

O capítulo 5 descreve as características do ambiente JEduc que foram implementadas para adequá-lo como ferramenta de apoio ao ensino de programação, identifica também as adequações decorrentes da derivação da ferramenta IDECoRe. O ambiente JEduc encontra-se em fase final de testes e está disponibilizado, juntamente com o pacote das classes "simplificadoras" e o catálogo de exemplos.

No capítulo 6 estão as conclusões, as sugestões para trabalhos futuros e as principais contribuições deste estudo.

O Anexo 1 mostra, na forma de tabela, os dados obtidos na pesquisa realizada sobre a primeira linguagem de programação usada nas universidades brasileiras.

Finalmente o Anexo 2 exibe o plano de ensino usado pela UFPE (Universidade Federal de Pernambuco) na disciplina introdutória de programação

2 Base Conceitual

Este capítulo descreve inicialmente os aspectos envolvidos e a maneira como ocorre o ensino introdutório de programação. Descreve os componentes que todo ambiente integrado de desenvolvimento deve possuir. Explica os princípios sob os quais a ferramenta IDECoRe - base de derivação do ambiente JEduc - foi idealizada, projetada e implementada. E, finalmente, aborda as características da reflexão computacional, destacando as que foram exploradas neste estudo.

2.1 Ensino introdutório de programação

A contínua expansão do alcance da informática faz com que o computador se torne cada vez mais comum e atinja um número maior de pessoas. Nessa diversidade de aplicações destaca-se como relevante a figura do profissional de programação, por seu papel ativo nesse processo. Como requisitos fundamentais à formação do profissional de programação estão a desenvoltura lógica e a correta construção de algoritmos. Particularidades relativas à abstração estão relacionadas com o desenvolvimento e aplicação da lógica [FOR 2000] no processo de solução de problemas.

O uso dos computadores no suporte às atividades de ensino superior aumentou significativamente nos últimos anos. Com isso, as instituições educacionais devem procurar preparar alunos e professores para realizar o progresso no campo da informática que evolui rapidamente [LIS 2002], assim como reconhecer a importância da atualização tecnológica e pedagógica. Os alunos necessitam desenvolver habilidades que os permitam evoluir da tecnologia de hoje para os desafios do futuro.

O processo de aprender a programar é difícil e árduo para a maioria dos alunos. Mesmo a aprendizagem de conceitos básicos e a sua aplicação na resolução de problemas concretos, introduz dificuldades para muitos alunos, sendo a principal delas relacionada com um estudo baseado na prática. O objetivo inicial e fundamental do ensino introdutório de programação é trazer compreensão e naturalidade à lógica de programação. A preocupação principal não está na codificação dos algoritmos desenvolvidos, mas, sim, com um alicerce sólido que possibilite a migração para qualquer linguagem de programação.

A aplicação de noções de lógica, aliada às técnicas de programação, auxiliam no desenvolvimento da coerência e racionalidade no aluno no momento de solucionar problemas sob a forma de programas. Lógica de programação significa o uso correto das leis do pensamento e de processos de raciocínio e simbolização formais na programação, objetivando a racionalidade e o desenvolvimento de técnicas que cooperem para a produção de soluções logicamente válidas e coerentes, que resolvam, com qualidade, os problemas que se deseja programar [GUI 85]. A lógica de programação pode ser representada por várias das inúmeras linguagens de programação existentes, desde que forneçam os mecanismos de abstração adequados.

Procurando se desvencilhar dos detalhes sintáticos complexos, as disciplinas introdutórias de programação utilizam abstrações para iniciar o ensino da programação, ou seja, neste primeiro instante a sintaxe de uma determinada linguagem de programação pode ser totalmente ignorada. Com isso, através de uma linguagem formal ou informal, é possível definir uma seqüência de passos ou um conjunto de abstrações

que visam solucionar um determinado problema. Somente numa próxima etapa as abstrações e os algoritmos são então implementados em alguma linguagem de programação.

O objetivo geral das disciplinas introdutórias de programação, pegando como exemplo a ministrada no Departamento de Física e Informática da USP [USP 2001], é introduzir conceitos básicos de computação para empregar em aplicações onde seja desejável, ou imprescindível, o uso dos recursos de informática. E como objetivos específicos podem: fornecer as distinções e características básicas dos elementos de *hardware* e dos tipos de *software* nos sistemas computacionais; utilizar editores de texto; tradução do conceito de ação; familiarizar-se com uma linguagem de programação de alto nível; desenvolver pequenas aplicações numéricas e não numéricas. Resumindo, disciplinas de introdução à programação visam capacitar o aluno a planejar e desenvolver pequenas tarefas com o auxílio de sistemas computacionais, dando assim as diretivas básicas para que esse desenvolvimento contribua eficientemente no seu campo profissional.

No decorrer do processo ensino-aprendizagem do modelo imperativo de programação é necessário, buscando validar a funcionalidade do algoritmo escrito pelo aluno, transcrevê-lo para a sintaxe da linguagem de programação definida para uso na disciplina, para que o aluno consiga compreender a execução de seu programa. Neste momento, entra uma característica que deve ser crucial na escolha da linguagem de programação a ser adotada com alunos iniciantes, a saber: a suavidade para a tradução do algoritmo para a sintaxe dessa linguagem.

2.2 Objetivos de disciplinas introdutórias

Comparando planos de ensino de disciplinas introdutórias nas universidades brasileiras [UNI 2001] [UNO 2001], voltados ao conceito de algoritmo que é a base conceitual do modelo imperativo de programação e independentemente da linguagem de programação adotada, percebe-se uma homogeneidade nos objetivos, nos tópicos abordados e na seqüência em que são apresentados aos alunos. Os objetivos específicos comumente encontrados nestes planos são:

a) expor ao aluno, de forma sucinta e natural, os conceitos para familiarização com a lógica e a estruturação de algoritmos;

b) abordar conceitos fundamentais que serão utilizados durante o decorrer da disciplina, tais como: tipo de informação (tipos de dados), constantes e variáveis, blocos lógicos e entrada e saída;

c) apresentar, em detalhes, cada uma das estruturas básicas de controle, das aplicações mais simples às mais complexas, procurando gradualmente esgotar o assunto, contando sempre com a utilização de muitos exemplos e exercícios. Esta etapa é considerada a de maior relevância, visto que simultaneamente aprimora e solidifica a desenvoltura lógica e que, em conjunto com as etapas anteriores, encerra o conhecimento mínimo e indispensável para a construção de programas;

d) tratar das diversas formas de representação e manipulação das informações, buscando a relação entre a informação em seu estado fluente, com estruturas capazes de comportá-la. É importante estabelecer, nesse momento, um

paralelo das definições com os algoritmos de manipulação de vetores, matrizes, registros e suas combinações;

e) estudar a manipulação de grandes quantidades de informação, em estruturas que são conhecidas por arquivos. Explorar os diversos conceitos que tratam tipos de arquivos seqüenciais, randômicos e, inclusive, indexados;

f) descrever a construção de subalgoritmos, ou seja, módulos que perfazem objetivos menores, mas que quando agrupados resolvem problemas complexos. Normalmente, são abordados contextos dos módulos (ação e resultado), escopo de variáveis e passagem de parâmetros.

Todos estes conceitos podem ser implementados em uma linguagem de programação orientada a objetos como Java (vide figuras 2.1 e 2.2), visto que este paradigma, por ser derivado do modelo imperativo, possui estes mecanismos mínimos de abstração e de estruturas de controle. Esta característica abre a possibilidade de adoção de Java sem adoção conceitual explícita do modelo de objetos.

Em um trabalho desenvolvido no Departamento de Informática e Ciência da Computação da UERJ (Universidade Estadual do Rio de Janeiro) [SOU 2001], foi constatada a suavidade requerida acima com o uso da linguagem Java e de duas classes que simplificaram a entrada e saída através do uso de caixas de diálogo existentes no pacote `java.swing` [DEI 2001]. Java trata os tipos primitivos como é feito na forma tradicional, sem o uso de objetos, exatamente como é encontrado nas linguagens imperativas. Já os outros tipos, como *strings* e os definidos pelo usuário, são implementados por classes, e suas instâncias são acessadas por referências. Para esclarecer melhor o trabalho da UERJ, mencionado acima, a seguir será ilustrado um exemplo adaptado de [SOU 2001]. A apresentação do exemplo foi organizada trazendo à esquerda a sintaxe do comando sob a forma algorítmica e a implementação em Java à direita (figura 2.1), e, depois, a codificação de um exemplo de programa para o comando em questão (figura 2.2).

```
SE condição
ENTÃO
   ações
SENÃO
   ações
FIM-SE
```

```
if (<condição>){
   comandos
}
else{
   comandos
}
```

FIGURA 2.1 - Exemplo de algoritmo e da respectiva sintaxe em Java

```
Início

Inteiro produto;
Real peso;
LEIA(produto);
LEIA(peso);
SE (peso>=100) ENTÃO
 ESCREVA(produto,peso,"aceito");

SENÃO
ESCREVA(produto,peso,"recusado");
FIM-SE


Fim
```

```
1  public class SeEntaoSenao{
2   public static void main(String args[]){
3    int produto;
4    double peso;
5    produto = teclado.lerInt("produto");
6    peso = teclado.lerDouble("peso");
7    if (peso>=100){
8      video.exibe(+produto+""+peso+" aceito");
9    }
10   else {
11    video.exibe(+produto+" "+peso+" recusado");
12   }
13   System.exit(0);
14  }
15 }
```

FIGURA 2.2 - Implementação do exemplo da figura 2.1

2.3 Ambientes integrados de desenvolvimento

Ferramentas e ambientes de programação que oferecem suporte à implementação de aplicativos são conhecidos pela sigla IDE - *Integrated Development Environment*, ou ambientes integrados de desenvolvimento. O objetivo principal dos IDEs é agrupar, em um único aplicativo, todas as ferramentas necessárias para o desenvolvimento de programas, proporcionando ao programador maior produtividade e eficiência. Eles costumam agregar ferramentas diversas, tais como compiladores, interpretadores, visualizadores, depuradores e editores de programas. A seguir são brevemente abordadas as características e finalidades de cada componente:

- compiladores: estão entre os tipos mais bem estudados de programas de computador. Para muitos compiladores tradicionais, o código fonte é traduzido diretamente para uma linguagem específica de máquina; em outros casos, a tradução é feita para uma linguagem intermediária;
- interpretadores: são similares aos compiladores, mas com uma importante diferença; um interpretador engloba ambas as atividades de tradução e execução. Uma pequena parte do código fonte, como um comando, é traduzido e executado, em seguida, outro é traduzido e executado, e assim por diante. Uma vantagem desta técnica é que ela elimina a necessidade de separar a fase de compilação, mas o programa geralmente apresenta lentidão na execução, pois o processo de tradução ocorre durante cada execução. A mais importante implicação da interpretação de código, é que os alunos receberão informações imediatas e precisas quando um programa falhar durante a execução [KIN 97]. No caso dos programas em Java, é possível mostrar toda a pilha de execução, descrevendo, com exatidão, qual comando causou a falha no programa;
- visualizadores: oferecem a representação pictórica de elementos estáticos e dinâmicos de um programa, tais como estruturas modulares, classes e seus relacionamentos, fluxo de execução de comandos, entre outros;
- depuradores: permitem executar o programa linha a linha e descobrir erros de lógica de programação muito facilmente. Normalmente possuem janelas de exibição de variáveis locais e globais, *buffers* de arquivos, árvore de objetos, avaliador de expressões, etc;
- editores de programas: usam comandos normais de edição, disponíveis na maioria dos produtos existentes. Estes editores normalmente permitem configurações de visualização, buscando atender as particularidades de cada usuário. Outros, desenvolvidos para uma determinada linguagem, possuem salientadores de palavras-chave e corretores automáticos de sintaxe.

Existem os mais diversos ambientes de desenvolvimento de programas Java, variando dos mais simples e gratuitos até ambientes profissionais com ferramentas complexas. Ao considerar a adoção desses ambientes para uso em disciplinas de introdução à programação, emergem algumas dificuldades, tais como:

- grande exigência de recursos de memória: devido principalmente ao grande número de componentes gráficos e recursos disponíveis;
- dificuldade de instalação: devido principalmente a um número elevado de componentes e propriedades a serem configuradas;

- excessiva quantidade de ícones e menus: a grande quantidade de elementos visuais sobrecarrega a interface com características desnecessárias para o propósito do ensino introdutório de programação;
- exigência de criação de "projetos": para o aluno iniciante em programação, torna-se um fator “complicador” adicional ter que trabalhar com o conceito de projetos. O aluno deve ter a oportunidade de visualizar apenas unidades simples de implementação.

Esses problemas são decorrentes, principalmente, do fato que esses IDEs estão voltados para o desenvolvimento de sistemas profissionais. Logo, isso os torna complexos para a maioria dos alunos que iniciam a aprendizagem da programação. Além disso, estes IDEs conduzem os usuários a evitar certos tipos de erros que fazem parte das etapas do amadurecimento em programação. Em alguns casos, são disponibilizadas versões educacionais destas ferramentas com redução no número de funcionalidades. Por outro lado, parece haver um consenso a respeito dos requisitos essenciais que uma ferramenta deve possuir para facilitar a aprendizagem de programação, entre eles:

- permitir a interação do aluno com o ambiente, de forma que ele possa atuar e desempenhar um papel ativo no processo de aprendizagem;
- seguir um padrão de interface visual comum à maioria das aplicações existentes, exigindo, assim, pouco tempo de aprendizagem e ao mesmo tempo tornando seu uso atrativo;
- possibilitar a instalação nos mais diversos sistemas operacionais e arquiteturas de hardware existentes atualmente;
- baixo custo de aquisição faz com que um elevado número de alunos e instituições adotem o produto como ferramenta didática.

2.3.1 Construção de ambientes adaptáveis

É inquestionável a necessidade de ferramentas que ofereçam suporte adequado aos desenvolvedores. Segundo Ossher [OSS 2000], é importante inovar na implementação de *software* e ferramentas, buscando facilitar tanto o desenvolvimento inicial quanto uma posterior adaptação e integração a novos contextos. A reutilização é uma das formas mais adotadas para produzir *software* de maneira rápida, com isso minimizando custos. A aplicação mais conhecida de reutilização consiste na engenharia de componentes. Construir aplicações a partir de componentes reutilizáveis surgiu como uma alternativa para a complexidade encontrada nas fases de desenvolvimento de *software* [PRE 95]. Os componentes possibilitam um aumento na reutilização e na qualidade do *software*, diminuem o tempo de desenvolvimento e direcionam o objetivo do desenvolvimento da aplicação [SZI 98]. Mas reutilizar componentes envolve alguns problemas: (a) em um grande conjunto de componentes é necessário identificar, rapidamente, aquele cujas funcionalidades atendam aos requisitos do sistema, (b) a adaptação de um componente deve preservar as funcionalidades pré-existentes, e garantir que falhas e erros não sejam introduzidos nessa adaptação [GRU 2000].

A primeira atitude que um desenvolvedor deve tomar, antes de iniciar o desenvolvimento de um *software,* é considerar a possibilidade de reutilizar componentes já desenvolvidos visando reduzir o tempo e o custo do desenvolvimento, aumentando conseqüentemente, a produtividade e a qualidade. Aplicando esses princípios ao

processo de desenvolvimento de um IDE simples e extensível para apoiar o desenvolvimento de programas, resultou o projeto de um conjunto de componentes denominado IDECoRe, a seguir descrito.

2.3.2 A ferramenta: IDECoRe

Com base no princípio de reutilização a ferramenta IDECoRe – *Integrated Development Environment Core Reuse* – foi desenvolvida. Sua implementação consiste na idéia de reutilizar não apenas componentes, mas uma ferramenta como todo, visando possibilitar a criação de ferramentas mais flexíveis e reutilizáveis, as quais foram denominadas derivações [BER 2002b]. Seu objetivo é claro: liberar o desenvolvedor do esforço de ter que identificar, adaptar e selecionar componentes que ofereçam os serviços básicos para a implementação de programas.

IDECoRe é uma ferramenta que pode ser reutilizada como núcleo, ponto de partida, para a construção de outros IDEs. Esse é o ponto fundamental que a distingue das demais ferramentas encontradas, tais como: JEdit, JBuilder™, Visual Age™, Delphi™, entre outras, as quais são voltadas para reutilização de componentes previamente definidos e não da ferramenta em si.

As funcionalidades básicas da ferramenta, bem como sua arquitetura, estão ilustradas na figura 2.3.

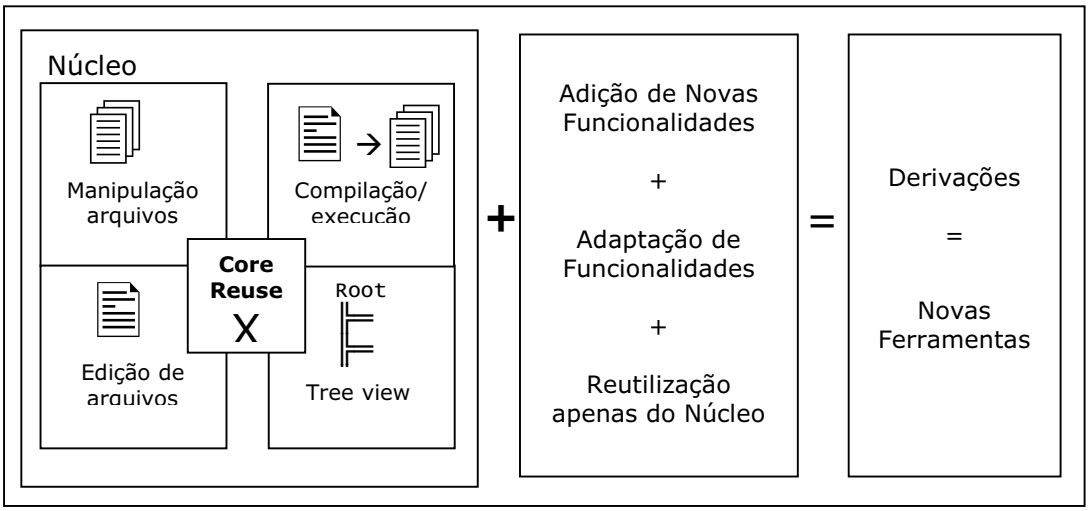


FIGURA 2.3 - Arquitetura da ferramenta IDECoRe

IDECoRe possui componentes que fornecem funcionalidades básicas, comuns à maioria dos IDEs, e permite que algumas características sejam customizadas de acordo com a linguagem que pretende-se utilizar. Seus principais componentes são:

- componentes para a manipulação de arquivos, cujas principais funcionalidades compreendem abrir, salvar e fechar arquivos;
- componentes para edição, os quais oferecem suporte para as ações de copiar, colar e recortar textos;

™ Borland, IBM, Borland, respectivamente.

- componentes para compilação e execução de programas, interrelacionados com os componentes usados para capturar as mensagens geradas pelo compilador e/ou interpretador;
- componentes para exibir, sob a forma de árvore (*tree view*), bibliotecas de classes e arquivos;
- componentes GUI – *Graphical User Interface* – responsáveis pela interface gráfica do ambiente.

A ferramenta IDECoRe foi desenvolvida utilizando a linguagem de programação Java, o que a torna reutilizável em qualquer plataforma que disponha do JRE – *Java Runtime Environment*. Atualmente, ela oferece suporte automático para o desenvolvimento de programas somente para a linguagem Java, mas é possível adaptar ou realizar a substituição do componente responsável pela compilação/execução para outra linguagem. A partir do IDECoRe foi feita a derivação das ferramentas JReflex e JEduc, como esquematizado na figura 2.4. JEduc é o IDE proposto neste trabalho e sua descrição detalhada encontra-se no Capítulo 5.

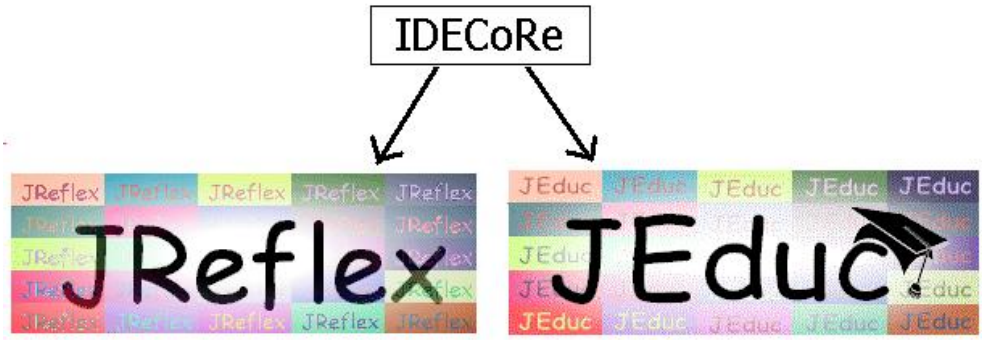


FIGURA 2.4 – Derivações de IDECoRe

2.4 Reflexão computacional

O uso da reflexão computacional tem sido enfatizado para a construção de sistemas adaptáveis, pois ela destaca, como características que tornam seu uso atrativo, a separação de aspectos e a transparência. Com isso, ao contrário dos mecanismos que são comumente usados na construção de programas em linguagens orientadas a objetos, a reflexão computacional pode ser vista como uma técnica que tem a sua maior aplicabilidade na construção de bibliotecas de componentes abertos, que objetivam facilitar o desenvolvimento de aplicações. No ambiente JEduc a reflexão computacional foi utilizada tanto na construção do ambiente como no desenvolvimento dos componentes por ele utilizados.

A reflexão computacional permite obter informações sobre a estrutura de classes e estados de objetos durante o processo de execução de programas [BER 2002a].

A partir do trabalho de Maes [MAE 87], que apontou a reflexão como uma característica intrínseca de linguagens orientadas a objetos, a reflexão computacional passou a ser considerada um importante mecanismo de extensão estática e dinâmica de linguagens orientadas a objetos e mesmo para integração de diferentes paradigmas de programação.

Linguagens de programação, por concepção, possuem uma semântica bem definida, a ser obedecida pelos seus usuários no desenvolvimento de aplicações. A reflexão computacional muda esta idéia, ao permitir alterações na implementação da linguagem [LIS 97]. Segundo Foote [FOO 98], uma das facilidades mais atraentes do modelo de orientação a objetos é a possibilidade de extensão; facilidades que não estão disponíveis em um determinado ambiente podem ser construídas pelo programador. A extensão é habitualmente implementada por bibliotecas de classes que exploram os conceitos de herança, polimorfismo e classes genéricas. Classes abstratas (interfaces) ou concretas são modeladas como descritores de dados e operações, que constituem as propriedades da classe. Propriedades que podem ser transmitidas, por herança, de forma integral ou seletiva a subclasses; propriedades estas que podem ser diretamente usadas, redefinidas ou ignoradas pelas classes descendentes [LIS 97].

A reflexão computacional permite a utilização dos mesmos conceitos e oferece novas facilidades de extensão de linguagens de programação, similares à implementação de mecanismos genéricos por alteração do compilador ou interpretador. Como vantagem adicional, novas facilidades podem ser incorporadas à linguagem de forma não permanente, para resolver problemas de um domínio específico, que não justificariam a sua ampla disponibilidade a todos os usuários da linguagem de programação [LIS 97].

Por exemplo, uma classe implementa seu comportamento através de métodos que atuam sobre seus campos, incluindo o comportamento de entrada e saída de dados para suas instâncias. Uma possível extensão seria a de oferecer o comportamento de entrada e saída sem que cada classe o implemente de forma particular. Para oferecer esse comportamento de forma transparente torna-se necessária a obtenção de informações sobre os campos e seus respectivos tipos. Portanto, implementar essa extensão de forma separada, sem alteração do compilador ou mesmo da linguagem, exige o uso de reflexão computacional.

Na linguagem Java, a reflexão computacional é um mecanismo intrínseco e que vem sendo constantemente ampliado e redesenhado para fins de melhorar o desempenho.

O processo de inspecionar um componente de um programa para a obtenção de meta-informações é chamado de introspecção e a possibilidade de alterar dinamicamente o comportamento do programa é conhecido como intercessão. Em Java estes mecanismos são implementados pela API de reflexão (`java.lang.reflect`), que oferece várias interfaces, classes e métodos os quais possibilitam o acesso aos mais diversos tipos de meta-informações (figura 2.5).

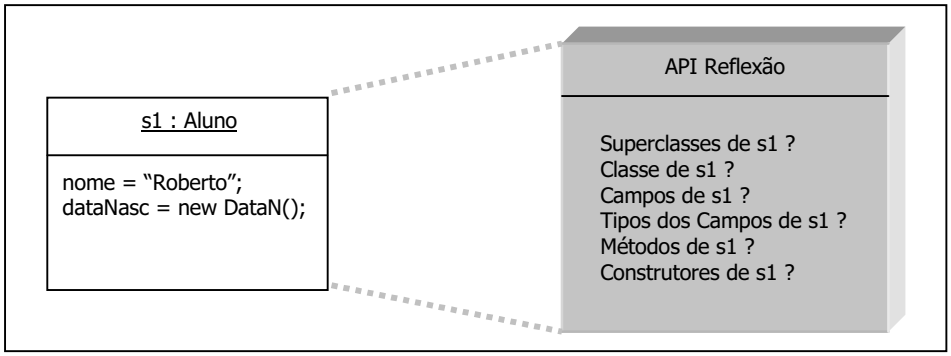


FIGURA 2.5 - Meta-informações

A biblioteca reflexiva de Java permite ao desenvolvedor:

- construir novas instâncias de classes;
- acessar e modificar atributos de objetos e classes;
- invocar métodos em objetos e classes;
- interceptar chamadas a métodos.

A reflexão tem sido extensivamente usada no desenvolvimento de ferramentas, a exemplo de: Java BeanShell [BEA 99], Jacl [TCL 99], Skij [SKI 99] e JPhyton [DOR 99], JavaBeans [HOW 98], ROB - *RemoteObject Browser* [BEC 97] e OCB - Object/Class Browser [KIR 97]. Além dessas, ela é muito adotada como solução de implementação para serialização, avaliação de expressões, construção de interpretadores de linguagens, fábricas de classes, descritores de objetos, arquiteturas *plug-in*, inspetores de classes, entre outros [SAG 98].

No contexto deste trabalho, a reflexão computacional é explorada principalmente para:

- seleção dinâmica de métodos a serem executados com base em meta-informações;
- alteração de valores nas propriedades de objetos, quando é feita uma solicitação de leitura dos referidos objetos;
- obtenção dos valores das propriedades de objetos, quando é solicitada a impressão do estado dos referidos objetos;
- investigação de objetos para exibir seus métodos e propriedades em janelas do ambiente JEduc.

2.5 Resumo do Capítulo

Com este capítulo procurou-se estabelecer as bases fundamentais deste estudo, centrado no ensino introdutório de programação e nas ferramentas associadas, na forma de ambientes de desenvolvimento de *software*. Buscou-se identificar processos de desenvolvimento visando a prototipação de um IDE específico para ensino, reutilizando componentes já desenvolvidos localmente. Também foi descrita a maneira em que a reflexão computacional foi aplicada neste trabalho. Foram adotadas como premissas fundamentais, a disponibilidade, a reutilização, o *software* livre e a simplicidade.

3 Ensino Introdutório com o Paradigma Orientado a Objetos e a Linguagem Java

Este capítulo discute os aspectos mais relevantes no ensino introdutório de programação considerando o paradigma orientado a objetos, e Java como a linguagem de programação onde os exemplos serão implementados. Planos de ensino que usam Pascal e Java são mostrados, visando possibilitar uma comparação entre o paradigma procedimental e o orientado a objetos. São consideradas também as soluções propostas para a aplicação deste novo paradigma e a forma com que elas afetam professores e alunos no processo ensino-aprendizagem.

3.1 Aspectos relevantes

Para ensinar programação é importante escolher uma linguagem que satisfaça aspectos teóricos, didáticos e tecnológicos. Quando uma linguagem de programação é avaliada para uso no ensino, principalmente com alunos iniciantes, deve ser observado se ela oferece expressividade suficiente para implementar os conceitos apresentados (aspectos teóricos), uma adequada simplicidade sintática e semântica que a torne fácil de dominar (aspectos didáticos) e suficiente disponibilidade de ferramentas de ensino (aspectos tecnológicos). Por outro lado, é importante direcionar os estudos, durante a graduação, visando satisfazer não apenas as tendências tecnológicas presentes, mas também objetivando propiciar uma sólida base conceitual que permita ao estudante enfrentar as mudanças futuras. No entanto, pode ser uma tarefa dura e desafiadora a migração de um conteúdo conhecido para conceitos e ferramentas pedagógicas mais novas, mas isso é necessário para acompanhar os avanços naturais da tecnologia [PER 2002a].

Disciplinas de introdução à programação ministradas a alunos de graduação, conforme previamente abordado na Seção 2.1, possuem duas missões importantes: (a) fazer com que os alunos aprendam a descrever soluções para problemas de acordo com a estrutura clássica entrada-processamento-saída e (b) implementar estas soluções em uma linguagem de programação.

A primeira missão tradicionalmente é cumprida através do ensino conceitual de operações básicas de entrada, processamento e saída e sua composição na forma de algoritmos e estruturas de controle: seqüenciais, condicionais e iterativas, como observado pelos planos de ensino mostrados nas Seções 3.3.1 e 3.3.2. Concorrentemente a estas noções são feitas as respectivas implementações na linguagem de programação selecionada, com o objetivo de testar, em um computador, a validade da solução. Isto implica em o aluno adquirir, ao mesmo tempo, habilidades de solucionar problemas e de codificar estes problemas em uma linguagem de programação, dominando passo a passo as complexidades da programação.

É importante salientar que, ao expressar a solução em uma linguagem de programação, não basta saber como codificar constantes, variáveis, expressões e comandos: há que estruturá-los na forma de uma unidade de execução, ou seja, um programa.

Neste momento, o do projeto do programa, surge a grande diferença entre os modelos de programação orientada a procedimentos e orientada a objetos. Enquanto o

primeiro modelo parte de uma estrutura básica de um programa principal contendo todas as unidades de execução (tipos, variáveis simples e estruturadas, funções e procedimentos e corpo principal), no modelo de objetos existe uma maior fragmentação das unidades executáveis, estruturadas na forma de classes, objetos e seus relacionamentos (tais como: uso, herança e agregação). Surge então a questão: Será o modelo de objetos adequado para disciplinas introdutórias? Em particular, será a popularíssima Java uma linguagem adequada para aprender conceitos de programação?

Uma outra questão diz respeito à simplicidade da linguagem de programação, com relação à facilidade de dominar a sintaxe da linguagem e os recursos disponíveis para a expressão direta de soluções. A linguagem Pascal, uma das linguagens mais populares para ensino de conceitos de programação, foi projetada com este atributo; o mesmo não se pode dizer em relação a Java.

Deve ser considerado, para que um erro maior não seja cometido, que um aluno iniciante não tem "maturidade" para desenvolver programas em nenhuma linguagem [LEA 96]. A maturidade, além de depender do tempo, requer a exposição do aluno a um repertório de problemas básicos, que podem ser usados para a transposição de soluções para problemas similares, e reutilizados como parte da solução de problemas mais complexos.

Na linguagem Java, um pequeno conjunto de comandos representa um volume de técnicas avançadas na metodologia de programação, técnicas estas que vem sendo estudadas no decorrer dos últimos quinze anos aproximadamente [LEA 96]. Os ambientes para o ensino de orientação a objetos possuem carências, em levantamento realizado [PAU 99], foi detectado que a área de implementação profissional é a que recebe mais ênfase no paradigma orientado a objetos.

Em recentes levantamentos sobre a linguagem de programação adotada como inicial nos cursos de graduação em computação, percebe-se ainda forte predominância de Pascal e a baixa penetração de Java nas universidades brasileiras [PER 2001b], bem como uma tendência de queda de Pascal e de crescimento de Java nas universidades americanas [REI 2001] [REI 2002]. Ao ser observada esta tendência, conclui-se que muitos professores que atualmente usam Pascal terão que migrar para Java.

Na tabela 3.1 é possível observar quais linguagens se destacam no ensino introdutório de programação, e Java, considerada complexa por muitos, está aparecendo na estatística.

TABELA 3.1 - Linguagem inicial nos cursos de graduação em computação

Linguagem	USA 2001 (*)	USA 2002 (**)	Brasil (***)
Pascal	99 (28%)	86 (23%)	40 (69%)
C	44 (12%)	43 (12%)	13 (22%)
C^{++}	89 (25%)	94 (25%)	1 (2%)
Java	11 (3%)	23 (6%)	3 (5%)
Outras	115 (32%)	126 (34%)	1 (2%)
Total	358	372	58

Levantamentos realizados por: (*) [REI 2001], (**) [REI 2002] e (***) [PER 2001].

O levantamento realizado nas universidades brasileiras foi conduzido por uma pesquisa via lista de assinantes da Sociedade Brasileira de Computação (SBC) [PER 2001b] e junto a coordenadores de cursos de graduação em Ciência da Computação, Engenharia da Computação e afins. As informações solicitadas foram as seguintes: nome da instituição, curso, linguagem introdutória de programação e o tempo de utilização desta linguagem. O objetivo principal, conforme tabulado no Anexo 1, visou qualificar e quantificar o universo a ser atingido, no caso, quais e quantos cursos de graduação em Informática e afins utilizam Pascal como linguagem introdutória à programação, quais e quantos já adotam Java como linguagem introdutória ou a adotam como uma linguagem ensinada no decorrer do curso.

3.2 Java como linguagem introdutória

Como uma motivação para a adoção de Java, Castilho [CAS 2001], em um de seus artigos, argumenta que apesar da proliferação de cursos de formação e capacitação de profissionais especializados em tecnologia da informação, há grande demanda por profissionais que programem em Java. Empresas como a Sun[1] e a IBM[2] estão buscando parcerias com algumas universidades, justamente para formar futuros profissionais familiarizados com a tecnologia Java. Elas fornecem *software* para as instituições, a fim de que os alunos possam trabalhar com estas ferramentas. Quando eles concluírem os cursos, com certeza estarão com uma boa base para ingressar no mercado de trabalho.

A tecnologia Java se tornou muito popular nos últimos cinco anos, as empresas que não contavam com estes profissionais não puderam tocar os projetos adiante. Agora, a tendência é que o mercado continue em expansão, já que no Japão, Europa e Estados Unidos as normas da televisão digital determinam que a linguagem padrão para o desenvolvimento das aplicações seja Java. Pesquisas do Gartner Group apontam que cerca de 60% das empresas no mundo estarão utilizando a tecnologia Java até 2004 [CAS 2001] [BRS 99].

Os cursos de graduação em computação devem preparar profissionais na teoria e na prática. Segundo Braun [BRA 2002], "...as universidades brasileiras vivem um dilema a respeito da formação em *software*: preparar o profissional, abrindo portas para as linguagens de programação orientadas a objeto, como Java e C^{++}, ou garantir a abrangência de conhecimento do aluno com base em linguagens tradicionais de programação, como Pascal e C". Este ponto de vista de Braun reflete a dificuldade de efetivar uma mudança na educação, preferindo muitas vezes a permanência de linguagens e conceitos vigentes há cerca de 30 anos, com base em argumentos frágeis, a exemplo de "abrangência de conhecimentos". Uma possível razão para este dilema seria o desconhecimento do modelo de objetos: Java, como representante deste modelo oferece as mesmas abstrações e estruturas de controle das tradicionais Pascal e C e mais a abstração de classes e objetos. Ou seja, permite oferecer um "conhecimento abrangente" e amplia este horizonte, ao acrescentar as abstrações de classes e objetos.

Para formar os alunos de Ciência da Computação, a UFPE (Universidade Estadual de Pernambuco) utiliza o paradigma orientado a objetos e para sua aplicação escolheu a linguagem Java. Segundo a coordenadora Edna Barros [BAE 2002], a escolha por apresentar inicialmente o paradigma orientado a objetos foi confirmada por

[1] Sun Microsystems
[2] International Business Machines

sua grande potencialidade de abstração e estruturação. O contato inicial do aluno com a linguagem de programação deve ser complementado por outras disciplinas no decorrer do curso, em especial pela disciplina que trabalha com a comparação entre os diversos paradigmas, como: funcional, lógico e orientado a objetos. O objetivo desta estruturação é que o aluno aprenda e compare os principais conceitos de linguagens de programação, de forma que seja capaz de aprender novos paradigmas com maior facilidade.

Também é ilustrativo o relato da mudança de paradigma e da linguagem Pascal para Java na universidade da Georgia - USA [HON 2001]: " Sentimos a necessidade de uma linguagem que incorporasse práticas modernas de programação. Algumas das tecnologias que estavam faltando em Pascal incluem referências seguras (em oposição a ponteiros), coleta de lixo e tratamento de exceções. Ainda mais importante, enquanto Pascal é uma linguagem orientada a procedimentos, sentimos que objetos constituem um paradigma melhor de projeto".

Outra experiência vem da universidade SUNY (State University of New York) em Oswego [OSW 2001], que foi uma das primeiras universidades a usar Java em disciplinas de introdução à programação, durante o outono de 1995. As experiências dos professores de Oswego – comentadas na página [LEA 96] – foram positivas, apesar do estágio imaturo das ferramentas Java e da falta de livros texto disponíveis na época. Foi constatado um desgaste menor, com o uso de Java como linguagem introdutória, do que o que ocorreu quando esta experiência foi realizada com C^{++}. Para o professor Doug Lea da SUNY, Java apresenta aspectos importantes: “Conceitos como programação distribuída, implementação baseada no uso de componentes, e suporte teórico na concorrência, distribuição, redes, entre outros recursos, que podem ser reservados para cursos avançados. Inicialmente devem ser introduzidas apenas as características básicas”.

Existem muitos outros exemplos onde o uso inicial do paradigma orientado a objetos é considerado um sucesso. A University of Technology - Sidney, por exemplo, aborda este paradigma há anos com alunos iniciantes em programação, usando a linguagem Eiffel [CLA 98].

Diversos autores [DEI 2001] [LEA 96] [LEW 2000] [KOF 99] [REG 2000] defendem a idéia que Java é adequada para este propósito e propõem cursos com diferentes abordagens. Deitel [DEI 2001] usa a técnica da programação estruturada tradicionalmente empregada no ensino de Pascal (ou outras linguagens orientadas a procedimentos) e gradualmente vai inserindo os conceitos específicos de programação orientada a objetos. Com isso, ele consegue aproveitar as técnicas de domínio gradual de complexidade, estabelecidas ao longo de várias décadas de ensino de programação, aliadas às novas tecnologias de programação.

Koffman e Wolz [KOF 99], em um de seus artigos, também discutem a existência de três abordagens especificamente relacionadas ao ensino de Java em disciplinas introdutórias de programação. Os dois extremos são (a) abranger toda a linguagem (incluindo objetos, applets e AWT – *Abstract Windowing Toolkit*) e utilizar GUIs desenvolvidas por alunos para realizar entradas e saídas desde o início e (b) ignorar a linguagem e evitar o desenvolvimento orientado a objetos, usar entradas e saídas em modo texto, promovendo um processo único de conhecimento. A abordagem proposta por [KOF 99] foi uma intermediária (c) que introduz as características de Java suavemente, usando pacotes fornecidos pelos professores para simplificar a interatividade do tipo GUI, e que introduz as particularidades da linguagem, como

applets, AWT, Swing e exceções no decorrer do curso, quando os conceitos básicos já foram assimilados, o que torna a tarefa de programar mais interessante.

Com base nestes trabalhos e relatos, consolida-se a idéia de que a linguagem Java, além de apresentar características desejáveis para um ensino conceitual bastante atual, oferece apoio ideal para o desenvolvimento de diversos tipos de aplicações. Acrescente-se ainda a sua inegável aceitação no meio acadêmico e no mercado, com a livre disponibilidade de ferramentas de desenvolvimento multiplataforma [KOL 2001]. Disso decorre que Java pode ser usada ao longo de todo um curso de graduação, em diferentes disciplinas, bem como pode ser estendida e especializada através do desenvolvimento de bibliotecas de classes particulares que atendam necessidades específicas das disciplinas. Segundo Laurence [VAN 99], atualmente não há nenhuma desculpa: Java deve ser usada para dar aos alunos o primeiro contato com a programação; caso contrário os alunos estariam perdendo o que há de melhor em termos de linguagem de programação.

Apesar dos muitos aspectos positivos que recomendam a adoção de Java como linguagem introdutória, também podem ser apontados problemas. Alguns estão relacionados com a linguagem em si, outros com o estado atual das ferramentas de desenvolvimento disponíveis (ambientes) e, bastante importante, a carência de professores que dominem a base conceitual própria do modelo de objetos.

A mudança de paradigma para o ensino-aprendizagem de programação orientada a objetos também não é simples [REG 2000], visto que não significa apenas a conversão de programas codificados em Pascal para Java. É necessário reestruturar a forma de implementar a solução de problemas, caso contrário o professor não estará ensinando a "real" programação orientada a objetos. Não é possível programar em Java ignorando totalmente o paradigma orientado a objetos.

Koffman e Wolz [KOF 99] dizem que o ensino de uma nova linguagem de programação nas disciplinas introdutórias requer que o professor tome importantes decisões na organização da seqüência dos conteúdos. Princípios de estrutura e funcionamento do computador, algoritmos, conceitos gerais sobre linguagens de programação de alto-nível, organização da memória durante a execução de programas, tipos de dados, variáveis, estruturas de controle, entrada e saída, tipos de dados estruturados, funções, procedimentos, entre outros, são alguns tópicos presentes na maioria das disciplinas de introdução à programação.

É importante fornecer ao aluno iniciante em programação situações para que problemas sejam solucionados, através do desenvolvimento de algoritmos e boas práticas de programação. Para tanto, nenhuma linguagem de programação deverá influenciar o aluno nesta etapa. A escolha da linguagem deve estar centrada no objetivo de que esta deve ser suporte ao processo de aprendizagem, e não em apenas usar esta linguagem por estar em evidência.

3.3 Planos de ensino de disciplinas introdutórias de programação

Para comprovar os objetivos principais e a seqüência de apresentação dos tópicos das disciplinas introdutórias de programação abordados até o momento, nas próximas subseções serão mostrados planos de ensino de duas universidades que utilizam diferentes linguagens de programação: Pascal e Java.

3.3.1 Plano de ensino de Pascal

Este plano de ensino, de disciplina introdutória de programação que utiliza a linguagem Pascal para implementação, foi extraído do curso de Bacharelado em Ciência da Computação da Universidade Federal do Rio Grande do Sul, na disciplina de Algoritmos e Programação [UNI 2001]. Ele ilustra claramente os objetivos que deverão ser atingidos em uma disciplina introdutória. A seguir é mostrado o plano de ensino completo na tabela 3.2.

TABELA 3.2 - Plano de Ensino de Pascal

Disciplina: Algoritmos e Programação – INF01202

Súmula:

A disciplina abrange os seguintes tópicos: noção de algoritmo, dado, variável, instrução e programa; construções básicas: atribuição, leitura e escrita; estruturas de controle: seqüência, seleção e iteração; tipos de dados escalares: inteiros, reais, caracteres, intervalos e enumerações; tipos estruturados básicos: vetores, matrizes, registros e *strings*; subprogramas: funções, procedimentos e recursão; arquivos.

Objetivos:

O aluno que cursou esta disciplina deve ser capaz de analisar problemas e elaborar programas que os solucionem, utilizando para isto a linguagem de programação PASCAL. Deve dominar os comandos básicos, estruturar os dados em tipos simples e estruturados, utilizar conceitos de subprogramação e recursão, além de manipular arquivos.

Conteúdos Programáticos:

1. Noção de algoritmo
2. Estrutura de um programa PASCAL
3. Comando de atribuição
4. Entrada e saída de dados
5. Estruturas e comandos de seleção simples, dupla e múltipla
6. Estrutura e comandos de repetição
7. Formatação de entrada e de saída
8. Arranjos de uma dimensão (vetores) e de mais dimensões (matrizes)
9. Subprogramação: procedimentos e funções
10. Recursividade
11. Manipulação de tipos de dados registros, intervalos e conjuntos
12. Arquivos

Carga Horária: 6 horas/aulas semanais

Procedimentos Didáticos:

As 6 horas de aula por semana são divididas em 4 horas/semana em sala de aula (aulas teórico-práticas) e 2 horas/semana em laboratório (aula prática). Os procedimentos didáticos a serem adotados nestas aulas são, respectivamente:

a)aulas teórico-práticas:
- exposições teóricas dos conteúdos;
- exercícios realizados pelos alunos individualmente ou em pequenos grupos;

b) aulas práticas:
- exercícios realizados pelos alunos diretamente nos computadores, avaliados a cada aula.

3.3.2 Plano de ensino de Java

Como plano de ensino de disciplina introdutória de programação que utiliza a linguagem Java (tabela 3.3) foi usado o do Departamento de Ciência da Computação da State University of New York (SUNY) em Oswego, na disciplina de Princípios de Programação [OSW 2001]. Este plano de ensino evidencia os objetivos que deverão ser atingidos em disciplinas introdutórias de programação que aplicam conceitos de orientação a objetos. No Anexo 2 está disponível um plano de ensino de uma universidade brasileira que também usa Java como primeira linguagem.

TABELA 3.3 - Plano de Ensino de Java

Disciplina: Princípios de Programação – CSC 212

Descrição da Disciplina:

A noção de "objeto" dirige a disciplina de programação apresentada neste curso. A linguagem de programação Java é o meio através do qual idéias chave são introduzidas. Os conceitos iniciais de envio de mensagens e fluxo de controle são apresentados, como também são os conceitos complexos de abstração, encapsulamento e hierarquia. Variáveis e tipagem, procedimentos e parâmetros são abordados. Funcionalidades proporcionadas em pacotes Java são empregadas. Algoritmos básicos são apresentados. Estratégias de solução de problemas são articuladas e exploradas. Pensamentos críticos são caracterizados durante todo o curso no contexto de solução de problemas e programação.

Justificativa:

Este é o primeiro curso com um elemento de programação no núcleo dos requisitos. As técnicas e conceitos ensinados aqui são fundamentais no resto do núcleo. Este curso irá satisfazer um dos requisitos computacionais/matemáticos da instrução geral. Este curso é exigido para graduação em ciência da computação.

Objetivos:

Após completar satisfatoriamente este curso, os alunos serão capazes de:

a) demonstrar conhecimento dos conceitos e metodologias fundamentais da programação orientada a objetos;
b) demonstrar conhecimento prático das estruturas de construções de seleção e repetição;

c) demonstrar conhecimento prático de tipos primitivos e objetos;
d) ler, escrever, compilar e executar programas em Java;
e) usar estratégias gerais de solução de problemas através de algoritmos;
f) entender e implementar vários algoritmos básicos, como algoritmos de intercalação, pesquisa e classificação;
g) desenvolver documentos para *web sites*.

Carga Horária: 3 horas/aulas semanais por semestre

Conteúdos:

A. Computadores e Programação
1. Componentes de um computador
2. Solução de problemas usando classes
3. Sintaxe e Semântica
4. Compiladores e Interpretadores
5. Criação e execução

B. Objetos e Dados Primitivos
1. Literais e variáveis
2. *Strings*
3. Introdução a objetos
4. Comandos de atribuições
5. Expressões aritméticas

C. Estruturas de Seleção
1. Estruturas de Controle
2. Expressões booleanas
3. Comando **`if`**

D. Estruturas de repetição
1. Comando **`while`**
2. Comando **`do-while`**
3. Comando **`for`**
4. Recursividade

E. Classes
1. Variáveis de instância, **`static`** e **`final`**
2. Encapsulamento e visibilidade
3. Pacotes

F. Métodos
1. Comando **`return`**
2. Parâmetros
3. Dados Locais, escopo

G. Vetores **`array`** e **`vector`**
1. Operações
2. Pesquisa e Ordenação
3. Vetores de objetos
4. Operações com **`vector`**

H. Processamento de Entrada e Saída
1. **`blue.io`**

Procedimentos Didáticos:

Aulas teóricas.
Discussões.

Aulas práticas em laboratório.

Exigências:
Leitura.
Construção e implementação de programas.
Participação nas aulas teóricas e práticas (laboratório).

3.4 Os dois paradigmas: orientado a procedimentos e orientado a objetos

A diferença conceitual entre estes dois paradigmas se inicia na estrutura estática dos programas. O paradigma procedimental baseia-se em um bloco principal composto de duas partes: as declarações, incluindo as declarações de subprogramas (funções e procedimentos) e os comandos, que contém a seqüência de ações a serem executadas. No modelo de objetos, um programa é constituído por classes que contém declarações de dados e um conjunto de definição de operações similares a funções e procedimentos que atuam sobre os dados. Uma das classes do programa é definida como a principal, ou seja, onde inicia o processo de execução.

A diferença prossegue na estrutura dinâmica, ou seja, como os elementos interagem durante o processo de execução. Na programação procedimental, o código de um programa é projetado em torno de funções parametrizadas e suas invocações; na programação orientada a objetos, os objetos são mais importantes. Essa mudança de papel revela-se na forma como os objetos são usados. Em vez de *passar* objetos (dados) para as funções, que os processam e possivelmente retornam ao local de invocação os dados alterados, os objetos são usados para *chamar* funções.

Supondo que `classifica` seja um procedimento que faz a ordenação de valores armazenados em uma estrutura de dados denominada `vet1`. No modelo procedimental, deve ser feita uma chamada do procedimento usando os dados como parâmetro: `classifica(vet1)`.

No modelo de objetos, `classifica()` é um método definido em uma classe e que atua sobre os dados também definidos na classe. Estes dados são criados por instanciação de objetos da classe - no caso, a estrutura de dados `vet1`. A chamada, denominada de mensagem, indica o objeto sobre o qual deve atuar o método desejado: `vet1.classifica()`.

Segundo Horstmann [HOR 2000], a regra para descobrir procedimentos é a mesma usada para encontrar métodos na programação orientada a objetos: procurar por verbos que identifiquem as ações a serem realizadas na descrição do problema. As diferenças importantes são que (a) na programação orientada a objetos primeiro se isolam as classes do projeto, somente depois se procuram os métodos da classe; e (b) cada método está relacionado com a classe que é responsável para efetuar a operação, o que difere dos procedimentos tradicionais.

A granularidade de componentes também é diferente nos modelos considerados. No modelo procedimental, são comuns as grandes funções que possuem lógica para muitos casos de uso. No modelo de objetos, o mecanismo de herança aliado ao polimorfismo, permite uma granularidade bem maior: podem ser criadas múltiplas

classes de objetos para representar os diferentes componentes lógicos de um programa, dos mais genéricos aos mais especializados.

Para Horstmann e Cornell [HOR 2001], a divisão dos procedimentos, no paradigma procedimental, se mostra muito adequada quando é aplicada em problemas pequenos. Mas para problemas maiores, as classes e os métodos apresentam uma grande vantagem: as classes propiciam um mecanismo conveniente de agrupar os métodos (figura 3.1). Um sistema para controlar uma linha de montagem pode exigir 2000 funções para sua implementação, ou exigir 100 classes com uma média de 20 métodos por classe. O controle, realizado pelo programador, da estrutura no segundo caso é muito mais fácil. O encapsulamento contido nas classes ajuda bastante, pois oculta a representação de seus dados e de todo o código, exceto a seus próprios métodos. Isso significa que se um erro de programação destruir os dados, será mais fácil procurar pelo culpado entre os 20 métodos que têm acesso a esses dados do que entre 2000 procedimentos. Isso tudo pode soar como o conceito de modularização: muitos usuários já escreveram programas divididos em módulos e estes somente se comunicavam entre si através de chamadas de procedimentos, e não compartilhavam dados. Se esses módulos forem bem estruturados, certamente atenderão ao conceito de encapsulamento. Contudo, em muitas linguagens, o menor deslize ou negligência na programação acaba por permitir o acesso aos dados de outro módulo, ou seja, é muito fácil desrespeitar o encapsulamento.

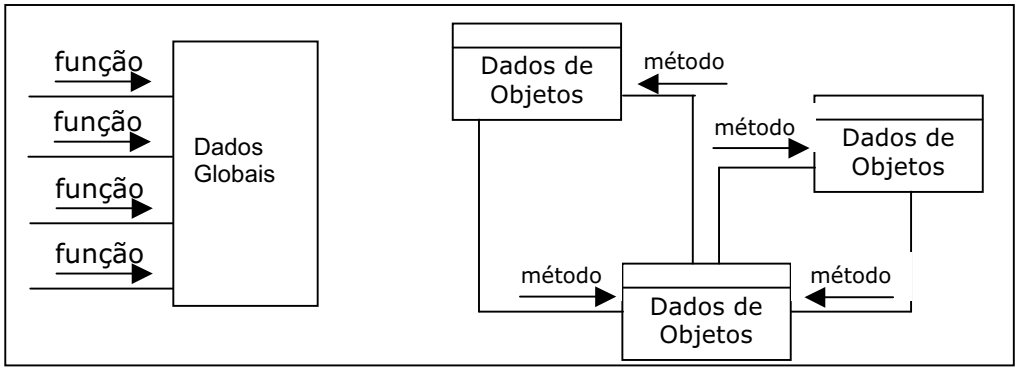


FIGURA 3.1 - Programação orientada a procedimentos e a objetos [HOR 2001]

A maioria das linguagens de alto nível, incluindo as linguagens orientadas a objeto, incorpora idéias da programação estruturada, na qual a ordem em que os comandos são executados segue uma seqüência, sem saltos para vários pontos do programa. De fato, a programação orientada a objetos agrupou a maior parte das lições aprendidas no passado, com a adoção de várias técnicas que obtiveram sucesso na abordagem procedimental e, desta forma, tornando-as fáceis de usar. Além disso, esta abordagem propôs técnicas adicionais buscando aperfeiçoar e minimizar os esforços na etapa de implementação. Por exemplo, as linguagens de programação orientadas a objeto disponibilizam mecanismos para criar objetos a partir de outros já existentes, proporcionando assim a reutilização de esforços do passado e a redução de tempo no desenvolvimento.

Acima de tudo, a abordagem orientada a objetos incentiva o desenvolvimento de programas pela construção de componentes. Estes componentes podem propiciar reutilização em mais que uma situação. A partir do uso de código reutilizável, a

necessidade de recriar funções, a cada novo sistema desenvolvido, reduz significativamente. Por sua vez, código reutilizável reduz tempo e custos no desenvolvimento de *software*, produzindo programas menos propensos a erros, uma vez que o código já foi testado em outras ocasiões, reduzindo também custos futuros com manutenções.

3.5 Objetos como elementos de programas

O modelo orientado a objetos, como o próprio nome diz, é baseado no uso de objetos como parte da implementação de sistemas. A programação orientada a objetos centraliza-se nos dados a serem manipulados e não nos procedimentos que os manipulam. Um dos maiores desafios do projeto do *software* orientado a objeto é a decomposição do sistema em classes e subclasses e a definição das propriedades de cada uma delas. Os principais componentes do paradigma orientado a objetos são: classe, objeto, mensagem e método. Eles têm um forte relacionamento e são definidos uns em termos dos outros. Todas as linguagens orientadas a objeto precisam suportar as propriedades de abstração, encapsulamento, herança e polimorfismo.

Em uma linguagem orientada a objetos, a interface para os métodos definidos numa classe pode ser especificada separadamente dos detalhes de implementação, permitindo que o projeto do sistema seja separado de sua implementação. Essa separação da definição de classe com interfaces dos métodos, e de sua implementação é fundamental na programação orientada a objetos para a obtenção da reutilização e controle dos custos de manutenção. O usuário somente necessita entender o comportamento dos objetos da classe como especificado nas interfaces dos métodos, sem se preocupar com sua implementação. A manutenibilidade é melhorada porque as alterações na implementação da estrutura de dados ou do algoritmo podem ser realizadas na região que implementa a classe. Nenhum efeito colateral é induzido no escopo fora da classe, pois a interface da classe é preservada. Esta interface forma a base de utilização da classe, especificando quais ações podem ser executadas nos objetos da classe de fora do seu escopo.

Pelo acima exposto, percebe-se que a orientação a objetos baseia-se em conceitos que tornam seu ensino complexo por exigir elevado nível de abstração para a solução dos problemas. Além disso, segundo Mendes [MEN 2001], existem outras características que contribuem com o aumento dessa complexidade:

- a necessidade de um bom nível de conhecimento e prática das técnicas de resolução de problemas;
- as deficiências das metodologias tradicionais, acostumadas a tratar com conceitos estáticos;
- sintaxes complexas que não têm representações na forma algorítmica.

Aprender a programar de forma orientada a objetos impõe problemas antes não encontrados, uma vez que o programador precisa descrever quais entidades fazem parte do universo do programa e como elas interagem entre si para atingir o resultado desejado. Dessa maneira, implementar sistemas orientados a objetos é muito mais do que aprender uma linguagem orientada a objetos. Na verdade, consiste em uma nova forma de estruturar e de pensar a solução de um problema.

A consciência dessas dificuldades fez com que educadores de todo o mundo [LEA 96] [LIS 2002] [ROB 2001] [KOF 2002] [KOF 99] [KIN 97] procurassem por estratégias e materiais que minimizassem as dificuldades encontradas por muitos alunos e professores de programação. Conforme já mencionado, a introdução e/ou migração de um conteúdo conhecido para conceitos e ferramentas pedagógicas mais novas se torna uma tarefa dura e desafiadora.

Ao longo dos anos, diversas ferramentas foram propostas com o objetivo de apoiar o esforço de aprendizagem da programação pelos alunos:

- mini-linguagens: as linguagens de programação, geralmente, costumam ser rígidas, extensas e complicadas para um aluno inexperiente. Para reduzir esses problemas uma nova abordagem tem sido proposta: criar mini-linguagens baseadas em subconjuntos de comandos de linguagens de programação convencionais. O MiniJava [ROB 2001] é um exemplo de uma mini-linguagem baseada em Java;

- mundos programáveis: é uma das técnicas utilizadas para facilitar a aprendizagem de programação, onde o objetivo principal é permitir a localização, em um ambiente simples, da ação do programa. Estes programas utilizam conceitos básicos de programação, explorando a capacidade dos alunos no desenvolvimento de programas em um ambiente simples. Um exemplo desse tipo de ferramenta é o "Karel the robot" [PAT 95];

- ambientes de desenvolvimento controlados: compreendem ambientes com um agregado de ferramentas voltadas para o desenvolvimento de sistemas profissionais. No entanto, esses ambientes são complexos para a maioria dos alunos que iniciam na aprendizagem da programação. Assim, têm sido propostos alguns ambientes simples, que incluem poucas funcionalidades, as quais oferecem suporte para algumas atividades dos alunos. O jGRASP [GRASP 2001], Ready [REA 2001], RealJ [REJ 2001] e o BlueJ [KOL 2001] [BLU 2001] são exemplos de ambientes dessa categoria;

- Ambientes para o desenvolvimento de modelos e simulações: foi iniciada em 1995 a definição e a implementação (em Smalltalk) do "Mundo dos Atores", cuja proposição inicial era a de servir como ferramenta prática para uma disciplina de introdução à programação orientada a objetos [MAA 98]. Sua potencialidade foi melhorada, passando a oferecer um ambiente riquíssimo para a implementação de modelos e simulações de outros conceitos, como: paralelismo, protocolos e sistemas distribuídos.

Como deve ser o ensino do modelo orientado a objetos para alunos iniciantes em programação? Acredita-se que a lista de conceitos do modelo de objetos deve ser ensinada, de forma simples e cautelosa, desde o início do curso [LEA 96]. Conceitos como herança, abstração, encapsulamento, polimorfismo, sobrecarga, etc., devem ser expostos aos alunos sob a forma de exemplos e exercícios, sem muita complexidade. Os detalhamentos teóricos dos aspectos do paradigma orientado a objetos devem ser ensinados, posteriormente, em outras disciplinas da grade curricular, deixando para o primeiro curso o aprendizado de programação em si.

3.6 O professor e o aluno

Este estudo de migração do paradigma procedimental para o orientado a objetos busca levantar e propor soluções para tornar o ensino-aprendizagem de programação orientada a objetos e da linguagem Java menos complexo e, com isso, obter o sucesso esperado ao final de um semestre letivo das disciplinas introdutórias de programação [PER 2001a].

Nos levantamentos citados na seção 3.1, é possível constatar que um grande número de cursos de graduação em computação adota Pascal como linguagem introdutória aliada à observação de uma tendência de crescimento de Java como linguagem acadêmica, e, com isso prevê-se que muitos professores deverão realizar esta migração de Pascal para Java. Assim, como premissa inicial, considera-se que quem faz a migração é o professor e não o aluno: este último é o objeto da aprendizagem.

Existe uma enorme carência de professores que dominem a base conceitual própria do modelo de objetos. Em paralelo, evidencia-se uma preocupação quanto à capacitação de professores para ensinar disciplinas introdutórias baseadas neste modelo, fazendo uso da linguagem Java [CLA 98]. A este respeito, Hong [HON 2001] assim se pronuncia: "Certifique-se que você possui alguém que realmente entenda Java em sua equipe. Você não vai querer ensinar projeto procedural em Java".

Como os professores estão habituados a ensinar programação com base no modelo orientado a procedimentos, alguns podem tentar a solução de simplesmente transcrever todos os programas de Pascal para a sintaxe usada em Java. Este processo de migração é muito mais complexo, pois exige transição conceitual. Na verdade é possível aproveitar alguns exemplos, mas a maioria perde todo o sentido quando são inadequadamente transcritos. Não significa apenas ensinar a mesma disciplina em outra linguagem, torna-se necessário planejar uma nova disciplina, com conteúdo similar, porém enfocando a abordagem orientada a objetos.

Com a utilização do paradigma orientado a objetos e de Java como linguagem de programação, o professor deverá fazer uso, em larga escala, de classes e objetos, implicando a necessidade do professor dominar estes conceitos e esta tecnologia [NIX 2001]. Além da mudança de paradigma, existem problemas relacionados a roteiros de aula e novos exemplos a serem propostos. A complexidade da sintaxe de Java também não deve ser ignorada.

É possível e viável obter sucesso na transição de paradigma usando Java em níveis introdutórios, mas depende de tempo e muito esforço. É importante destacar a diferença entre ensinar programação “com” Java e ensinar programação “em” Java; linguagens de programação são instáveis, surgem e desaparecem, mas fundamentos não [HON 2001]. Embora, para pessoas que já estão habituadas com o paradigma procedimental, o paradigma orientado a objetos seja complexo, os conceitos devem ser ensinados de forma simples e gradual para quem está iniciando em programação.

Com um programa simples de troca de valores, é possível exemplificar melhor um dos problemas da transcrição direta dos exercícios de Pascal para Java. Observa-se que o funcionamento de alguns algoritmos convencionais, implementados em Pascal, tem que sofrer alterações no núcleo da implementação para poderem apresentar o mesmo resultado quando implementados em Java (considerando também o paradigma orientado a objetos).

O modelo imperativo usa tipos de dados primitivos (como `integer, float, boolean`) como valores de primeira ordem, enquanto o modelo de objetos vê objetos (tipos estruturados) como valores de primeira ordem. Esta diferença atinge a estrutura dos programas e o caminho usado para resolver problemas. Os exemplos implementados nas figuras abaixo (figura 3.2, figura 3.3, figura 3.4 e figura 3.5) pretendem ilustrar como a passagem de parâmetros é totalmente diferente comparando Pascal com Java. Em todos os exemplos são usadas funções que trocam os valores dos dois parâmetros recebidos.

```
program main;
var
  a, b: integer;

  procedure troca(var i, j: integer);
  var aux: integer;
  begin
    aux := i;
    i := j;
    j := aux;
  end;

begin
  a := 10;
  b := 20;
  writeln('Valores antes: ', a, b);
  troca(a, b);
  writeln('Valores depois: ', a, b);
end.
```

```
Resultados:
  Valores antes: 10 20
  Valores depois: 20 10
```

FIGURA 3.2 - Programa troca de valores em Pascal

Este exemplo em Pascal adota a semântica de passagem de parâmetros por referência, explicitada através da sintaxe na definição dos parâmetros formais. O exemplo em Java a seguir exibe uma dificuldade associada à linguagem: a liberdade de escolher entre tipos primitivos e objetos, e a inflexibilidade da semântica de passagem de parâmetros associada. Tipos primitivos são sempre passados por valor.

```
public class TrocaErrado{
  public static void TrocaVal(int i, int j){
    int aux;
    aux = i;    i = j;    j = aux;
  }
  public static void main(String[] args){
    int a,b;    a = 10;    b = 20;
    System.out.println("Valor antes: "+a+" "+b);
    TrocaVal(a,b);
    System.out.println("Valor depois: "+a+" "+b);
}
```

```
Resultados:
  Valores antes: 10 20
  Valores depois: 10 20
```

FIGURA 3.3 - Programa troca de valores em Java (errado)

```
public class Num{
  public int valor;
  Num(int v) {
    valor = v; //Construtor
  }
}

public class Troca{
  public static void TrocaVal(Num i, Num j){
    Num aux = new Num(i.valor);
    i.valor = j.valor;
    j.valor = aux.valor;
  }
  public static void main(String[] args){
    Num a = new Num(10);
    Num b = new Num(20);
    System.out.println("Valores antes: "+a.valor+" "+b.valor);
    TrocaVal(a,b);
    System.out.println("Valores depois: "+a.valor+" "+b.valor);
}
```

```
Resultados:
  Valores antes: 10 20
  Valores depois: 20 10
```

FIGURA 3.4 - Programa troca de valores em Java (adaptado para funcionar)

```
class TrocaInt{
  int a,b;

  TrocaInt(int va, int vb){
    a = va; b = vb; //Construtor
  }

  public void TrocaVal(){
    int temp;
    temp = a; a = b; b = temp;
  }
}

public class TrocaOO{
  public static void main(String[] args){
    TrocaInt dupla = new TrocaInt(10,20);
    System.out.println("Valor antes: "+dupla.a+" "+dupla.b);
    dupla.TrocaVal();
    System.out.println("Valor depois: "+dupla.a+" "+dupla.b);
}
```

```
Resultados:
  Valores antes: 10 20
  Valores depois: 20 10
```

FIGURA 3.5 - Programa troca de valores em Java (solução orientada a objetos)

O primeiro exemplo (figura 3.2) está implementado em Pascal e se comporta conforme o esperado: os valores passados como parâmetro são trocados. Pascal aceita que a passagem de parâmetros dos tipos primitivos seja realizada por valor ou por referência [BOR 2001]. O exemplo da figura 3.3 é uma transcrição direta da implementação utilizada em Pascal para a sintaxe de Java, e não apresenta o resultado

esperado, pois em Java a passagem de parâmetros dos tipos primitivos é feita apenas por valor. No exemplo da figura 3.4 foi feita uma adaptação para poder funcionar. Foi criada uma classe `Num` que possui como propriedade apenas um valor inteiro (`valor`); como em Java a passagem de parâmetros dos objetos é feita por referência, o programa apresenta o resultado esperado. Já o exemplo da figura 3.5 define um objeto com propriedades públicas como parâmetros, e a função que troca os valores acessa essas propriedades diretamente através do apontador (*handle)* do objeto. Essa última implementação é a recomendada pelo modelo orientado a objetos, pois proporciona encapsulamento.

O aluno alvo deste levantamento é o que desconhece programação. Neste caso não haverá nenhum vício ou abordagem de programação sendo considerados. No aprendizado de programação orientada a objetos, com uso da linguagem Java, existem muitos conceitos e habilidades diferentes que os alunos devem conhecer e possuir simultaneamente. Cada uma das habilidades apresenta diferentes desafios mentais. Alunos novatos freqüentemente se sentem imobilizados quando surge um obstáculo real que depende de um sólido conhecimento de programação.

Os conceitos e habilidades chave, que os alunos precisam compreender antes de começar a escrever seus próprios programas, segundo Proulx et al. [PRO 2002], são os seguintes:

a) Criar um modelo mental do que é uma classe e uma instância de um objeto de uma classe;

b) Criar um modelo mental da organização de uma classe: dados e métodos, bem como aspectos de encapsulamento;

c) Aprendizado da sintaxe e organização do código fonte de Java: comandos, colchetes, parênteses, declaração de variáveis, declaração de métodos, etc.;

d) Entender, no contexto da linguagem Java, o que é uma classe/tipo, uma instância de um objeto/variável, um valor, e uma referência;

e) Aprender os detalhes da sintaxe: chamadas às funções membro, criação de novos objetos, controle de acesso a membros, etc;

f) Entender a lógica de um algoritmo, ou seja, os passos necessários para executar uma determinada ação;

g) Entender a transcrição de algoritmos para as definições de sintaxe da linguagem Java;

h) Entender o encapsulamento que uma classe promove para: assegurar a integridade dos dados da classe; esconder detalhes de implementação e outras tarefas complexas;

A forma pedagógica adequada para abordar esta situação seria trabalhar, se possível, um tópico por vez. Isso proporciona aos alunos inexperientes o tempo necessário para eles se familiarizarem com as novas maneiras de raciocínio e organização de idéias, sem ter que inicialmente expressá-las em alguma sintaxe de linguagem.

As primeiras aulas desse perfil de aluno deveriam começar com cenários que explicam o conceito de objetos e algoritmos usando exemplos do mundo real, sendo familiares e fáceis de compreender. Os primeiros exercícios devem introduzir a

programação orientada a objetos começando com a observação do comportamento de um objeto, e, gradualmente ir abordando a sintaxe de estruturas e programas.

Apesar do enfoque do presente estudo não estar voltado para o uso de GUIs no ensino introdutório de programação, em um estudo realizado por Proulx et al. [PRO 2002], foi proposta uma série de aulas práticas, para introduzir os conceitos de classes e objetos aos alunos iniciantes em programação, fazendo uso de interface GUI. Nessas aulas são explorados e observados o estado dos dados e o comportamento dos objetos em resposta a diferentes ações, com isso, o aprendizado de importantes conceitos vai sendo gradualmente assimilado através de ilustração.

Existem também outros dois perfis de alunos que devem ser considerados: o que conhece programação procedimental e o que além da procedimental conhece a programação orientada a objetos. Por outro lado, não pode ser ignorada a diversidade do nível de conhecimento destes alunos. Enquanto alguns tiveram um ensino secundário com introdução à informática, outros adquiriram a maioria dos seus conhecimentos sozinhos, nas horas vagas ou em cursos de extensão. Isso pode ter acarretado maus hábitos de programação. Estes alunos estão muito motivados para aprender uma linguagem de programação que, atualmente, irá torná-los mais competitivos no mercado.

Considerando o primeiro perfil, que são os alunos que conhecem programação procedimental, é necessário que o professor aponte e compare semelhanças versus diferenças, vantagens versus desvantagens entre os dois modelos. O aluno deve mudar a maneira de raciocinar e solucionar problemas através da implementação de programas. Às vezes, esta tarefa se torna mais pesada do que trabalhar com alunos que não conhecem programação.

3.7 Transição de linguagens para alunos

Tendo em vista os alunos conhecedores dos dois paradigmas, o trabalho vai girar em torno da transição de linguagens; o aluno utiliza qualquer outra linguagem orientada a objetos e vai ter que usar Java.

3.7.1 C^{++} e Java

O projeto de Java foi orientado pelo conceito fundamental de oferecer maior simplicidade e confiabilidade do que poderia ser oferecido por C^{++} [SEB 2000]. C^{++} é a linguagem orientada a objetos mais popular no ensino do modelo de objetos, mas, atualmente, tem recebido muitas críticas. Segundo King [KIN 97], Java também suporta programação orientada a objetos, com vantagens significativas sobre a linguagem C^{++}:

- o aluno tem que usar objetos. Apenas os tipos primitivos não são objetos. Não é possível definir funções ou procedimentos de forma isolada; todos devem pertencer a uma classe;
- os objetos são sempre alocados dinamicamente e manipulados através de referências, desse modo a semântica é simplificada;
- o gerenciamento de armazenamento é controlado automaticamente, o que reduz significativamente a dificuldade em se escrever muitas classes;

- as características extravagantes, como sobrecarrega de operadores e herança múltipla, não existem em Java. Os alunos têm menos a aprender antes de escrever classes reutilizáveis, e podem concentrar-se melhor no aprendizado do paradigma orientado a objetos, ao invés de dominar um número elevado de detalhes esotéricos;
- possui o coletor de lixo (*garbage collection)*, responsável por liberar objetos da memória quando estes não estão sendo referenciados.

3.7.2 Smalltalk e Java

Em seu suporte à programação orientada a objetos, Java assemelha-se mais ao Smalltalk do que ao C^{++}. King [KIN 97], em uma de suas publicações, afirma que a linguagem Smalltalk é mais orientada a objetos do que Java, e também é uma boa linguagem para o ensino introdutório. Entretanto, Smalltalk tem uma sintaxe mais complexa, tornando difícil a assimilação de seus comandos pelos novatos, entre outros inconvenientes.

Com alunos que já conhecem programação e, mais especificamente, programação orientada a objetos, é viável o uso de GUIs no desenvolvimento de suas aplicações. Pode parecer retrógrado continuar a ensinar os alunos a escreverem somente programas que executam em modo caracter, considerando que as GUIs fazem parte da implementação das ferramentas atuais. Em Java estão disponíveis dois pacotes para o desenvolvimento de aplicações GUIs, AWT e Swing.

3.8 Outros perfis de alunos

Existe também um outro tipo de aluno, o que não cursa a graduação na área de ciência da computação, mas que precisa aprender noções de programação. O Departamento de Tecnologia da Informação do Rochester Institute of Technology [ROZ 2001] oferece um curso de desenvolvimento para Web, onde a linguagem utilizada é Java, os alunos possuem experiência limitada em programação e pequena expectativa de se tornarem programadores profissionais. Java foi escolhida pelos coordenadores porque é anunciada como “fácil de aprender”, por possuir dispositivos independentes, e, por ser orientada a objetos, o que a ajustou totalmente aos aspectos do currículo do curso. A abordagem principal neste tipo de curso é o desenvolvimento exclusivamente usando *Applets*. Os alunos trabalham com PCs, Macs e estações de trabalho Unix, usando ambientes que incluem *softwares* como Netscape, Symantec Cafe e o JDK (*Java Development Kit*). Algumas conclusões relevantes foram definidas nesta experiência:

- Java pode ser efetivamente ensinada a alunos de outros cursos como primeira linguagem de programação, embora apresente melhores resultados com alunos que possuam um conhecimento introdutório prévio;
- Os alunos estão motivados a aprender esta linguagem por possuir alta demanda de profissionais no mercado de trabalho. Após este primeiro contato, eles querem aperfeiçoar o conhecimento sobre *sockets, multithreading* e interfaces de banco de dados fazendo outros cursos específicos;

- Trabalhar com um ambiente independente de plataforma é uma vantagem, apesar da maioria dos alunos preferir a plataforma PC.

3.9 Resumo do capítulo

Neste capítulo encerra-se a parte inicial do estudo, que se concentrou no levantamento de dados e de questões relativas ao uso de Java como linguagem para ensino de disciplinas introdutórias de programação e, em particular, questões referentes à migração do paradigma procedural para o paradigma de orientação a objetos. Como principais tópicos a serem destacados citam-se:

- as particularidades da linguagem Java, com relação a Pascal;
- existe uma enorme carência de professores que dominem a base conceitual própria do modelo de objetos;
- evidencia-se uma preocupação quanto à capacitação de professores para ensinar disciplinas introdutórias baseadas neste modelo, fazendo uso da linguagem Java;
- diversidade do perfil dos alunos das disciplinas introdutórias de programação.

4 Proposta de Simplificação

Neste capítulo são apontadas algumas complexidades encontradas na linguagem Java, destacando-se a entrada e saída de valores via console do sistema. É apresentado um pacote de classes para facilitar, ou até mesmo ocultar, o ensino destas complexidades a alunos novatos – `ufrgs.inf.iosimple`. A documentação implementada para auxiliar os professores também está amplamente descrita.

4.1 Introdução

Este estudo se propõe a amenizar a transição pedagógica decorrente da migração de Pascal para Java e também simplificar algumas características da linguagem Java, visando agilizar o processo de desenvolvimento dos primeiros programas pelos alunos, reduzindo a quantidade de codificação exigida para a implementação e o conhecimento minucioso das bibliotecas de classes de sua API padrão [PER 2002a]. O objetivo principal é dispor de um ambiente de desenvolvimento de programas munido de um conjunto de pacotes de classes "simplificadoras" que auxiliem o estudante a escrever programas rapidamente, sem exposição às complexidades de implementação de Java. Ao mesmo tempo, facilitar aos professores o ensino desta linguagem.

Embora possa parecer um paradoxo, uma possível solução para aumentar a simplicidade de Java consiste em aumentar o número de classes. Essas classes buscariam a simplificação das construções nativas através da definição de novas interfaces.

O objetivo da simplificação sintática pretendida é, unicamente, esconder detalhes de implementação sem descaracterizar os princípios de programação orientada a objetos e sem tentar assemelhar a sintaxe de Java à sintaxe de Pascal. Esta última, apenas serve como paradigma de clareza de conceitos e simplicidade de expressão.

A metodologia adotada na busca de simplificação pode ser assim resumida:

a) identificar pontos de complexidade para alunos de disciplinas introdutórias de programação;

b) projetar interfaces mais simples do que as originais, e implementá-las de forma a reduzir a quantidade de código a ser escrito pelo aluno;

c) implementar as interfaces em código Java, sem uso de extensões da linguagem e ou pré-processadores.

A seguir é apresentada a simplificação de um ponto identificado como sendo complexo para alunos iniciantes: a entrada de dados via console. O exemplo da figura 4.1 ilustra a complexidade do código Java (observar linhas 2, 3, 5 e 9) se comparado ao código Pascal, exemplo da figura 4.2, que resolve o mesmo problema.

Para fornecer a simplificação sintática, mencionada anteriormente, foram definidas e construídas algumas classes, denominadas "simplificadoras". Estas classes foram disponibilizadas através do pacote `ufrgs.inf.iosimple`, cujo intuito é reduzir a complexidade ilustrada nos exemplos abaixo (figura 4.1 e figura 4.2).

```
1. class Benvindo{
2.    private static BufferedReader in = new BufferedReader
                                (new InputStreamReader(System.in));
3.    public static void main( String args[]){
4.      String nome = "";
5.      try{
6.        System.out.print("Como é seu nome?: ");
7.        nome = in.readLine();
8.        System.out.print("Benvindo "+nome);
9.      } catch (IOException e){
10.         System.out.println(" Erro!");
11.      }
12.   }
13. }
```

FIGURA 4.1 - Classe `Benvindo` em Java

```
1.  program Benvindo;
2.    var nome : String;
3.  begin
4.    writeln('Como é seu nome? ');
5.    readln(nome);
6.    writeln('Benvindo ', nome);
7.  end.
```

FIGURA 4.2 - Classe `Benvindo` em Pascal

4.1.1 Entrada e saída via console: questão crucial

Se para alunos iniciantes, a leitura de dados via console pode ser destacada como um ponto de complexidade, para saída pode ser usado simplesmente `System.out.println()`. Este método é parte de um objeto pré-definido chamado `System.out`, cujo propósito é precisamente mostrar as saídas para o usuário. No entanto, para entrada pode ser usado o método `System.in.read()` do correspondente objeto `System.in`, mas este disponibiliza poucas facilidades, requerendo habilidades avançadas de programação em Java para fazer seu uso efetivo. Pode-se criar também uma instância das classes `DataInputStream` ou `InputStreamReader`. No primeiro caso, é realizada a leitura de tipos de dados primitivos com a utilização de métodos específicos para cada tipo, enquanto que no segundo caso a leitura é feita através da leitura de *bytes* que são traduzidos para caracteres no padrão especificado. Em ambos os casos, é possível que ocorra uma exceção, do tipo `IOException`, que deve ser capturada e tratada, implicando ensinar exceções aos alunos, para fazer apenas uma entrada de dados via console.

O exemplo a seguir mostra como é feita a entrada de dados usando apenas classes originais e como pode ser feita a simplificação.

Supondo:

a) uma classe denominada `Data` (figura 4.3) que define um tipo registro com três atributos inteiros e nenhum método.

```
1.  public class Data {
2.     public int dia;
3.     public int mes;
4.     public int ano;
5.  }
```

FIGURA 4.3 - Classe `Data`

b) uma classe `Aluno` (figura 4.4) que também define um tipo registro que usa a classe `Data` para declarar uma de suas propriedades.

```
1.  public class Aluno {
2.     public String nome;
3.     public Data dataNasc = new Data();
4.  }
```

FIGURA 4.4 - Classe `Aluno`

c) entrada de dados para todos as propriedades da classe `Aluno` deve ser feita através da console, utilizando-se do método `main()` de uma classe que declara e instancia um objeto do tipo `Aluno`, conforme mostra o exemplo da figura 4.5.

```
1. class TesteAlunoSC{
2.    private static BufferedReader in = new BufferedReader
3.                (new InputStreamReader(System.in));
4.    public static void main( String args[]){
5.       Aluno a1= new Aluno();
6.       try{
7.          System.out.print("Entre com uma String: ");
8.          a1.nome=in.readLine();
9.          System.out.print("Entre com um int: ");
10.         a1.dataNasc.dia=Integer.parseInt(in.readLine());
11.         System.out.print("Entre com um int: ");
12.         a1.dataNasc.mes=Integer.parseInt(in.readLine());
13.         System.out.print("Entre com um int: ");
14.         a1.dataNasc.ano=Integer.parseInt(in.readLine());
15.     } catch (IOException e){
            16.   System.out.println(" Erro!");
            17.   }
18.   }
19. }
```

FIGURA 4.5 - Classe `TesteAlunoSC` (usando classes da API padrão de Java)

Neste exemplo (figura 4.5) podem ser destacados os seguintes pontos de complexidade:

- declaração do *buffer* de entrada (linhas 2 e 3), através do uso das classes `BufferedReader` e `InputStreamReader`;
- sistema de tratamento de exceções (linhas 6 e 15), com um bloco `try..catch`;
- acesso às propriedades de `Data (dataNasc)` (linhas 10, 12 e 14);
- conversão de tipos (linhas 10, 12 e 14)). Como as propriedades de `dataNasc` são do tipo `int` e a entrada é realizada através da leitura de uma linha (`String`), é necessário converter a leitura `String` para `int`.

Com a utilização de uma classe denominada `ConsoleP`, que implementa métodos simplificados de entrada e saída, o programa pode ser assim representado (figura 4.6):

```
1. import ufrgs.inf.iosimple.ConsoleP;
2. class TesteAlunoCC {
3.    public static void main( String args[]){
4.       Aluno a1= new Aluno();
5.       a1.nome=ConsoleP.readString();
6.       a1.dataNasc.dia=ConsoleP.readInt();
7.       a1.dataNasc.mes=ConsoleP.readInt();
8.       a1.dataNasc.ano=ConsoleP.readInt();
9.    }
10. }
```

FIGURA 4.6 - Classe `TesteAlunoCC` (usando a classe "simplificadora" `ConsoleP`)

Uma outra simplificação pode ser feita utilizando a classe `ConsoleO`, que oferece serviços de entrada e saída de objetos complexos, os definidos pelo usuário. O mesmo exemplo seria assim apresentado (figura 4.7):

```
1. import ufrgs.inf.iosimple.ConsoleO;
2. class TesteAlunoCO {
3.    public static void main( String args[]){
4.       Aluno a1= new Aluno();
5.       ConsoleO.readAll(a1);
6.    }
7. }
```

FIGURA 4.7 - Classe `TesteAlunoCO`(usando a classe "simplificadora" `ConsoleO`)

4.1.2 Considerações sobre o uso de pacotes personalizados para entrada e saída: prós e contras

Recentemente, em um artigo de Koffman [KOF 2002], foi amplamente discutida a questão do uso de pacotes personalizados para facilitar a entrada e saída de dados em programas Java desenvolvidos por alunos novatos em programação. E também a questão do uso de interfaces GUI nestes pacotes.

Existem diversos livros de introdução à programação com Java que utilizam pacotes para simplificar a entrada e saída. Algumas classes foram desenvolvidas para facilitar a entrada e saída via console, dentre as várias existentes são destacadas:

- `ConsoleReader` [HOR 2000];
- `Text` [BIS 97];
- `SimpleInput` [BAR 2000];
- `SavatchIn` [SAV 99];
- `Keyboard` [LEW 2000].

Essas classes foram desenvolvidas porque os autores dos livros e vários professores, de disciplinas de introdução à programação, reconheceram que principalmente a entrada de dados via console de Java era muito complexa. Outros autores, no entanto, criaram pacotes de classes que também permitiam a realização de entrada e saída com caixas de diálogo – interfaces GUI. Esses autores apontaram a entrada e saída via console como sendo muito tediosa e desmotivadora, visto que os alunos de hoje estão acostumados com aplicações que utilizam este tipo de interface [KOF 2002]. Para implementar estes pacotes, os autores utilizaram os pacotes AWT ou Swing. Eles argumentam que o uso dos pacotes GUI é uma abordagem simples para realizar entrada e saída, na qual uma janela do tipo console é usada para as saídas dos programas e uma janela de diálogo é usada para realizar as entradas.

Neste artigo de Koffman também são apontadas algumas argumentações de professores com relação aos prós e contras no uso destes pacotes "simplificadores" para ensinar programação "com" Java.

> "H. Conrad Cunningham, da University of Mississippi, fez um comentário resumindo os dois pontos de vista:
>
> Por um lado, o uso simplificado de I/O ou pacotes GUI podem tornar Java fácil de se ensinar, permitindo seu uso em disciplinas introdutórias de programação. Então, os vários pacotes disponíveis com os livros texto podem ser úteis. E nós queremos transmitir aos nossos alunos que é fácil adicionar capacidades significativas em Java, tal como acontece em outras linguagens.
>
> Por outro lado, há uma preocupação em que o uso desses pacotes "simplificadores" confundam os alunos de primeiro ano. Os alunos tendem a considerar este tipo de aprendizado como sendo parte de Java. Quando eles tiverem que mudar para um outro ambiente, eles ficarão relativamente perdidos e confusos. Terão então, como sobrecarga, que aprender como fazer uso de um dos pacotes de I/O padrão da API de Java, ou fazer a instalação de um pacote que eles estejam acostumados a usar. Deste ponto de vista há algum benefício no uso das classes padrão da linguagem Java.
>
> David Arnow, do Brooklyn College, em New York, tem o seguinte argumento contra o uso de outros pacotes: existe um outro argumento a favor da NÃO utilização de pacotes especiais para I/O com alunos iniciantes em programação com Java. Enquanto construções do tipo:

```
BufferedReader br = new BufferedReader(
                      new InputStreamReader(
                        new FileInputStream(
                          new File("myinput"))));
```

introduzem conceitos complexos, eles também têm a oportunidade de: (a) enfatizar a idéia de classes como modelo, e como conjuntos particulares de comportamento; (b) ilustrar o uso de composição, e (c) sustentar a noção e a utilidade da abstração." [KOF 2002, p.19]

4.2 Classes "simplificadoras" do pacote `ufrgs.inf.iosimple`

Apesar da existência de diversos pacotes que objetivam facilitar a entrada e saída de tipos em programas implementados em Java, neste trabalho houve a necessidade da implementação de um novo pacote – `ufrgs.inf.iosimple`. Isso ocorreu devido à constatação da necessidade de alguns controles adicionais. Além disso, em nenhum dos pacotes foram encontradas classes que possibilitavam a entrada e saída de objetos inteiros declarados pelos usuários. A definição, o projeto e a construção dessas classes ocorreram em etapas, visando sempre obter o maior grau de reutilização possível. Essas classes e seus relacionamentos estão ilustrados na figura 4.8.

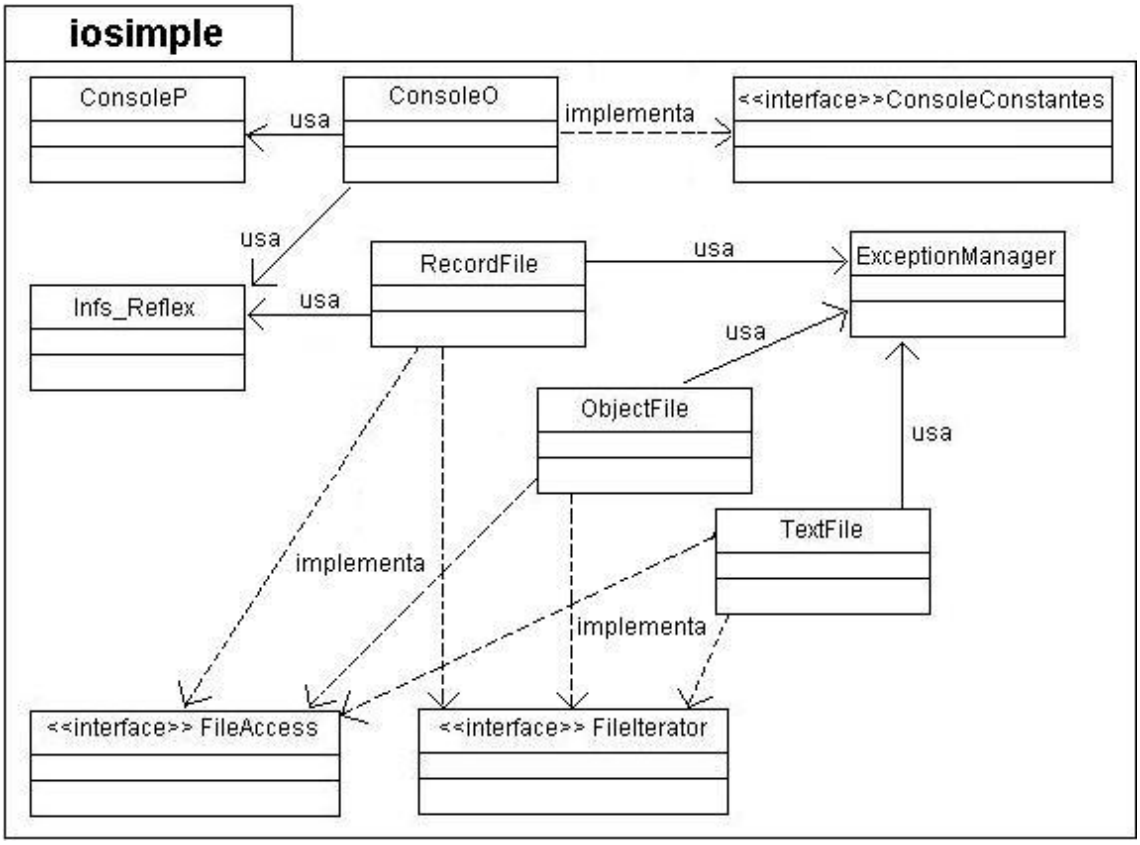


FIGURA 4.8 - Pacote `ufrgs.inf.iosimple`

Em um primeiro momento, foi definida e construída a classe `ConsoleP`, para simplificar a entrada e saída de tipos primitivos, *strings* e das classes invólucros

(*wrappers*). A segunda etapa consistiu no projeto e construção da classe `ConsoleO`, que visa simplificar a entrada e saída de objetos definidos pelo usuário. Os detalhes da implementação e a interface dessas duas classes estão amplamente descritos nas seções 4.2.2 e 4.2.3.

Na terceira etapa, foram definidas as classes "simplificadoras" para a leitura, escrita e pesquisa em arquivos: `TextFile, ObjectFile` e `RecorFile`. Estas classes foram desenvolvidas por Brugnara [BRU 2002a] em seu Projeto de Diplomação no curso de Bacharelado em Ciência da Computação do Instituto de Informática da UFRGS (Universidade Federal do Rio Grande do Sul).

4.2.1 Classes simplificadoras de arquivos

Embora a leitura e gravação de arquivos texto não ofereça maiores dificuldades, o mesmo não se pode afirmar em relação a arquivos de objetos, ou seja, arquivos de registros de formato fixo, com acesso seqüencial ou direto. Além da linguagem Java não possuir o conceito de "estrutura de registro", e, visto considerar como registro os campos de uma classe e seus valores como o estado de cada objeto instanciado a partir desta classe, existe a obrigatoriedade de cada classe do usuário implementar a interface `Serializable` do pacote `java.io`, para que possa ser gravado em um "arquivo de objetos". Adicione-se a isso o fato de um arquivo de objetos admitir diferentes tipos de objetos, que podem apresentar diversidade de campos, tipos e tamanho.

Durante a implementação das classes de manipulação de arquivos, foram definidas duas interfaces, que estabelecem um padrão de assinaturas de métodos a ser seguido pela implementação das três classes destinadas à entrada e saída de arquivos. Essa padronização objetiva facilitar a utilização de todas as classes de arquivo deste pacote. O aluno, após aprender a aplicação e a funcionalidade dos métodos de uma classe, terá grande facilidade em utilizar as outras.

As interfaces criadas são denominadas `FileAccess` (figura 4.9) e `FileIterator` (figura 4.10). A primeira define assinaturas de métodos para as operações básicas de entrada e saída em arquivos, como leitura e gravação. A segunda define operações para percorrer um arquivo, como retornar a posição atual do ponteiro ou questionar a existência de mais registros.

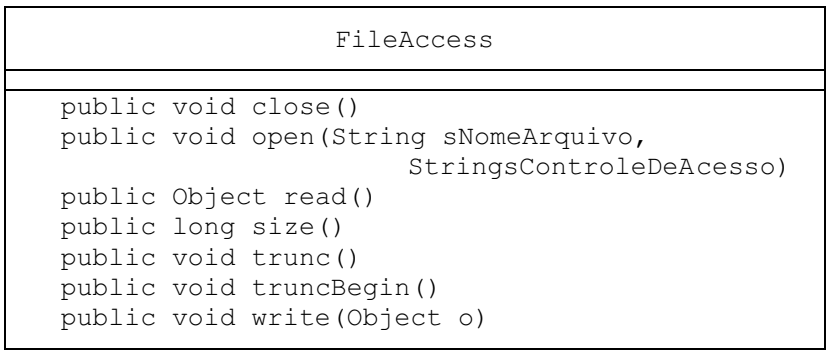


FIGURA 4.9 - Interface `FileAccess`

Observe-se que a interface `FileIterator` estende a interface `Iterator` do pacote `java.util`, que possui as assinaturas dos métodos representados na figura 4.11,

os quais devem ser implementados por qualquer classe que implemente a `FileIterator`. Com isso, pretende-se obter uma hegemonia sintática de arquivos com a biblioteca de coleções.

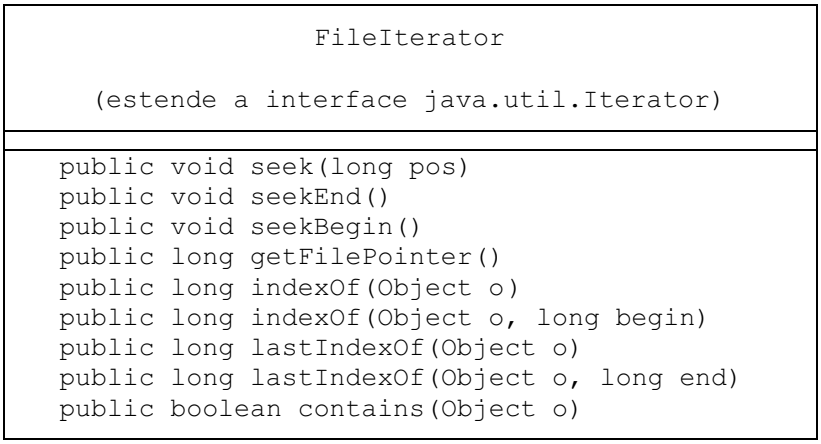


FIGURA 4.10 - Interface `FileIterator`

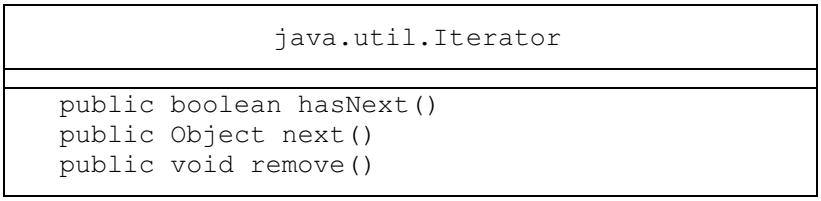


FIGURA 4.11 - Interface `java.util.Iterator`

A classe `TextFile` (figura 4.12) possibilita o acesso, o posicionamento e a manipulação de arquivos texto, ou de arquivos onde não seja necessário um tratamento especial.

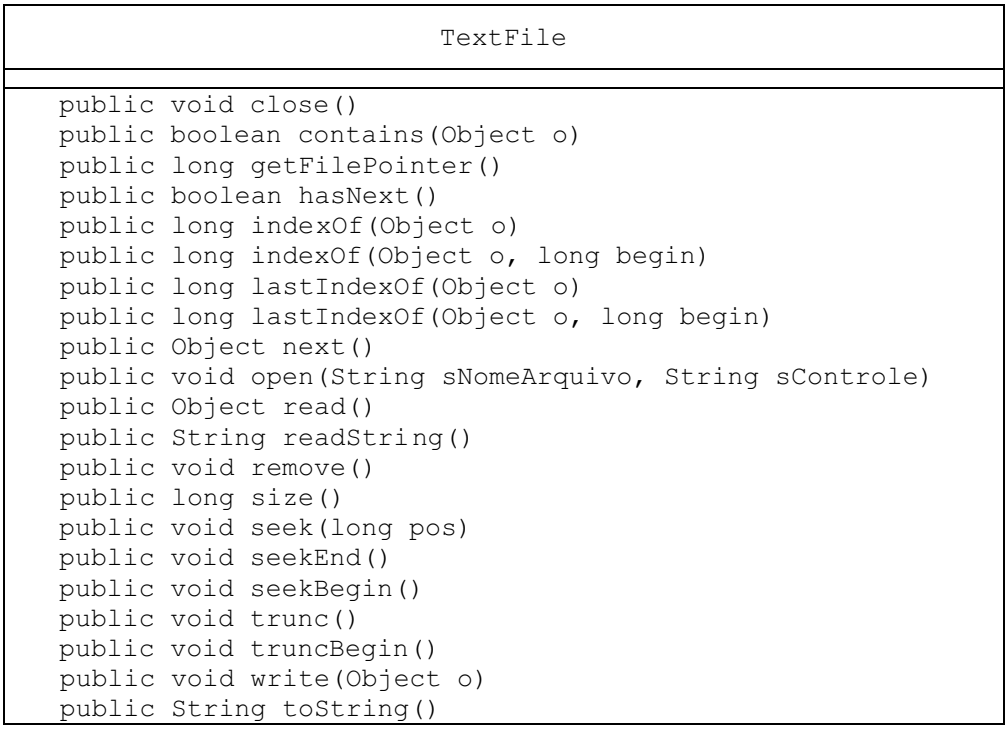


FIGURA 4.12 - Interface da classe `TextFile`

As classes `ObjectFile` (figura 4.13) e `RecordFile` (figura 4.14) permitem acesso, posicionamento e manipulação em arquivos que armazenam objetos, porém cada classe foi implementada utilizando diferentes formas de armazenamento. A classe `ObjectFile` usa a API padrão de Java para serialização enquanto a `RecordFile` usa a classe `Infs_Reflex` que utiliza o mecanismo de reflexão computacional.

ObjectFile
public void close() public boolean contains(Object o) public long getFilePointer() public boolean hasNext() public long indexOf(Object o) public long indexOf(Object o, long begin) public long lastIndexOf(Object o) public long lastIndexOf(Object o, long end) public Object next() public void open(String sNomeArquivo, String sControle) public Object read() public void remove() public long size() public void seek(long pos) public void seekEnd() public void seekBegin() public void trunc() public void truncBegin() public void write(Object o) public String toString()

FIGURA 4.13 - Interface da classe `ObjectFile`

RecordFile
public void close() public boolean contains(Object o) public static int getClassSize(Class c) public long getFilePointer() public boolean hasNext() public long indexOf(Object o) public long indexOf(Object o, long begin) public long lastIndexOf(Object o) public long lastIndexOf(Object o, long end) public Object next() public void open(String sNomeArquivo, String sControle) public Object read() public void remove() public long size() public void seek(long pos) public void seekEnd() public void seekBegin() public void trunc() public void truncBegin() public void write(Object o) public String toString()

FIGURA 4.14 - Interface da classe `RecordFile`

Além das classes acima mencionadas, foi desenvolvida uma classe para tratamento de exceções, buscando tornar mais robusto e menos complexo o sistema de tratamento de arquivos definidos no pacote `ufrgs.inf.iosimple`.

A classe `ExceptionManager` (figura 4.15) foi criada com o propósito de tratar as exceções que possivelmente venham a ocorrer no escopo das classes de manipulação de arquivos (`TextFile, ObjectFile` e `RecordFile`). Desse modo, tais exceções não são propagadas para o programa do usuário. Em algumas ocasiões, como a gravação de um arquivo no disquete, se o disquete não estiver na unidade, a classe `ExceptionManager` mostra ao usuário a exceção gerada e pergunta pela ação a ser tomada.

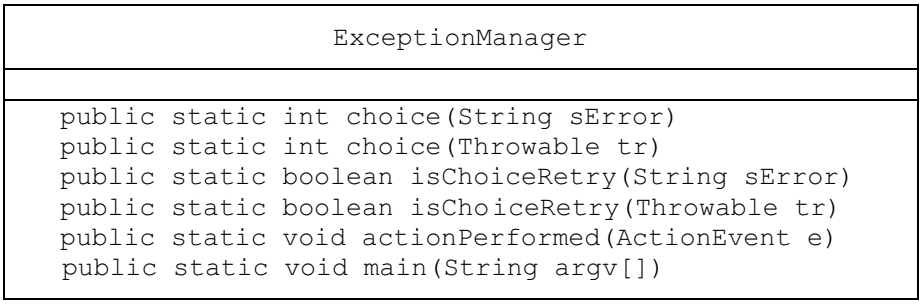


FIGURA 4.15 - Interface da classe `ExceptionManager`

Durante o desenvolvimento das classes "simplificadoras", alguns cuidados quanto ao uso da reflexão computacional foram observados:

- Este mecanismo foi usado somente onde existia necessidade, pois a chamada dinâmica de métodos usando reflexão causa sobrecarga no desempenho;
- o encapsulamento foi um fator fundamental para tornar transparente o uso de reflexão, pois ao utilizar este recurso o código torna-se mais complexo e de difícil entendimento;
- o tratamento de exceções foi analisado com cuidado porque erros na chamada dinâmica de métodos são descobertos somente no momento da execução e não mais pelo compilador.

4.2.2 Projeto e implementação da classe `ConsoleP`

A classe `ConsoleP` realiza a entrada e saída, via console, de tipos primitivos, *strings* e das classes invólucros (*wrappers*). Além disso, foi realizado o tratamento local das exceções que poderiam ocorrer para cada entrada, evitando assim a necessidade de uso do bloco `try..catch` no código do aluno. A definição dessa classe sofreu influência direta da classe `Keyboard` desenvolvida por Lewis e Loftus [LEW 2000]. A classe `Keyboard` possibilita apenas a entrada de tipos primitivos e *strings,* não disponibilizando métodos para entrada dos tipos invólucro (*wrapper)* e para a saída de valores. A interface da classe `ConsoleP` é mostrada na figura 4.16.

É possível notar que alguns métodos possuem nomes praticamente duplicados, e que fogem ao padrão recomendado para nomear métodos na linguagem Java. Este padrão diferenciado foi adotado para facilitar a assimilação da sintaxe pelo aluno, ou

seja, quando for realizada a entrada de um tipo primitivo, o nome do método está todo escrito com letras minúsculas (usando a descrição referente ao nome do tipo de dados em questão), e, para a entrada de tipos invólucros (*wrappers*) os nomes dos métodos seguem a recomendação padrão [SUN 2002b] e também a descrição referente à classe que está sendo utilizada.

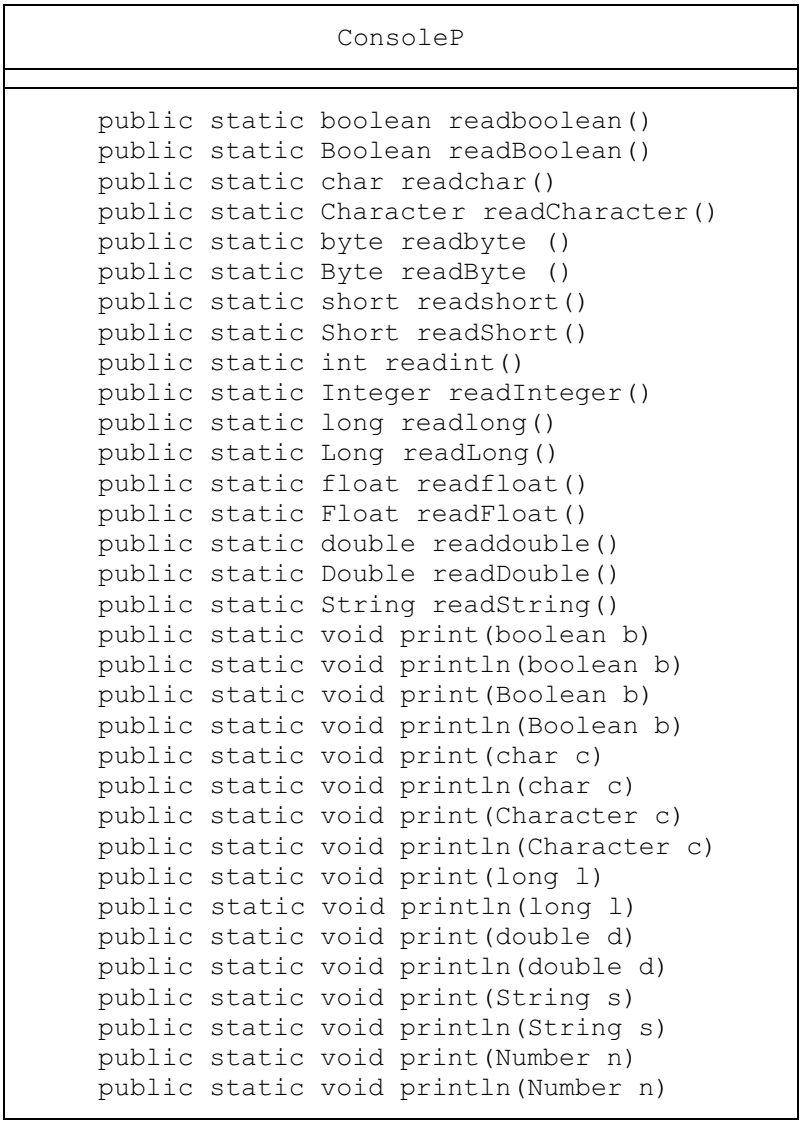


FIGURA 4.16 - Interface da classe `ConsoleP`

Todos os métodos da interface da classe `ConsoleP` possuem visibilidade estática (`static`). A implementação foi definida desta forma para que seu uso não fique vinculado a uma instância de um objeto. Estes métodos são denominados métodos de classe.

O tratamento de exceções é feito localmente dentro de cada método, tornando possível detectar, sem abortar a execução do programa, se o aluno informou um valor compatível ao que lhe foi solicitado. Em caso de erro, uma mensagem é exibida para informar o erro ocorrido e automaticamente é solicitado que o aluno informe novamente o valor. Esta é uma abordagem diferente da utilizada por Lewis e Loftus [LEW 2000] na

classe `Keyboard`, que retorna um valor nulo ou mínimo, de acordo com o tipo lido, quando o valor informado é inválido. Para ilustrar melhor esta implementação, na figura 4.17 está o tratamento realizado na `ConsoleP` e na figura 4.18 o tratamento feito pela classe `Keyboard`.

```
//Leitura de um tipo primitivo
public static int readint(){
  String token = getNextToken();
  int value;
  try {
    value = Integer.parseInt(token);
  }
  catch (Exception exception) {
    error ("Erro ao ler um valor int, tente novamente.");
    value = readint();
  }
  return value;
 }

//Leitura do tipo invólucro respectivo
public static Integer readInteger() {
  String token = getNextToken();
  Integer retInt;
  try {
    retInt = new
           Integer(Integer.toString(Integer.parseInt(token)));
  }
  catch (Exception exception) {
    error ("Erro ao ler um valor Integer, tente novamente.");
    retInt = readInteger();
  }
  return retInt;
}
```

FIGURA 4.17 - Método para entrada dos tipos `int` e `Integer` na classe `ConsoleP`

```
public static int readInt() {
  String token = getNextToken();
  int value;
  try {
    value = Integer.parseInt (token);
  }
  catch (Exception exception) {
    error ("Error reading int data, MIN_VALUE value returned.");
    value = Integer.MIN_VALUE;
  }
  return value;
}
```

FIGURA 4.18 - Método para entrada do tipo `int` na classe `Keyboard`

Como base para a criação desta classe, foram utilizadas algumas classes de entrada e saída do pacote de classes padrão `java.io: BufferedReader` e `InputStreamReader`, e do pacote `java.lang.Object: StringTokenizer`. A seguir estão algumas observações sobre as classes utilizadas [SUN 2002c]:

- `BufferedReader`: foi usada na `ConsoleP` por possuir a habilidade de ler um *string* de caracteres. Quando um objeto dessa classe é instanciado ele redefine a entrada padrão de *stream* para ajudar nas atividades de entrada e saída. O método `readLine` será chamado para ler o *string* de caracteres digitado pelo usuário através do teclado (*prompt*). É necessário utilizar *buffers* para a entrada permitindo assim a utilização das teclas de função como: *delete, backspace*, entre outras. Os caracteres, neste momento, estão armazenados no *buffer* de entrada e somente após o uso da tecla *enter* é que o controle vai para o programa do usuário. Este objeto necessita de outra classe como parâmetro, neste caso a `InputStreamReader`, para ler a entrada de *bytes*;
- `InputStreamReader`: é uma classe de manipulação de *streams* de entrada, que serve como canal para a entrada de caracteres;
- `StringTokenizer`: o propósito desta classe é separar um *string* em pedaços, chamados *tokens*, baseada em um conjunto de delimitadores (espaços, tabuladores, etc.).

Conforme mostrado na figura 4.19, na classe `ConsoleP` existem métodos privados responsáveis pela manipulação dos *buffers* e *streams* lidos nas entradas.

```
/* Captura o próximo pedaço (token) assumindo que este deverá
     estar nas linhas subsequentes de entrada */
 private static String getNextToken()

 // Captura o próximo pedaço (token) que já foi lido
 private static String getNextToken (boolean skip)

 /* Captura o próximo pedaço (token) de entrada, que deve vir da
    entrada na linha corrente ou subsequente. O parâmetro
    determina se as linhas subsequentes serão utilizadas.  */
 private static String getNextInputToken (boolean skip)

 /* Retorna true se não existem mais pedaços (token) para
     serem lidos na linha corrente de entrada.  */
 private static boolean endOfLine()
```

FIGURA 4.19 - Métodos usados para manipular *streams* e *buffers* na `ConsoleP`

Os métodos responsáveis pela saída foram implementados de uma maneira bastante simples, e inclusive possuem os mesmos nomes dos métodos `print()` e `println()` do objeto `System.out`. A definição dos nomes ocorreu desta forma para apresentar um vínculo maior com esses métodos, facilitando uma assimilação futura por parte dos alunos que não queiram mais utilizar a classe `ConsoleP`. Na `ConsoleP` eles são sobrecarregados, e, em alguns casos, de acordo com o tipo de dados, houve a necessidade de criar dois métodos para representar o respectivo tipo. Os tipos suportados por sobrecarga podem ser identificados na tabela 4.1:

TABELA 4.1 - Tipos suportados nos métodos de saída da `ConsoleP`

Assinatura do método	Tipos suportados
`public static void print(long s);` `public static void println(long s);`	`byte, short, int e long`
`public static void print(double s);` `public static void println(double s);`	`float e double`
`public static void print(Number s);` `public static void println(Number s);`	`Byte, Short, Integer, Long, Float e Double`
`public static void print(boolean s);` `public static void println(boolean s);`	`boolean`
`public static void print(char s);` `public static void println(char s);`	`char`
`public static void print(String s);` `public static void println(String s);`	`String`
`public static void print(Boolean s);` `public static void println(Boolean s);`	`Boolean`
`public static void print(Character s);` `public static void println(Character s);`	`Character`

4.2.3 Projeto e implementação da classe `ConsoleO`

A classe `ConsoleO` (figura 4.20 e figura 4.21) visa simplificar a entrada e saída de objetos definidos pelo usuário, caso este objeto herde propriedades de outros objetos, as propriedades declaradas como `private` não serão lidas. Com o uso dessa classe, o sistema lê e imprime objetos instanciados a partir de classes do usuário com a mesma semântica de entrada e saída de tipos primitivos, ou seja, sem a necessidade de definir, em cada classe, métodos específicos para a entrada e saída dos campos definidos e herdados.

Para possibilitar esse nível de simplificação foi utilizado o mecanismo de reflexão computacional, pois com ele é possível obter informações sobre todas as variáveis de instância de cada objeto, incluindo nome, conteúdo e tipo. O tipo pode ser um tipo padrão ou definido pelo usuário.

Os tipos das propriedades da classe são analisados um a um, com auxílio da reflexão computacional, de forma a percorrer toda a hierarquia de herança da classe do objeto até que sejam encontrados apenas os tipos padrão da linguagem Java: os primitivos, os invólucros (*wrappers*) e s*trings*. Após a identificação das propriedades definidas e herdadas, pode ser feita a utilização dos métodos definidos na classe `ConsoleP` para fazer a entrada e saída de cada variável de instância do objeto. Conforme abordado na seção anterior, a classe `ConsoleP` possui métodos para entrada e para a saída de tipos primitivos, invólucros e *strings*, que são os tipos resultantes da decomposição das propriedades dos objetos.

A classe `ConsoleO` implementa a interface `ConsoleConstantes`, onde estão declaradas algumas constantes de controle utilizadas em seu código. Essa interface não possui métodos.

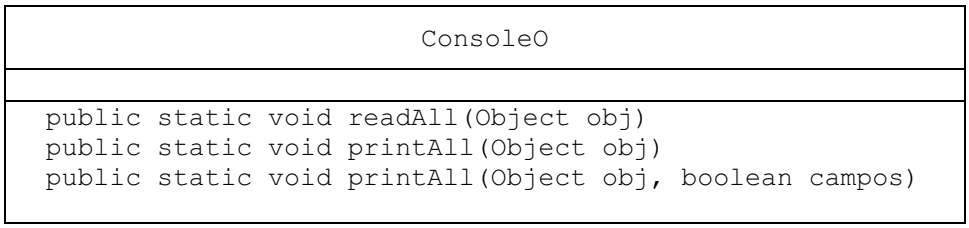


FIGURA 4.20 - Interface da classe `ConsoleO`

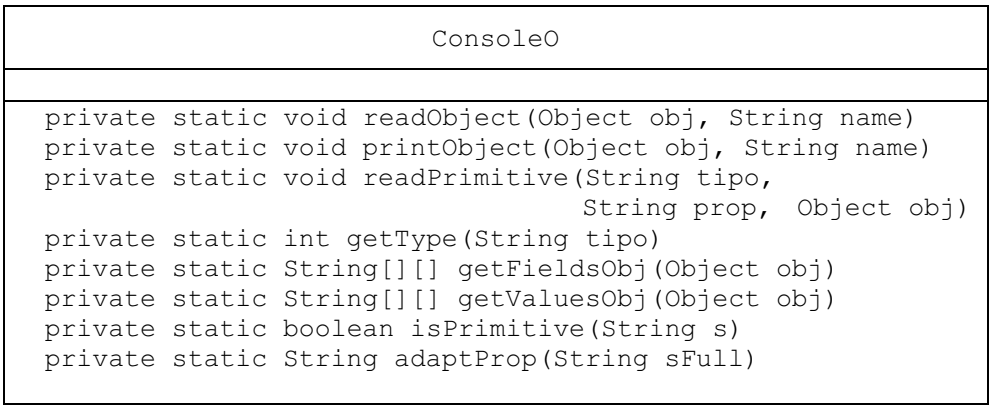


FIGURA 4.21 - Métodos privados da classe `ConsoleO`

A interface da classe `ConsoleO` é bastante simples, possuindo apenas três métodos, os quais necessitam dos métodos definidos na classe `Infs_Reflex`. Os métodos da classe `Infs_Reflex` são todos reflexivos, isto é, utilizam a técnica de reflexão computacional para realizar introspecção ou intercessão sobre as propriedades e métodos de um objeto.

O método `readAll(complexObj)` é um dos principais métodos da `ConsoleO`, sendo responsável pela execução da leitura de todas as propriedades declaradas em um determinado objeto definido pelo usuário. Os passos que o método `readAll` segue para realizar a leitura de um objeto são:

a) identificação das propriedades existentes no objeto complexo passado como parâmetro;

b) identificação inicial das propriedades que são de algum dos tipos padrão, considerando: primitivos, *strings* e *wrappers*;

c) realização da leitura dos tipos identificados no passo (b);

d) identificação das propriedades do objeto em questão que são de tipos complexos, ou seja, são outros objetos;

e) instanciação de um novo objeto do tipo identificado no passo (d);

f) execução, de forma recursiva, dos passos anteriores, mas desta vez o objeto que foi instanciado no passo (e) será o parâmetro para o passo (a) disparar novamente as identificações necessárias.

O método `printAll(complexObj)` praticamente faz a operação inversa do método `readAll()`. Neste caso é necessário imprimir os valores que as propriedades de um determinado objeto possui, ou seja, imprimir o estado do objeto do usuário. Este método foi disponibilizado em duas versões, a primeira (`printAll(complexObj)`) exibe apenas os valores das propriedades na ordem em que foram declarados na classe do objeto, e a segunda versão (`printAll(complexObj, show)`) onde é possível configurar a exibição ou não dos nomes das propriedades juntamente com o conteúdo do objeto. Supondo a impressão de um objeto que possua uma propriedade `dataNasc.dia`, quando o segundo parâmetro for `true`, ele exibirá `dataNasc.dia->7`, quando for `false`, exibirá apenas o número `7`.

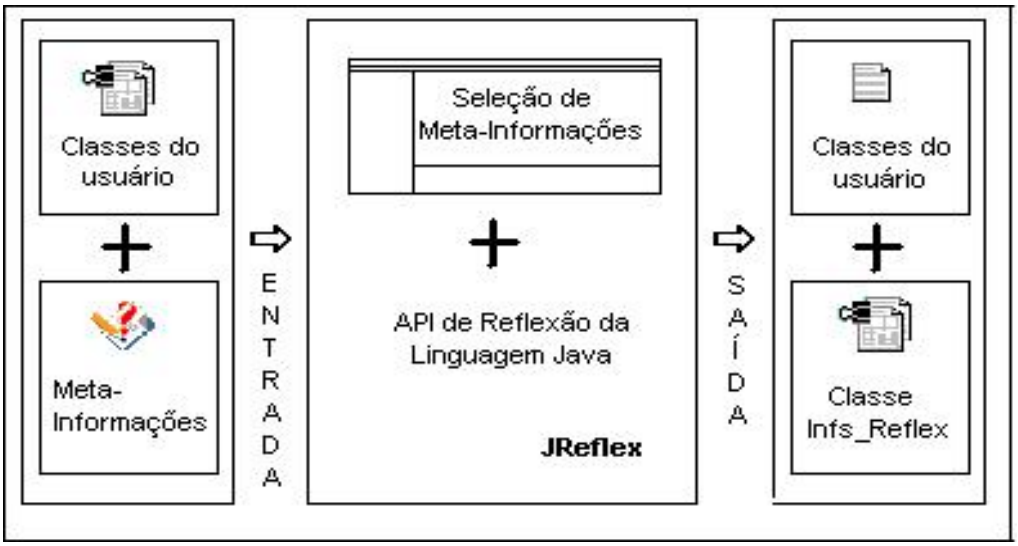


FIGURA 4.22 - Gerando Classe e Métodos Reflexivos

A classe `Infs_Reflex` (figura 4.23) foi gerada utilizando-se o ambiente JReflex [JRE 2002] [BER 2000a], que é um gerador de *software* genérico reflexivo. Na verdade, ele é um ambiente integrado de desenvolvimento (IDE) que automatiza a geração de código reflexivo. Essa classe é gerada automaticamente, sendo adaptável a qualquer tipo de aplicação [BER 2000b]. O usuário do ambiente seleciona de forma interativa, através de uma interface visual, as meta-informações que deseja obter e o ambiente gera, na classe `Infs_Reflex`, os métodos que as implementam (figura 4.22).

Infs_Reflex
public static String[] findFieldTypeName(Object obj) private static Object expand(Object str) private static String[] getInheritedFieldsTypeName(Class cl) public static void alterField(String fieldName, Object obj, Object value) private static boolean alterDeclaredField(String fieldName, Object obj, Object value) private static boolean alterInheritedField(String fieldName, Object obj, Object value) public static String[] findFieldValues(Object obj) public static Object newObject(String className, Object[] parameterTypes, Object[] parameters)

FIGURA 4.23 - Métodos da classe `Infs_Reflex`

Existem várias vantagens em utilizar o ambiente JReflex, como auxílio no desenvolvimento de implementações que necessitem de informações que somente são obtidas através do mecanismo de reflexão computacional, são elas:

- o JReflex foi desenvolvido localmente no Instituto de Informática da UFRGS e está disponível como *software* livre;
- ele mantém os princípios de portabilidade proporcionados pela linguagem Java, uma vez que foi totalmente implementado na linguagem Java com a utilização das classes do pacote Swing;
- apresenta fácil manutenção;
- gera automaticamente métodos que fornecem as informações reflexivas necessárias;
- os métodos da classe `Infs_Reflex` (figura 4.23), quando necessário, buscam os atributos privados, protegidos e públicos em toda a hierarquia de classes.

Outras vantagens encontradas em utilizar o ambiente JReflex estão no pouco conhecimento que o programador deve possuir sobre a API de reflexão da linguagem Java, na confiabilidade do código gerado e na rapidez apresentada para gerar a classe reflexiva.

4.3 Suporte ao Professor: Catálogo de Exemplos

A idéia de elaborar este catálogo surgiu após a leitura de diversos artigos onde foi constatada uma certa dificuldade, por parte dos professores, em migrar de um paradigma de programação para outro. Esta complexidade se agrava ainda mais quando, além da troca de paradigmas, envolve a troca de linguagens. Esse tipo de migração envolve tempo de amadurecimento e experiência de uso da nova tecnologia, o que se torna extremamente preocupante. Este assunto já foi amplamente abordado em algumas das seções anteriores.

Esta ajuda é composta por tópicos freqüentemente abordados nos planos de ensino das disciplinas introdutórias de programação e por um catálogo de exercícios referente a estes tópicos. Todos os exercícios são implementados em Java e possuem o seu correspondente em Pascal. Os exemplos utilizam as classes "simplificadoras" do pacote `ufrgs.inf.iosimple`. Este tipo de trabalho visa aproveitar o conhecimento que o professor já possui em Pascal, visto esta ser considerada a linguagem mais utilizada em disciplinas introdutórias, exibindo a solução implementada em Java de maneira rápida e prática. A bibliografia disponível atualmente aborda as linguagens de forma separada, ficando sob responsabilidade do professor fazer a analogia necessária entre Pascal e Java.

O catálogo está disponível para acesso de dentro do ambiente JEduc, através do menu HelpJEduc (figura 4.24). Este menu somente estará visível quando o JEduc for executado em seu módulo implementado para uso dos professores. É possível também visualizar o catálogo utilizando qualquer navegador de internet, pois este foi totalmente implementado no formato de páginas HTML (*HyperText Markup Language*).

As cores da interface deste catálogo foram definidas sempre com a preocupação de não sobrecarregar as telas e facilitar a impressão, caso isso seja necessário. A cor de fundo definida foi o branco com as letras em preto. No início de cada tela existe uma barra com *links* para as outras que compõem o catálogo, bem como os tópicos disponíveis para acesso. A definição da navegação nas telas ficou bastante simples e intuitiva, propiciando assim acesso rápido à opção procurada.

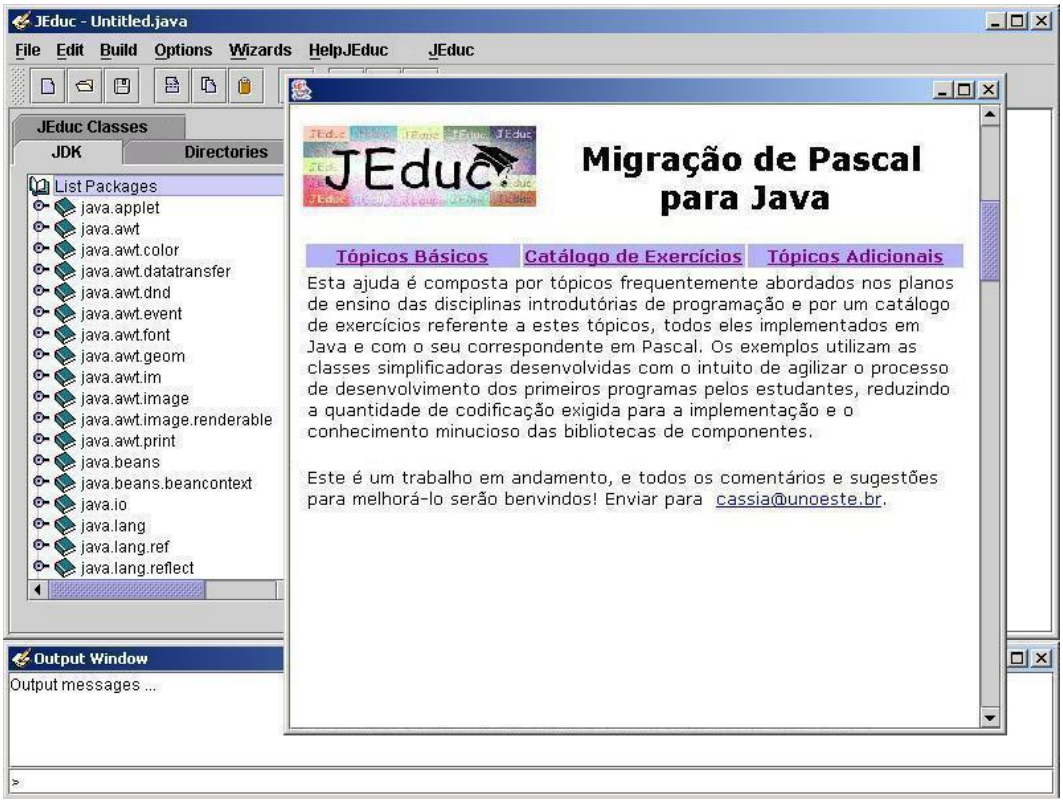


FIGURA 4.24 - Tela de abertura do catálogo

Na tela de abertura do catálogo estão disponíveis três *links*: Tópicos Básicos, Catálogo de Exercícios e Tópicos Adicionais. Foram assim divididos para atender as necessidades do usuário, onde este pode acessar apenas a parte teórica de um determinado tópico, ou somente a respectiva implementação dos exercícios propostos. Nas próximas subseções o objetivo e a finalidade destes *links* de navegação são melhor explicados.

4.3.1 Tópicos básicos

Nesta tela do catálogo estão os tópicos que normalmente compõem os planos de ensino das disciplinas introdutórias de programação (figura 4.25). Neste caso, os tópicos foram definidos com base nos planos de ensino das seções 3.3.1 e 3.3.2. Desta forma, foi considerado o conteúdo visto em uma disciplina que segue o paradigma procedimental, com utilização da linguagem Pascal, e, também o de uma disciplina em que a base é o paradigma orientado a objetos, com aplicação da linguagem Java. Pôde ser observada quase que uma padronização do conteúdo a ser ensinado nestas disciplinas, independente do paradigma ou da linguagem, dos conceitos a serem

assimilados pelos alunos. Esta seqüência, com exceção da aplicação das particularidades da programação orientada a objetos, foi definida e está sendo utilizada por diversas instituições há algum tempo.

FIGURA 4.25 - Tópicos disponíveis na tela “Tópicos Básicos”

Em Java:

```
import ufrgs.inf.iosimple.*;
public class TesteEC {
  public static void main (String[] args) {
    ConsoleP.println("Informe o n° de Alunos: ");
    int i = ConsoleP.readint();
    ConsoleP.println(i==1 ? "Existe apenas um aluno"
                          : "Existem vários alunos");
}}
```

Em Pascal:

```
program TesteEC
var i: integer;
begin
  write('Informe o n° de Alunos:');
  readln(i);
  if i = 1 then
    writeln('Existe apenas um aluno')
  else
    writeln('Existem vários alunos');
end.
```

FIGURA 4.26 - Comando de seleção como expressão condicional

Cada tópico disponível possui uma breve explicação de seu conteúdo, cujo enfoque principal é abordagem realizada na linguagem Java em comparação à abordagem na linguagem Pascal (figura 4.27). A principal ajuda proporcionada por este catálogo aos professores é a exploração das diferenças existentes entre comandos similares disponíveis nas duas linguagens, ou comandos que estão disponíveis apenas na linguagem Java. Por exemplo, em Java existe um comando de seleção que pode ser utilizado como expressão condicional; já a linguagem Pascal não possui o correspondente. A figura 4.26 mostra um exemplo de aplicação do comando de seleção como expressão condicional em Java. Em Pascal, para obter o mesmo resultado, é necessário utilizar o comando de seleção dupla `if..then..else`.

Este catálogo não pretende ensinar Java nem substituir livros texto. É necessário, para obter esclarecimentos mais aprofundados, consultar outras fontes de conhecimento. O objetivo desta tela, conforme já mencionado, é abordar cada conteúdo de maneira bastante direta, e sempre que possível, proporcionando dicas práticas.

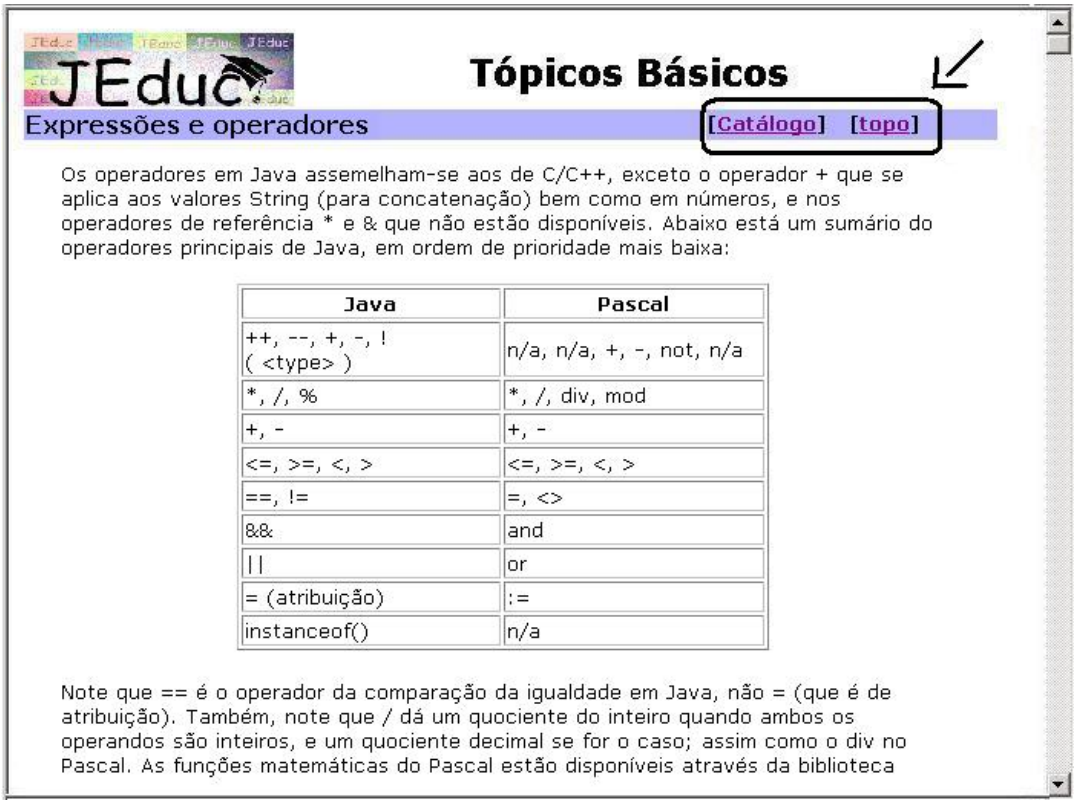


FIGURA 4.27 - Forma de abordar os tópicos

A navegação definida em cada tela é bastante funcional, o título de cada tópico está destacado, com cor diferente, do restante do texto. À direita de cada título são localizados dois *links* (seta indicativa na figura 4.27), um para acessar diretamente os exercícios implementados (na tela "Catálogo de Exercícios") do tópico em questão, e outro para posicionar no topo da página, possibilitando assim, a seleção de um outro tópico para consulta. Em alguns tópicos de assuntos mais informativos, no entanto, o

link “Catálogo” não estará disponível, indicando assim que neste caso não existem exercícios implementados.

4.3.2 Catálogo de exercícios

Esta tela do catálogo também segue o padrão de navegação definido na tela dos tópicos básicos. O objetivo principal é fornecer alguns exercícios implementados em Java com o respectivo correspondente em Pascal. Esses exemplos foram coletados de diversos *sites* e livros relacionados ao ensino introdutório de programação [DEI 2001] [MAR 94] [GOT 94] [MAN 2000] [RIC 2001] [LEW 2000] [HOR 2001]. Foram também agrupados por tópicos, conforme pode ser observado através da figura 4.28.

FIGURA 4.28 - Tópicos disponíveis na tela “Catálogo de Exercícios”

Na figura 4.29, é ilustrado um exemplo da organização desta tela, evidenciando primeiramente a implementação dos exemplos em Java e depois em Pascal. Em cada exercício existe um parágrafo, identificado como “Funcionalidade”, que explica o objetivo a ser atingido com a implementação deste. É possível também acessar, através do *link* “Tópicos Básicos”, a explicação teórica referente ao tópico do exercício. Todos os exemplos foram testados no ambiente JEduc, com isso, é possível executá-los facilmente. Para que isso ocorra basta apenas selecionar a região desejada na tela do catálogo de exercícios, copiar via área de transferência, colar no editor de código fonte do JEduc e, finalmente, selecionar a opção para compilar e executar o programa.

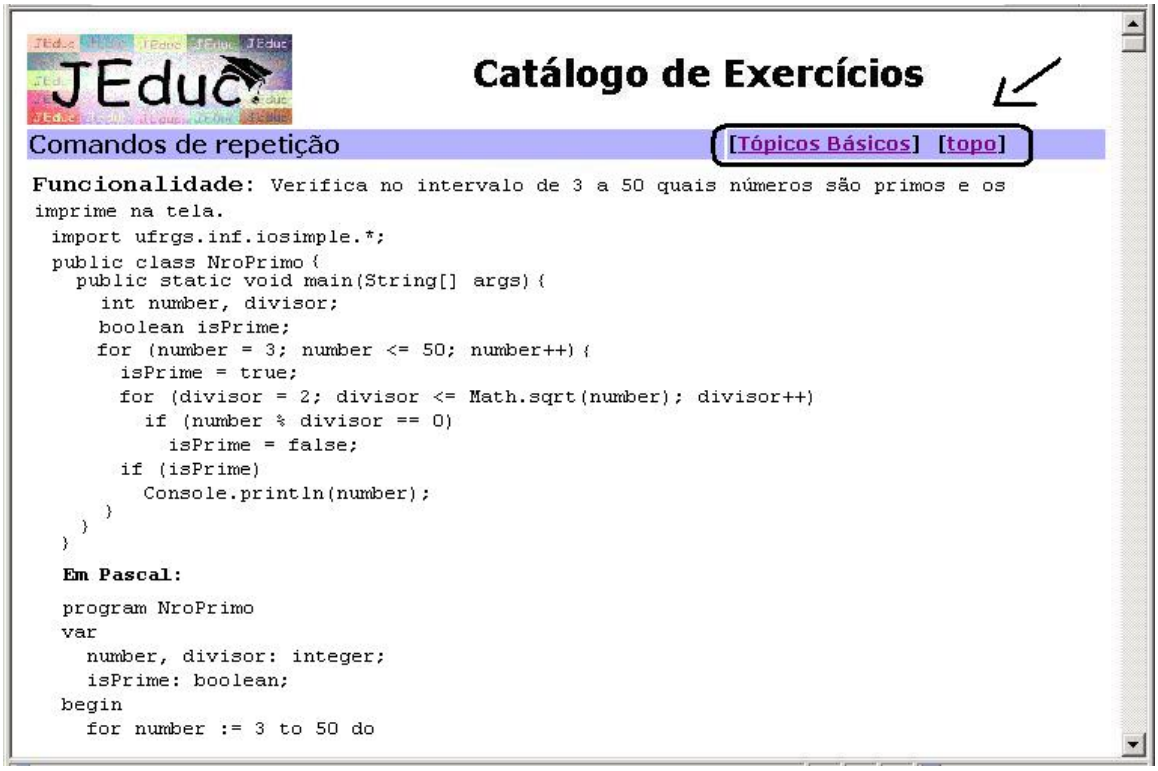


FIGURA 4.29 - Exemplos práticos

4.4 Resumo do capítulo

Os componentes descritos neste capítulo podem ser usados e alterados de forma totalmente independente do ambiente JEduc, descrito em detalhes no próximo capítulo.

As classes simplificadoras foram reunidas em um pacote que pode ser incorporado em qualquer ambiente de desenvolvimento ou adicionado à biblioteca de classes Java.

Esta é apenas a primeira versão do catálogo de suporte ao professor, por isso é considerado um trabalho em andamento. Ele foi submetido apenas às opiniões das autoras, ainda são necessárias uma maior divulgação e aplicação prática, visando possibilitar sua melhora através dos futuros comentários e sugestões. Este catálogo é uma tentativa de auxiliar os professores em um momento crítico, como o de migração.

5 JEduc: Ambiente Dedicado ao Ensino de Java

Neste capítulo são introduzidas algumas características do ambiente JEduc, as quais representam adequações decorrentes da derivação da ferramenta IDECoRe. O ambiente JEduc encontra-se em fase final de testes e está disponibilizado, juntamente com o pacote das classes "simplificadoras" e o catálogo de exemplos, em [JED 2002].

5.1 Objetivos

O ambiente JEduc [JED 2002] é completamente implementado em Java e seus componentes visuais utilizam as classes do pacote Swing. Ele foi desenvolvido visando minimizar alguns problemas relacionados ao ensino de Java como primeira linguagem de programação, bem como propiciar um IDE fácil de instalar e utilizar, proporcionando uma pequena curva de aprendizado. Dentre os diversos problemas levantados, esse ambiente procura resolver os seguintes:

- simplificar algumas características da linguagem Java;
- agilizar o processo de desenvolvimento dos primeiros programas em Java pelos alunos iniciantes em programação;
- reduzir o conhecimento minucioso das bibliotecas de componentes da API da linguagem Java, principalmente as bibliotecas de entrada e saída;
- disponibilizar um ambiente e um pacote de classes para auxiliar alunos e professores;
- facilitar aos professores a administração das tarefas de ensino;
- amenizar a complexidade da transição pedagógica entre os modelos de programação, quando se aplicar.

O que motivou o desenvolvimento do ambiente JEduc (figura 5.1) e do suporte que ele oferece, foi a constatação que um grande número de cursos de graduação em computação estão avaliando a possibilidade de iniciar o ensino de programação com o uso do paradigma orientado a objetos e de Java como linguagem de programação [STE 2001]. Prevê-se que muitos professores devem migrar de Pascal para Java. A migração envolve, além da capacitação na linguagem, a adoção de novo referencial conceitual. Propõe-se amenizar esta transição de três maneiras:

- por comparação - como se faz em Pascal e como se traduz para Java;
- por ocultamento - bibliotecas simplificadoras que tornam desnecessário, em um primeiro momento, explicar conceitos complexos;
- por simplificação - disponibilizando um ambiente integrado de desenvolvimento simples, amigável e livre.

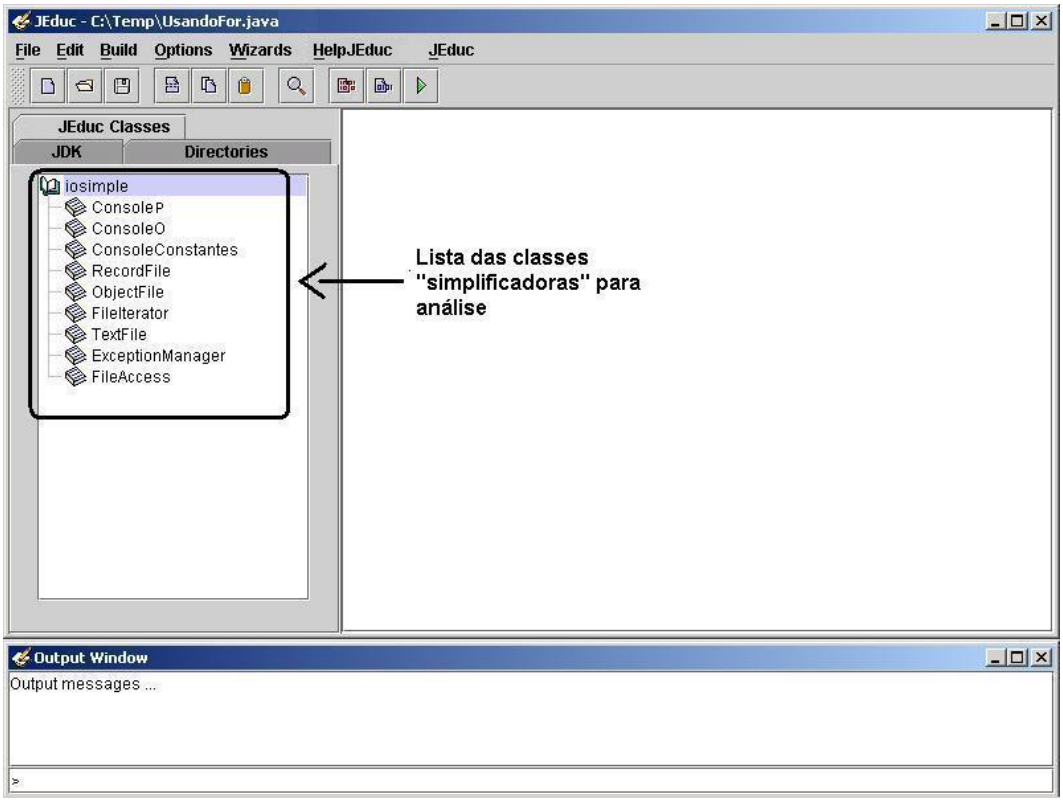


FIGURA 5.1 - JEduc: usando as classes “simplificadoras”

5.2 Origens: IDECoRe

Conforme abordado na seção 2.3.2, o JEduc foi implementado realizando uma extensão da ferramenta IDECoRe. Várias funcionalidades ligadas ao ensino da linguagem Java foram incluídas, mas as funcionalidades básicas do IDECoRe foram preservadas. A seguir são listadas as principais características do ambiente, onde são apontadas as preservadas do IDECoRe e indicadas as adaptadas ou adicionadas:

- editor de programas – característica preservada: permite tanto ao ambiente quanto ao usuário fazer alterações no código fonte dos programas, que, posteriormente serão compilados e executados pela máquina virtual Java;
- visualizador de pacotes – adaptação do componente “*tree view*”: exibe as características (variáveis, métodos e construtores) de qualquer classe disponível nos pacotes fundamentais da API padrão da linguagem Java. No módulo do aluno, estão apenas disponíveis as classes dos pacotes `java.io`, `java.lang` e `java.util`; já no módulo do professor estão disponíveis todas as classes da API padrão;
- gabaritos sintáticos – funcionalidade adicionada: visa reduzir a ocorrência de erros e facilitar o uso da linguagem Java;
- classes “simplificadoras” – adaptação do componente “*tree view*”: exibe a lista das classes implementadas para facilitar a entrada e saída de tipos primitivos, *String*, tipos invólucros, objetos definidos pelo aluno e arquivos na linguagem Java;

- diretório de trabalho – adaptação do componente "*tree view*": lista para o usuário os arquivos `.java` existentes no atual diretório de trabalho;
- tratamento das mensagens de erro – funcionalidade adicionada: em seu módulo desenvolvido para os alunos existe um filtro, reduzindo o número de linhas das mensagens de erros a serem exibidas;
- assistente para o professor – funcionalidade adicionada: um assistente, no formato de páginas HTML, cujo objetivo é auxiliar o ensino da linguagem Java como primeira linguagem de programação. Contém um catálogo de programas com exemplos de codificação em Pascal e em Java, apontando diferenças conceituais.

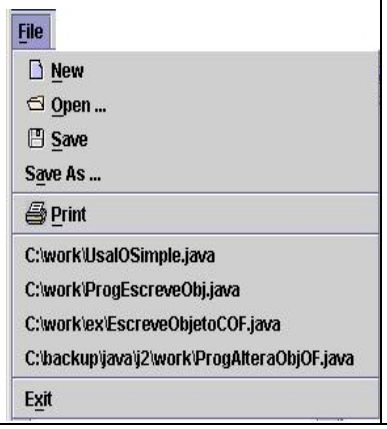
5.3 Módulo Professor e Módulo Aluno

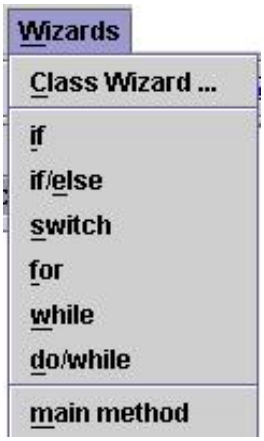
No decorrer deste estudo foram desenvolvidos dois módulos do JEduc, o do aluno e o do professor. No módulo do aluno está disponível um ambiente leve de Java, específico para iniciantes em programação, com possibilidade de utilização das classes "simplificadoras", indicadas através de um *inspector* (figura 5.1), com filtro de mensagens de erros e acesso à documentação de apenas um subconjunto dos pacotes da API padrão de Java.

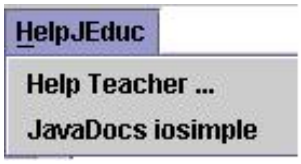
Já no módulo do professor, além de estar disponível todo o ambiente do módulo do aluno sem filtros, está apresentado, sob a forma de documentação HTML, todo um estudo de transcrição dos exemplos de Pascal para Java do conteúdo a ser ensinado numa disciplina introdutória de programação em cursos de computação.

As opções disponíveis no menu principal foram identificadas e definidas visando facilitar e agilizar o uso do ambiente. As opções de menu mais freqüentemente utilizadas estão inseridas na barra de ferramentas. A funcionalidade das opções estão descritas na tabela 5.1. O ambiente, mostrado em sua versão original em inglês, pode ser apresentado também em uma versão em português.

TABELA 5.1 - Opções do menu principal do ambiente JEduc

Menu	Opções	Funcionalidade
File		Neste menu estão disponíveis opções para manipulação dos arquivos de código fonte, ou seja, do editor de textos. As opções são: New, Open, Save, Save As, Print e Exit. Além disso, existe a possibilidade de visualizar os últimos 4 arquivos que foram abertos no ambiente JEduc.

Edit		As opções disponíveis neste menu são as seguintes: Cut, Copy, Paste, Select All e Find. Todas estão relacionadas ao uso do editor de código fonte.
Build		Neste menu estão as facilidades fundamentais deste IDE, que é a possibilidade de compilar e executar programas Java. A opção Compile executa o `javac` (compilador) no arquivo que está aberto no editor. A opção Execute executa o `java` (interpretador) também no arquivo aberto. Já a opção Run engloba as funcionalidades das duas opções anteriores, compila e executa o arquivo aberto. Em todas as opções os usuários poderão fazer uso de teclas de atalho `Alt+F5`, `Alt+F7` e `Alt+F9`, respectivamente.
Options		Este menu foi montado visando possibilitar a personalização do ambiente. As opções disponíveis são: Editor (Font e Background Color) para personalizar o editor de código fonte; Look & Feel para mudar o padrão de cores da visualização gráfica do ambiente; e View Output, para que a janela responsável pelos *inputs* e *outputs* dos programas seja exibida quando outra janela a estiver sobrepondo.
Wizards		O menu Wizards foi incorporado ao JEduc devido à constatação da dificuldade que os alunos iniciantes têm em decorar sintaxe de comandos. O objetivo principal deste menu é facilitar esta tarefa a partir da inserção de código com a sintaxe dos principais comandos da linguagem Java: `if`, `if/else`, `switch`, `for`, `while`, `do/while` e do método `main`. Possui também a opção Class Wizard, que insere o código fonte completo da declaração de uma classe a partir dos dados informados pelo usuário.

HelpJEduc		Esta opção é acessível apenas no módulo do professor. O objetivo é proporcionar acesso direto e rápido ao catálogo de exemplos desenvolvido em formato HTML e à documentação das classes "simplificadoras", gerada através da ferramenta javadoc.
JEduc		É apenas uma caixa do tipo *About* com informações sobre: a versão do JEduc, os desenvolvedores e o local onde está disponível para *download*.

5.4 Simplificação proporcionada com o uso do JEduc

A partir dos problemas e dos requisitos enumerados na Seção 2.3, conclui-se que para uso acadêmico o ambiente de desenvolvimento não pode ser complexo, sob pena de fazer com que os professores percam muito tempo para explicar o funcionamento do IDE, ao invés de concentrar-se nos exemplos e conceitos de programação [PER2002b].

Ao ser ativado, o ambiente JEduc exibe uma janela principal, a partir da qual pode ser criado um novo programa ou aberto um programa existente.

Alguns cuidados foram tomados com relação à excessiva quantidade de ícones e elevado número de opções de menus, pois a grande quantidade de elementos visuais sobrecarrega a interface com funcionalidades desnecessárias em um primeiro momento no ensino introdutório de programação.

As simplificações indicadas na figura 5.2, consistem em reduzir o número de elementos visuais, existentes na maioria dos ambientes de programação, visando maximizar a usabilidade do ambiente JEduc. Dessa forma, alunos e professores podem preocupar-se apenas com o ensino-aprendizagem dos conceitos básicos de programação, sem que um tempo maior seja dispendido no aprendizado do funcionamento do IDE. O objetivo deste deve ser de apenas intermediar o processo de aplicação prática dos conceitos teóricos vistos em aula.

Outro item retirado da especificação do ambiente JEduc é a definição de "projetos", requisito obrigatório em vários IDEs, sempre que o aluno necessita implementar algum programa. Esta simplificação visual permite que o aluno iniciante visualize apenas unidades simples de código fonte, pois no início de uma disciplina de introdução à programação é solicitado apenas que sejam implementados exemplos isolados com conceitos. Somente depois, quando o aluno possuir mais experiência e familiaridade com a programação, os conceitos isolados são cobrados em implementações maiores e mais elaboradas.

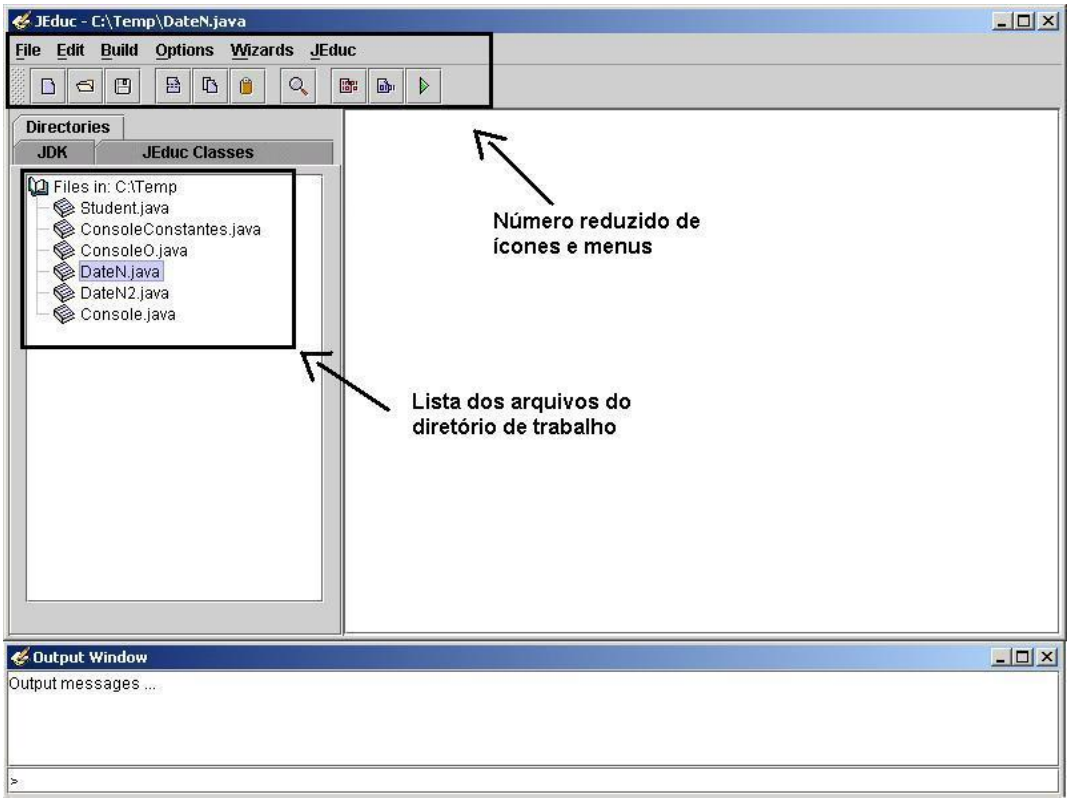


FIGURA 5.2 - Simplificações Visuais

Além disso, a ativação de JEduc no módulo Aluno apresenta um conjunto reduzido de informações quando comparada à ativação no módulo 'Professor':

- esconde o item de menu "HelpJEduc";
- ao ser solicitada a visualização de classes padrão da linguagem Java exibe apenas três pacotes de classes: `java.io, java.lang` e `java.util`;
- quando ocorre um erro de execução são exibidas apenas as mensagens de erro mais significativas referentes ao programa do usuário.

A seguir são mostrados alguns cenários de utilização do JEduc, destacando as principais simplificações adotadas.

Cenário 1: Entrada de dados via Console

A figura 5.3 ilustra um programa em execução, no momento da solicitação de uma entrada de dados via teclado através da classe `ConsoleP`. Observe-se que o valor de entrada é solicitado na parte inferior da "Output Window".

É importante notar que o tratamento de exceções é realizado localmente nas classes "simplificadoras", não sendo necessário ao aluno iniciante o conhecimento sobre exceções, poupando também o professor de ensinar os complexos mecanismos para realizar todo esse tratamento. Caso ocorra algum erro de entrada (por exemplo, a entrada de um tipo incorreto de dado), a classe `ConsoleP` irá detectar e solicitar uma nova entrada de dados, sem interromper a execução do programa.

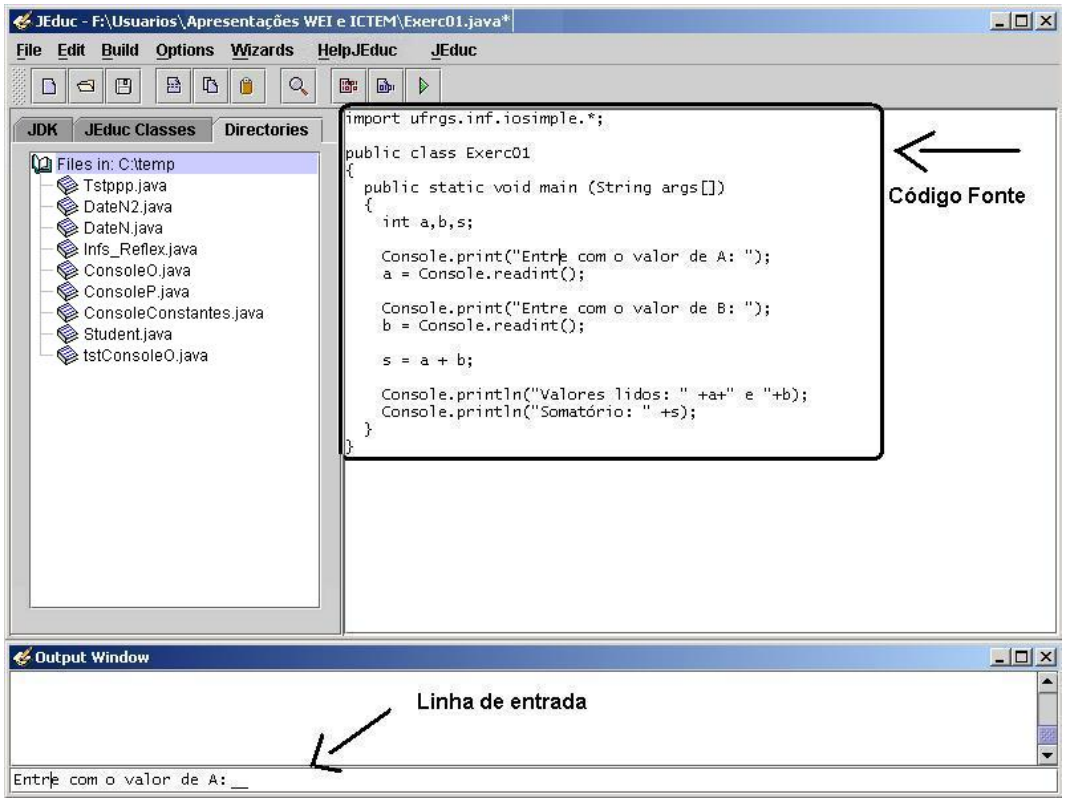


FIGURA 5.3 - Exemplo de uma entrada no JEduc

Cenário 2: Erro na execução do programa

Na fase de execução, os erros diagnosticados pelo ambiente de execução de Java são exibidos de forma bastante completa, mostrando o estado de toda a pilha de execução no momento da ocorrência do erro. Estas informações incluem todas as invocações de métodos de classes que foram usadas pelo ambiente, ou seja, que não foram explicitamente usadas no programa. Tal prolixidade de informações, conquanto que úteis a programadores experientes, são incompreensíveis e traumatizantes para os novatos.

O ambiente JEduc, em seu módulo desenvolvido para os alunos, inclui um filtro de mensagens de erros, reduzindo o número de linhas a serem exibidas. Já no módulo desenvolvido para os professores, é exibido todo o estado da pilha de execução, como ocorre nos demais IDEs, pois para este tipo de usuário é importante que ele tenha uma visão total dos erros detectados.

Cenário 3: Uso de gabaritos sintáticos

Existem outros problemas relacionados à sintaxe de Java para iniciantes. São vários os conceitos a serem compreendidos ao mesmo tempo, começando pela declaração do programa principal,

```
public static void main (String argv[])
```

esta construção envolve conceitos de métodos, visibilidade (`public`), métodos de classe (`static`) e instância, tipos de retorno (`void`), nomes de método (`main`), parâmetros (`argv`) e vetores (`[]`). É necessário atacar este problema por partes, inicialmente explicando o que essa declaração significa e tentando assegurar que esta assinatura de método seja sempre reproduzida de modo correto. Há também a complexidade envolvida na sintaxe das estruturas de controle de seleção e iteração, ocasionando freqüentes erros sintáticos. Visando reduzir a ocorrência de erros e buscando facilitar o uso da linguagem, foi embutido no ambiente JEduc um menu específico para a inserção de esqueletos de código - os gabaritos sintáticos. A figura 5.4 ilustra o menu Wizards do JEduc e alguns exemplos inseridos no editor de programas.

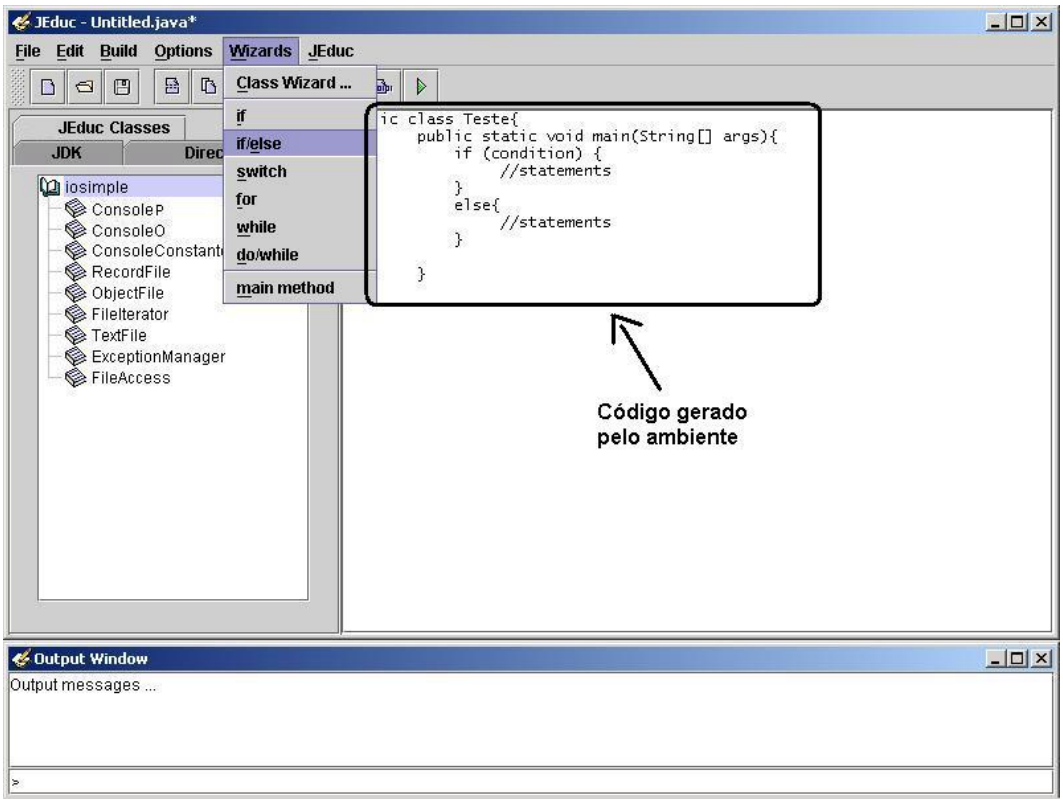


FIGURA 5.4 - Menu Wizards do JEduc

Cenário 4: Pacotes da API padrão de Java

Devido ao grande número de pacotes disponíveis na implementação da API padrão da linguagem Java, alunos iniciantes sentem dificuldade para localizar informações úteis. Os novatos não compreendem o limite entre a linguagem e os pacotes. A solução encontrada foi disponibilizar, através do ambiente JEduc, a visualização apenas dos pacotes fundamentais, figura 5.5, ocultando do aluno pacotes destinados a aplicações mais avançadas. Os demais pacotes deverão ser utilizados no decorrer do curso de graduação, quando os conceitos básicos de programação já foram corretamente assimilados, neste momento, entende-se que o aluno já possui maturidade suficiente para abstrair e compreender conceitos mais complexos.

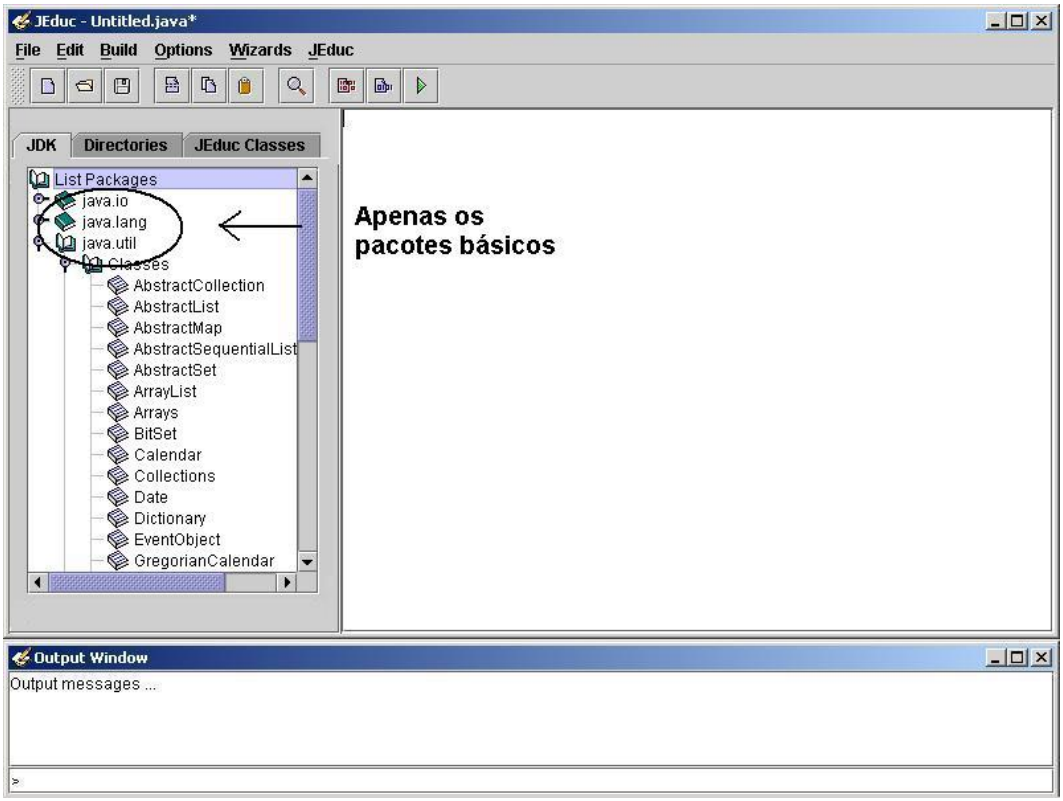


FIGURA 5.5 - Pacotes básicos: Módulo Aluno

5.5 Particularidades do Módulo Professor

A figura 5.6 destaca as características implementadas na versão do JEduc disponível para os professores:

- Menu HelpJEduc com as opções que acessam o catálogo de exemplos e a documentação das classes "simplificadoras" do pacote `ufrgs.inf.iosimple`;
- Acesso à visualização de todas as classes disponíveis na API padrão de Java através da aba JDK do *inspector*;
- Visualização de todas as linhas das mensagens de erro geradas durante a execução do programa.

Além disso, este modo de ativação também disponibiliza o acesso a um catálogo de exemplos desenvolvido em formato HTML (figura 5.7). Conforme descrito na Seção 4.3, esse catálogo de exemplos contém a codificação de programas em Pascal e em Java, apontando diferenças conceituais. Os exemplos do catálogo foram extraídos de programas de ensino de disciplinas introdutórias em diversas universidades. A intenção desse catálogo é permitir a apresentação rápida de exemplos, exercícios propostos e soluções.

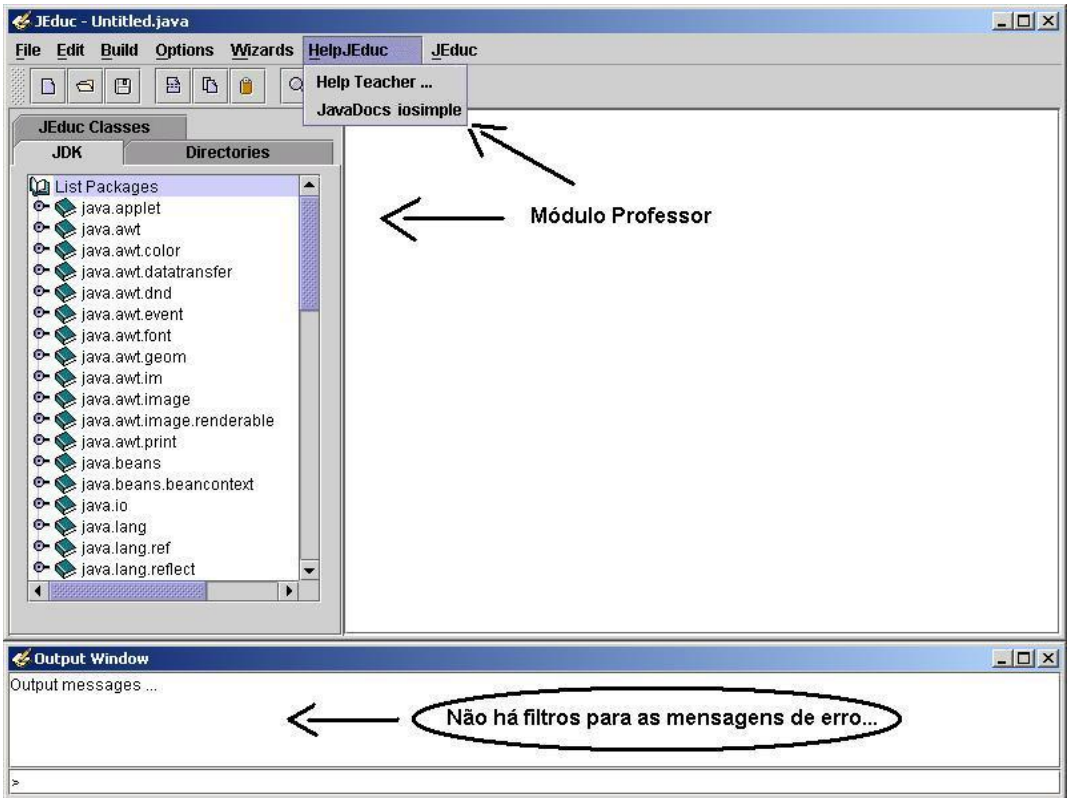


FIGURA 5.6 - particularidades do Módulo Professor

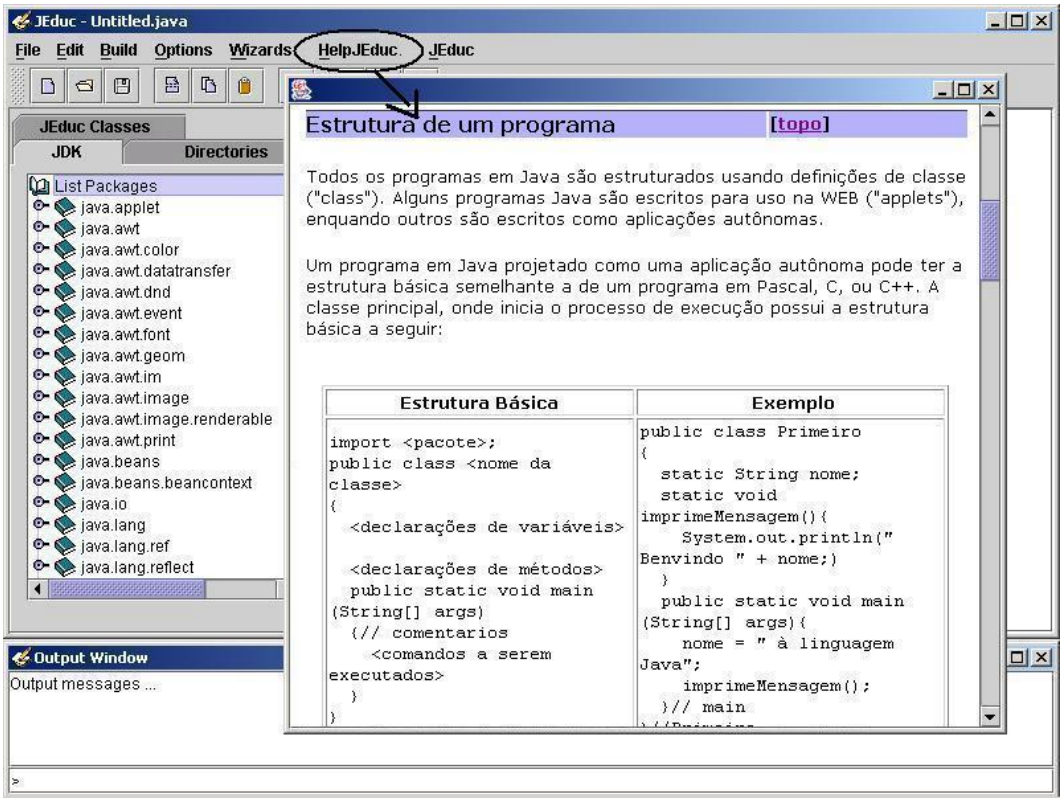


FIGURA 5.7 - Ajuda ao professor em formato HTML (Módulo Professor)

5.6 Resumo do Capítulo

Este capítulo descreveu o ambiente JEduc desde a sua concepção (derivação de IDECoRe) até seu funcionamento como um todo: descrevendo as telas, as opções do menu, os modos de ativação para alunos e professores, abas do *inspector* e finalmente a simplificação proporcionada para uso, como IDE, em disciplinas introdutórias de programação.

6 Conclusões

Alguns dos problemas relacionados ao ensino dos conceitos de programação orientada a objetos são apontados, e, em particular, problemas relacionados à linguagem Java. Soluções são propostas no sentido de amenizar a sintaxe, reduzir a quantidade de codificação e o conhecimento minucioso das bibliotecas de componentes desta linguagem. Em paralelo, evidencia-se uma preocupação quanto à capacitação de professores para ensinar disciplinas introdutórias baseadas no modelo de objetos.

Java tem vantagens significativas, não somente como uma linguagem comercial, mas também como uma linguagem didática (voltada ao ensino). Permite que os alunos aprendam programação orientada a objetos sem exposição às complexidades existentes em outras linguagens.

Com a simplicidade de uso da linguagem Java apresentada neste estudo, proporcionada através desenvolvimento de um IDE apropriado (JEduc), do uso das classes "simplificadoras" e apoio aos professores, com o catálogo de exemplos, ilustra o desejo de contribuir no conjunto de discussões, tutoriais e artigos dedicados a estimular, atualmente, a adoção de Java como a primeira linguagem de programação.

As principais vantagens no uso do JEduc são: exigência de pouca memória para execução; instalação simples (é necessário apenas informar a localização física das ferramentas – interpretador e compilador – na máquina); reduzido número de ícones e menus, visando simplificar a interface visual; os programas são desenvolvidos de forma independente sem a necessidade de vinculação à "projetos"; e, devido à simplicidade proporcionada, permitir rápida adaptação do usuário ao ambiente. Desta forma, pretende-se assegurar a solução de alguns problemas descritos nesta dissertação.

O ambiente JEduc foi definido seguindo a filosofia de *software* livre, porque "... o *software* livre oferece um benefício mais profundo para a educação: o conhecimento embutido no *software* livre é de conhecimento público, não secreto. A caixa preta selada de um sistema de *software* proprietário é projetada para manter as pessoas no escuro" [KUH 2001]. A idéia é propiciar aos alunos mais avançados na linguagem Java, a possibilidade de estudar o ambiente que eles utilizam no desenvolvimento de seus programas, para aprender seu funcionamento. E assim, se necessário, escrever melhorias o integrar novas funcionalidades, gerando um *software* cada vez mais funcional e de melhor qualidade [BRO 2002].

Ainda não foi possível aplicar o estudo realizado, através da implementação do JEduc, das classes "simplificadoras" e do catálogo de exemplos em nenhuma turma iniciante dos cursos de computação. Este tipo de experimento demandaria muito tempo, no mínimo um semestre, e, mesmo assim, o resultado estaria baseado em apenas um grupo, o que é pouco. A solução encontrada foi a de expor o ambiente e o pacote de classes, de forma mais rápida, a alunos de iniciação científica com bons conhecimentos em programação estruturada e noções básicas de programação orientada a objetos. Desta iniciativa resultou um conjunto de sugestões para a melhoria do ambiente a serem implementadas em trabalhos futuros.

6.1 Trabalhos futuros

As ferramentas de programação apresentam constante evolução, deixando o seu desenvolvimento sempre sujeito a aprimoramentos e extensões. Pretende-se continuar trabalhando no IDE do JEduc, realizando várias melhorias identificadas ao longo de todo este estudo, como:

- um reconhecedor que identifique e destaque as palavras-chave da linguagem Java, de forma que a leitura e a escrita de programas se torne mais clara e fácil;
- implementação de um depurador;
- projeto e desenvolvimento de mais componentes e *wizards,* pensando também nos alunos, para facilitar o ensino de tópicos básicos como: conceitos de classes, métodos, visibilidade, etc.; e tópicos mais avançados como: GUIs, programação para WEB, programação concorrente e distribuída, etc. Sem comparação com a linguagem Pascal;
- realização de um estudo mais detalhado sobre a usabilidade da ferramenta;
- disponibilizar a tradução de pseudocódigo em um código fonte de Java [SOU 2001];
- implementação de um gerenciador de atividades, onde o professor propõe atividades aos alunos em um laboratório e os monitora através de verificações *on-line,* ocorridas diretamente no micro de cada aluno;
- disponibilizar um vizualizador da hierarquia de classes, onde podem ser vistos os membros das classes;
- melhoramentos gerais na interface. Opiniões neste sentido podem ser obtidas dos usuários à medida em que a ferramenta for utilizada;
- melhoria no desempenho do ambiente: o uso dinâmico de reflexão implica maior tempo de processamento; a possibilidade de adotar persistência dos dados reflexivos deve ser estudada.

As indicações de trabalhos futuros aqui realizadas não têm a pretensão de esgotar todas as possibilidades: esta lista de atividades com certeza pode ser estendida. À medida que o tempo passa e novas tecnologias surgem, necessidades se criam e assim novas características e recursos podem ser implementados, visando sempre contribuir para a evolução do ambiente JEduc, do pacote de classes "simplificadoras" e do catálogo de exemplos.

6.2 Considerações finais

Como resultado final, as contribuições mais importantes deixadas por este trabalho, considerando os estudos no sentido de simplificar a aplicação do paradigma orientado a objetos em disciplinas introdutórias de programação e a utilização da linguagem Java, são:

- principal contribuição: projeto e implementação de um IDE, denominado JEduc, onde é possível encontrar os recursos básicos e principais para a

implementação de programas em Java. Disponibilização do pacote `ufrgs.inf.iosimple`, com classes que diminuem algumas das complexidades existentes no uso de Java com aluno iniciantes;

- quando considerados individualmente, os conceitos utilizados nessa ferramenta e no pacote não são totalmente novos. Entretanto, a forma simples em que eles foram organizados resolve problemas ainda não tratados por outras ferramentas;
- atenção dada aos conceitos de engenharia de *software*: reutilização (núcleo IDECoRe), orientação a objetos, padronização da interface, modularidade e encapsulamento;
- preocupação com a capacitação dos professores para enfrentar a migração de paradigmas, através da implementação do catálogo de exemplos.

Em síntese, este trabalho apresentou: (a) um IDE adequado e simples; (b) um pacote de classes "simplificadoras" e (c) um catálogo de exemplos com programas codificados em Pascal e em Java. Todos esses itens buscam, como objetivo comum, facilitar o ensino introdutório de programação orientada a objetos com o uso da linguagem Java. Procura também oferecer um ambiente que aplique os principais conceitos de engenharia de *software*. Apesar do recente tempo de uso e dos poucos testes realizados, conclui-se que a ferramenta é capaz de facilitar o desenvolvimento dos primeiros programas por alunos iniciantes, pois oferece ao usuário uma interface visual simples, intuitiva e amigável.

Anexo 1 Levantamento nas Universidades Brasileiras

Este levantamento ilustrado na tabela A1.1 está disponível em [PER 2001], com o título: "Linguagens de programação adotadas nos cursos de Bacharelado em Ciência da Computação, Engenharia da Computação e Tecnologia de Informática".

Esta pesquisa foi realizada na lista de assinantes da SBC (Sociedade Brasileira de Computação) em Novembro de 2001. O objetivo principal era fazer um levantamento da linguagem de programação utilizada com alunos iniciantes nos cursos de computação das universidades brasileiras, onde foram obtidas cerca de 30 respostas. Para conseguir um número maior de universidades, foi realizado um trabalho de busca dessas informações diretamente nos *sites* de diversas instituições.

TABELA A1.1 - Levantamento sobre linguagens de programação

Instituição	Curso	Primeira Linguagem	Outras Linguagens	Obs.
Associação Catarinense de Ensino http://www.ace.br/	Bacharelado em Ciência da Computação	Delphi, Pascal		*
Centro Universitário La Salle http://www.lasalle.tche.br/	Bacharelado em Ciência da Computação	Pascal		*
Centro Universitário Luterano de Palmas (Tocantins) http://www.ulbra-to.br	Bach. em Sistemas de Informação	Pascal	C, C++ Builder e ASP	
FACIC – Faculdade de Ciência da Computação de Formiga – MG http://www.facic.fuom.br/cursos/cienciadacomputacao/index.php	Bach. em Ciência da Computação	Pascal		
Faculdade de Administração e Informática http://www.fai-mg.br/	Bacharelado em Ciência da Computação	C		*
Faculdade de Ciências da Administração dePetrolina http://www.facape.br/	Bacharelado em Ciência da Computação	Pascal		*
Faculdade Três de Maio – RS http://www.setrem.com.br/fatrem/si/sis_inf.htm	Bach. em Sistemas de Informação	Pascal	C	
Faculdades Integradas de Rondonópolis http://www.unir-roo.br/	Bacharelado em Ciência da Computação	Pascal		*
FURG – Fundação Univ. Federal do Rio Grande http://www.furg.br	Engenharia de Computação	Pascal		
ICMC-USP São Carlos http://www.icmsc.sc.usp.br/	Bach. em Ciência da Computação	Pascal	C	
IME – Instituto Militar de Engenharia http://www.cecilio.ime.eb.br/disciplinas.htm	Engenharia de Computação	C		
IMES – Centro Universitário Municipal de São Caetano do Sul http://www.imes.com.br/	Bacharelado em Ciência da Computação	C		*
Pontifícia Universidade Católica de São Paulo http://www.pucsp.br/	Bacharelado em Ciência da Computação	Pascal	C, C++	*
PUC-Rio http://www.puc-rio.br/depto/ccg	Engenharia de Computação/Bach. em Ciência da Computação	C Lisp/Scheme/ C	C	
PUC-RS http://www.pucrs.br/uni/poa/info/index.htm	Bach. em Ciência da Computação/ Sistemas de Informação/Análise de Sistemas	C/Pascal	C++, Java	

UEPG – Univ. Estadual de Ponta Grossa http://www.uepg.br/	Bacharelado em Informática/Engenharia de Computação	C		
UFBA – Universidade Federal da Bahia http://www.ufba.br/	Bacharelado em Ciência da Computação	C, C++		*
UFPB – Universidade Federal da Paraíba http://www.ufpb.br/	Bacharelado em Ciência da Computação	Pascal		*
UFPE – Univ. Federal de Pernambuco http://www.cin.ufpe.br/~graduacao	Bach. em Ciência da Computação	Java	C	
UFPel – Univ. Federal de Pelotas http://www.ufpel.tche.br	Bach. em Ciência da Computação	Pascal	C e Java	
UFPR http://www.inf.ufpr.br/info/	Bach. em Ciência da Computação	Java		
UFRGS – Univ. Federal do Rio Grande do Sul http://www.inf.ufrgs.br	Engenharia de Computação/ Bach. em Ciência da Computação	Pascal		
UFRJ http://www.dcc.ufrj.br/informatica/home.html	Bach. em Informática	C		
UFSC http://www.ufsc.br/	Bach. em Ciência da Computação/ Sistemas de Informação	Pascal	Java	
UFSCar http://www.dc.ufscar.br/grad/index.html	Engenharia de Computação	Pascal		
UFU (Uberlândia) http://www.prograd.ufu.br/cursos/principal.htm	Bach. em Ciência da Computação	Haskell	Sixtus Prolog, C e Java	
UFV – Univ. Federal de Viçosa http://www.dpi.ufv.br/Graduacao/	Bach. em Ciência da Computação	C++		
ULBRA – Universidade Luterana do Brasil http://www.ulbra.br/	Bacharelado em Ciência da Computação	Pascal	Delphi	*
Unicamp – Univ. Estadual de Campinas http://www.ic.unicamp.br/cg/	Engenharia de Computação/Bach. em Ciência da Computação	Pascal	C	
Unicamp – Univ. Estadual de Campinas (CESET) http://www.ceset.unicamp.br/graduacao/ti/index.htm	Curso Superior de Tecnologia em Informática	Pascal, Delphi, C (no 1º ano)	C++, Java, SQL e VB	
UniCarioca http://www.carioca.br/	Bacharelado em Ciência da Computação	C	Delphi	*
UNIFRA – Centro Universitário Franciscano http://www.unifra.br	Bach. em Ciência da Computação/Sistemas de Informação	C/Pascal	Delphi e Java	
Unimar – Universidade de Marília http://www.unimar.br/	Bacharelado em Ciência da Computação	Pascal		*
UNIMEP – Univ. Metodista de Piracicaba http://www.unimep.com.br/	Bach. em Ciência da Computação	C		
Unioeste – Univ. Estadual do Oeste do Paraná http://www.unioeste.br/prg/	Bach. em Ciência da Computação	Pascal	C Ansi, C++ e Java	
UNIP - Universidade Paulista http://www.unip.br/	Bacharelado em Ciência da Computação	Pascal		*
UNIRONDON - Faculdades Integradas Cândido Rondon http://www.unirondon.br	Curso Superior de Tecnologia em Processamento de Dados Bacharelado em Ciência da Computação	Pascal	Delphi, SQL e Java C, C++, SQL e Java	*
UNISINOS http://www.inf.unisinos.br/cursos/graduacao	Curso de Informática – Hab. de Software Básico e Análise de Sistemas	Pascal		
Univ. Federal de Minas Gerais http://www.ufmg.br/cursos/	Bach. em Ciência da Computação	Java		
Univ. Federal de Sergipe http://www.ufs.br/	Bach. em Ciência da Computação	Pascal (O.O.)	Java	

Universidade Anhembi Morumbi http://www.anhembi.br/	Bacharelado em Ciência da Computação	Pascal		*
Universidade Católica de Petrópolis http://www.ucp.br/	Bacharelado em Ciência da Computação	Pascal		*
Universidade Católica Dom Bosco http://www.ucdb.br/cursos/cursos.htm	Engenharia de Computação	C		
Universidade de Brasília http://www.unb.br/	Bacharelado em Ciência da Computação	Pascal		*
Universidade do Vale do Paraíba http://www.univap.br/	Bacharelado em Ciência da Computação	Pascal		*
Universidade Estadual do Vale do Acaraú http://www.uvanet.br/	Bacharelado em Ciência da Computação	Pascal	C	*
Universidade Federal de Juiz de Fora http://www.ufjf.br/	Bacharelado em Ciência da Computação	Pascal		*
Universidade Federal de Lavras http://www.ufla.br/	Bacharelado em Ciência da Computação	C		*
Universidade Federal de Santa Maria http://www.ufsm.br/	Bacharelado em Ciência da Computação	Pascal	C	*
Universidade Federal de Sergipe http://www.ufs.br/	Bacharelado em Ciência da Computação	Pascal	C, C++, Java e Eiffel	*
Universidade Federal do Ceará http://www.ufc.br/	Bacharelado em Ciência da Computação	Pascal	C e Java	*
Universidade Federal do Mato Grosso do Sul http://www.ufms.br/	Bacharelado em Ciência da Computação	Pascal	C	*
Universidade Regional de Blumenau http://www.furb.br/	Bacharelado em Ciência da Computação, Sistemas de Informação	Pascal (OO)		*
Universidade Tuiuti do Paraná http://www.utp.br/	Bacharelado em Ciência da Computação, Sistemas de Informação	C		*
UNOESC - São Miguel do Oeste http://www.unoescsmo.edu.br/	Bach. em Sistemas de Informação/Tecnólogo em Informática	Pascal		
UNOESTE – Universidade do Oeste Paulista http://www.unoeste.br	Bach. em Ciência da Computação/ Sistemas de Informação	Pascal	C, C++, Delphi, Java e ASP	
URI – Campus de Erechim http://www.uricer.edu.br/	Bach. em Ciência da Computação	C	C++, Java	
USP – Universidade de São Paulo http://www.usp.br/	Bacharelado em Ciência da Computação	Pascal	C	*

* informações obtidas em pesquisa aos *sites* das universidades

Anexo 2 Plano de Ensino de Java da UFPE

Este plano de ensino, de disciplina introdutória de programação que utiliza a linguagem Java para implementação dos algoritmos, foi extraído do curso de Bacharelado em Ciência da Computação da Universidade Federal de Pernambuco, na disciplina de Introdução à Programação [UNP 2002]. Ele descreve a forma como os tópicos são abordados durante o semestre letivo. O plano de ensino é bastante ambicioso e abrangente, mas, para cumprir todo o conteúdo programático, a grade curricular dedica 8 horas-aula semanais a esta disciplina. A seguir é mostrado o plano de ensino completo na tabela A2.1.

TABELA A2.1 - Plano de Ensino de Java da UFPE

Disciplina: Introdução à Programação

Ementa:

- Conceitos básicos de linguagens de programação
- Conceitos básicos de programação orientada a objetos e qualidade de software
- Estruturas de controle e estruturas de dados compostas
- Ambiente de programação
- Conceitos básicos de programação imperativa
- Padrões e arquiteturas de software convencionais
- Estilo de programação
- Projeto de implementação

Conteúdos Programáticos:

- Tipos de dados elementares, variáveis e constantes, tipos enumerados, expressões
- Apresentação, programa, sistema, linguagem de programação, conceitos básicos de OO
- Conceitos e fatores básicos de Qualidade de Software, conceitos básicos de arquitetura em camadas
- Atributos, métodos e passagem de parâmetros, construtores, classes, objetos, referências, *information hiding, aliasing, overloading*
- Estruturas de controle e comandos
- Matrizes e *strings*, pacotes, exceções, processos leves e sincronização
- Herança e subtipos
    - Herança, polimorfismo de subtipo, classes abstratas e interfaces
- Bibliotecas de classes
    - Eventos e interfaces gráficas
    - Arquivos, objetos remotos
- Ambiente de programação visual
- Atributos e métodos estáticos, variáveis globais e procedimentos, diferenças entre linguagens imperativas e linguagens OO
- Registros

- Ponteiros e gerenciamento de memória
- Métodos nativos e integração de linguagem OO com linguagem imperativa
- Realização do projeto
- Apresentação de projetos
- Avaliação

Co-Requisito: Introdução a Computação

Carga Horária Semanal: 4 horas/aulas Teóricas e 4 horas/aulas Práticas

---

Nº de Créditos: 06

Bibliografia

[BAR 2000] BARNES, D. J. **Object-Oriented Programming with Java**. Upper Saddle River, NJ: Prentice Hall, 2000.

[BAE 2002] BARROS, E. **Current Positions, Courses and Projects**. Disponível em: <http://www.cin.ufpe.br/~ensb/>. Acesso em: abr. 2002.

[BEA 99] BEANSHELL: Lightweight Scripting for Java. Disponível em: <http://www.beanshell.org/>. Acesso em: nov. 1999.

[BEC 97] BERG, C. How Do I Browse and Dynamically Invoke Remote Objects? **Dr. Dobb's Journal**, New York, p. 121-123, Dec. 1997.

[BER 2000a] BERTAGNOLLI, S. C. **Ambiente Visual para o Desenvolvimento de Aplicações Java Reflexivas.** 2000. 92f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[BER 2000b] BERTAGNOLLI, S. C.; LISBOA, M. L. B. Ambiente para automatizar o processo de desenvolvimento de software reflexivo. In: JORNADAS IBEROAMERICANAS DE INGENIERIA DE REQUISITOS Y AMBIENTES DE SOFTWARE, IDEAS, 2000, México. **Memórias**. Cuernavaca: Centro Nacional de Investigación y Desarollo Tecnológico, 2000.

[BER 2002a] BERTAGNOLLI, S. C.; LISBOA, M. L. B.; PEREGO C. A.; BRUGNARA, T. Reflexão Computacional como Mecanismo de Simplificação Sintática. Artigo submetido ao Simpósio Brasileiro de Linguagens de Programação, SBLP, 2002, Rio de Janeiro.

[BER 2002b] BERTAGNOLLI, S. C.; LISBOA, M. L. B.; PEREGO, C. A. IDECoRe: Reutilizando Componentes e Ferramentas. Artigo submetido ao Simpósio Brasileiro de Engenharia de Software, SBES, 2002, Gramado.

[BIS 97] BISHOP, J. **Java Gently**. Essex, England: Addison-Wesley Longman, 1997.

[BLU 2001] BLUEJ: The Interactive Java Environment. Disponível em: <http://www.bluej.org>. Acesso em: 21 ago. 2001.

[BOR 2001] BORLAND. **Borland Turbo Pascal 7.0**. Disponível em: <http://www.borland.com/pascal/>. Acesso em: 21 ago. 2001.

[BRA 2002] BRAUN, D. Linguagens vão à universidade. **Revista ComputerWorld**, [S.l.], 08 maio 2002.

[BRS 99] BRODY, S. **Java kept alive by big money**. 1999. Disponível em: <http://www.cnn.com/TECH/computing/9904/09/java.idg/>. Acesso: 2001.

[BRO 2002] BROEGLIN, D. **Educational Information System**. Disponível em: <http://www.ofset.org/freeduc/>. Acesso em: jan. 2002.

[BRU 2002a] BRUGNARA, T. **Uma Biblioteca de Components de Entrada e Saída Java para Ambiente de Ensino-Aprendizagem**. 2002. 57f. Projeto de Diplomação (Bacharelado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[BRU 2002b] BRUGNARA, T.; BERTAGNOLLI, S. C.; LISBOA, M. L. B.; PEREGO, C. A. JEduc: uma ferramenta livre para auxiliar o ensino da linguagem de programação Java. In: WORKSHOP SOFTWARE LIVRE, WSL, 2002, Porto Alegre. **Anais…** Porto Alegre: Sociedade Brasileira de Computação, 2002.

[CAS 2001] CASTILHO, J. A busca por especialistas em Java. **Revista ComputerWorld**, [S.l.], 05 set. 2001.

[CLA 98] CLARK, D.; MACNISH, C.; ROYLE, G. F. Java as a teaching language opportunities, pitfalls and solutions. In: AUSTRALASIAN CONFERENCE ON COMPUTER SCIENCE EDUCATION, 3., 1998, Brisbane, Australia. **Proceedings**… [S.l.:s.n.], 1998.

[DEI 2001] DEITEL, H. M.; DEITEL, P.J. **Java:** como programar. 3. ed. Porto Alegre: Bookman, 2001.

[DOR 99] DORIAN, L. A. C. **Scripting Tools and Java**. Disponível em: <http://www.devx.com/upload/free/features/javapro/1998/12dec98/ld1298/ld1298.asp>. Acesso em: nov. 1999.

[FOO 98] FOOTE, B. **Objects, Reflection and Open Languages**. Disponível em: <http://www.laputan.org/foote/papers>. Acesso em: dez.1998.

[FOR 2000] FORBELLONE, A. L. V.; EBERSPÄCHER, H. F. **Lógica de Programação:** a Construção de Algoritmos e Estrutura de Dados. 2. ed. São Paulo: Makron Books, 2000.

[FRE 2001] FREE SOFTWARE FOUNDATION. **GNU e Educação**. Disponível em: <http://www.gnu.org/education/education.pt.html>. Acesso em: set. 2001.

[FRE 2002] FREE SOFTWARE FOUNDATION. **GNU Education**. Disponível em: <http://www.gnu.org/gnulist/production/education.html>. Acesso em: jan. 2002.

[GOT 94] GOTTFRIED, B. S. **Programação em Pascal**. 2. ed. Lisboa, Portugal: McGraw-Hill, 1994.

[GRU 2000] GRUNDY, J. C. Multi-perspective specification, design and implementation of components using aspects. **International Journal of Software Engineering and Knowledge Engineering**, [S.l.], v. 10, n. 6,Dec. 2000.

[GUI 85] GUIMARÃES, A. M.; LAGES, N. A. C. **Algoritmos e Estruturas de Dados**. Rio de Janeiro: L.T.C., 1985.

[HON 2001] HONG, J. **The Use of Java as an Introductory Programming Language**. Disponível em: <http://www.acm.org/crossroads/xrds4-4/introjava.html>. Acesso em: 23 jun. 2001.

[HOR 2000] HORSTMANN, C. **Computing Concepts with Java 2 Essentials**. 2. ed. New York: John Wiley & Sons, 2000.

[HOR 2001] HORSTMANN, C. S.; CORNELL, G. **Core Java 2:** fundamentos. São Paulo: Makron Books, 2001. v.1.

[HOW 98] HOWARD, C. Reflecting on Beans. **Dr. Dobb's Journal**, New York, p. 36, Jan. 1998.

[JED 2002] JEDUC: Informações, Links e Download. Disponível em: <http://www.inf.ufrgs.br/~silviacb/JEduc>. Acesso em: maio 2002.

[JRE 2002] JREFLEX: Informações, Links e Download. Disponível em: <http://www.inf.ufrgs.br/~silviacb/JreflexSimplificado>. Acesso em: jan. 2002.

[KIN 97] KING, K. N. The Case for Java as a First Language. In: ACM SOUTHEAST CONFERENCE, 1997, Murfreesboro, Tenn. Disponível em: <http://www.gsu.edu/~matknk/java/reg97.htm>. Acesso em: 17 jun. 2002.

[KIR 97] KIRBY, G.; MORRISON, R. OCB: An Object/Class Browser for Java. In: INTERNATIONAL WORKSHOP ON PERSISTENCE AND JAVA PJW, 2., 1997, HalfMoonBay, California. **Proceedings...** [S.l.:s.n.], 1997.

[KOF 99] KOFFMAN, E.; WOLZ, U. CS1 Using Java Language Features Gently. **SIGCSE Bulletin**, New York, v. 31, n. 3, p. 40-43, Sept. 1999. Trabalho apresentado no ACM SIGCSE Technical Symposium on Computer Science Education, 1999.

[KOF 2002] KOFFMAN, E. Input/output for a CS1 course in Java. In: INFORMATICS CURRICULA TEACHING METHODS AND BEST PRACTICE, ICTEM, 2002. **Proceedings...** Florianópolis, Santa Catarina: [s.n.], 2002.

[KOL 99] KÖLLING, M.; ROSENBERG, J. Tools and Techniques for Teaching Objects First in a Java Course. **SIGCSE Bulletin**, New York, v. 30, p. 368, Mar. 1999. Trabalho apresentado no ACM SIGCSE Technical Symposium on Computer Science Education, 1999.

[KOL 2001] KÖLLING, M.; ROSENBERG, J. Guidelines for teaching object orientation with Java. In: ANNUAL CONFERENCE ON INNOVATION AND TECHNOLOGY IN COMPUTER SCIENCE EDUCATION, 6., 2001. **Proceedings...** [S.l.:s.n.], 2001.

[KUH 2001] KUHN, B. M. **Richard Stallman Inaugura a Fundação para o Software Livre-Índia**. Disponível em: <http://www.gnu.org/press/2001-07-20-FSF-India.pt.html>. Acesso em: jul. 2001.

[LEA 96] LEA, D. **Some questions and answers about using Java in Computer Science Curricula**. 1996. Disponível em: <http://gee.cs.oswego.edu/dl/html/javaInCS.html>. Acesso em: 26 abr. 2001.

[LEW 2000] LEWIS, J.; LOFTUS, W. **Java Software Solutions:** Foundations of Program Design. 2. ed. [S.l.]: Adisson-Wesley, 2000.

[LIS 97] LISBOA, M. L. B. **Tutorial:** arquiteturas de meta-nível. Fortaleza: UFC, 1997. 35p. Tutorial apresentado no Simpósio Brasileiro de Engenharia de Software, 11., 1997.

[LIS 2002] LISBOA, M. L. B.; PEREGO, C. A.; BERTAGNOLLI, S. C. Introductory programming: moving from Pascal to Java. In: INFORMATICS CURRICULA TEACHING METHODS AND BEST PRACTICE, ICTEM, 2002. **Proceedings...** Florianópolis, Santa Catarina: [s.n.], 2002.

[MAE 87] MAES, P. Concepts and experiments in computational reflection. **ACM SIGPLAN Noticies**, New York, v. 22, n.12, p.147-155, 1987.

[MAN 2000] MANZANO, J. A. N. G.; YAMATUMI, W. Y. **Programando em Turbo Pascal 7.0**. 6. ed. São Paulo: Ed. Érica, 2000.

[MAA 98] MARIANI, A. C. **O Mundo dos Atores:** uma perspectiva de introdução à programação orientada a objetos. Disponível em: <http://www.inf.ufsc.br/poo/atores/sbie98/sbie98-atores.html>. Acesso em: set. 2001.

[MAR 94] MARTINS, J. P. **Introdução à Programação usando o Pascal**. Lisboa, Portugal: McGraw-Hill, 1994.

[MEN 2001] MENDES, A. J. N. **Software educativo para apoio à aprendizagem de programação**. Disponível em: <http://www.c5.cl/ieinvestiga/actas/tise01/pags/charlas/charla_mendes.htm>. Acesso em: dez. 2001.

[NIX 2001] NIXON, P. **A First Course in Computer Science – Object Oriented Programming Using Java**. Disponível em: <http://www.cs.rit.edu/ñcs/Uppsala97/>. Acesso em: 18 jul. 2001.

[OSS 2000] OSSHER, X. O. Software Engineering Tools and Environments: Roadmap. In: MAGNENAT-THALMANN, Nadia; THALMANN, Daniel (Ed.). **New Trends in Animation and Visualization**. England: John Wiley & Sons, 2000.

[OSW 2001] OSWEGO, State University of New York at. **Computer Science Course CSC 212:** Principles of Computing. Disponível em: <http://www.cs.oswego.edu/emma/outlines/csc/csc212.html>. Acesso em: 26 abr. 2001.

[PAU 99] PAULA, V. C. **MiCrOO:** um micro-mundo para o ensino-aprendizagem de programação orientada a objetos.1999. Disponível em: <http://www.dcc.ufmg.br/pos/html/spg99/anais/valeria/valeria.html>. Acesso em: ago. 2001.

[PAT 95] PATTIS, R. E. Karel the Robot. 2. ed. [S. l.]: John Wiley & Sons, 1995.

[PER 2001a] PEREGO, C. A. **Adequando Java para Aprendizagem de Programação**. 2001. 55f. Trabalho Individual I (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[PER 2001b] PEREGO, C. A.; LISBOA, M. L. B. **Linguagens de Programação Adotadas nos Cursos de Computação**. Disponível em: <http://apec.unoeste.br/~cassia/pesquisa.html>. Acesso em: nov. 2001.

[PER 2002a] PEREGO, C. A.; LISBOA, M. L. B.; BERTAGNOLLI, S. C. A Migração de Pascal para Java: problemas e propostas de solução. In: WORKSHOP SOBRE EDUCAÇÃO EM COMPUTAÇÃO, WEI, 2002, Florianópolis. **Convergências tecnológicas:** redesenhando as fronteiras da Ciência e da Educação. Florianópolis: SBC, 2002.

[PER 2002b] PEREGO, C. A.; BERTAGNOLLI, S. C.; LISBOA, M. L. B. JEduc: Um ambiente dedicado ao ensino de Java. In: CONGRESO IBEROAMERICANO DE EDUCACIÓN SUPERIOR EN COMPUTACIÓN, CIESC, 2002, 25 a 27 de novembro de 2002, Montevidéo, Uruguai. A ser apresentado.

[PRE 95] PRESSMAN, R. S. **Engenharia de Software**. São Paulo: Makron Books, 1995.

[PRO 2002] PROULX, V. K.; RAAB, J.; RASALA, R. Objects From the Beginning – With GUIs. In: INNOVATION AND TECHNOLOGY IN COMPUTER SCIENCE EDUCATION. ITiCSE, 2002, Aarhus, Denmark. **Proceedings…** [S.l.:s.n.], 2002.

[REA 2001] READY: Ready to Program. Disponível em: <http://www.holtsoft.com/ready/home.html>. Acesso em: jul. 2001.

[REJ 2001] REALJ: the IDE for Java. Disponível em: <http://www.realj.com>. Acesso em: 21 ago. 2001.

[REG 2000] REGES, S. Conservatively Radical Java in CS1. **SIGCSE Bulletin**, New York, v. 32, n. 1, p. 85-89, Mar. 2000. Trabalho apresentado no ACM SIGCSE Technical Symposium on Computer Science Education, 2000.

[REI 2001] REID, D. **The Return of the Reid List**. Disponível em: <http://www.csee.wvu.edu/~vanscoy/REID21.HTM>. Acesso em: 06 fev. 2001.

[REI 2002] REID, D. **The Reid First Course Language List - Version 23**. Disponível em: <http://www.csee.wvu.edu/~vanscoy/REID23.HTM>. Acesso em: 1 fev. 2002.

[RIC 2001] RICARTE, I. L. M. **Curso de Java**. Disponível em: <http://www.dca.fee.unicamp.br/courses/PooJava/io/>. Acesso em: 23 jul. 2001.

[ROB 2001] ROBERTS, E. An Overview of MiniJava. **SIGCSE Bulletin**, New York, v. 33, n. 1, p. 1-5, Mar. 2001.

[ROZ 2001] ROZANSKI, E. **Java as the First Programming Language for Animators and Multimedia Students**. Disponível em: <http://www.cs.rit.edu/ñcs/Uppsala97/>. Acesso em: 18 jul. 2001.

[SAG 98] SAGAR, A.; HILERIO, I. Reflection & Instrospection: Objects Exposed. **Java Developers Journal**, [S.l.], v. 3, n. 5, p. 24-33, May 1998.

[SAV 99] SAVITCH, W. **Java**: An Introduction to Computer Science and Programming. Upper Saddle River, NJ: Prentice Hall, 1999.

[SEB 2000] SEBESTA, R. W. **Conceitos de linguagens de programação**. 4. ed. Porto Alegre: Bookman, 2000.

[SKI 99] SKIJ. Disponível em: <http://www.alphaworks.ibm.com/formula/skij>. Acesso em: nov. 1999.

[SOU 2001] SOUZA, G. H.; PINTO, M. R.; BRITO, M. A. S. Uma contribuição à disciplina de introdução à programação empregando Java. **Caderno do IME:** Série Informática, Rio de Janeiro, v. 10, 2001.

[STE 2001] STEWART, B. **Java as a Teaching Language**. Disponível em: <http://java.oreilly.com/news/teachjava_0101.html>. Acesso em: 14 jul. 2001.

[SUN 2002a] SUN MICROSYSTEMS. **Java Tutorial**. Disponível em: <http://java.sun.com/docs/books/tutorial/>. Acesso em: jan. 2002.

[SUN 2002b] SUN MICROSYSTEMS. **Java Language Specification**. Disponível em: <http://java.sun.com/docs/books/jls/>. Acesso em: fev. 2002.

[SUN 2002c] SUN MICROSYSTEMS. **JavaTM 2 Platform, Standard Edition, v 1.4.0 API Specification**. Disponível em: <http://java.sun.com/j2se/1.4/docs/api/index.html>. Acesso em: fev. 2002.

[SZI 98] SZIPERSKI, C. **Component Software Beyond Object-Oriented Programming**. New York: Addison-Wesley, 1998.

[TCL 99] TCL: Java Integration. Disponível em: <http://tcl.activestate.com/software/java/>. Acesso em: nov. 1999.

[UNI 2001] UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL. **Graduação em Ciência da Computação**: algoritmos e programação. Disponível em: <http://nina.inf.ufrgs.br/INF202pr.html>. Acesso em: 05 maio 2001.

[UNO 2001] UNIVERSIDADE DO OESTE PAULISTA, FACULDADE DE INFORMÁTICA DE PRESIDENTE PRUDENTE. **Caderno de Planos de Ensino do 1º Semestre de 2001.** Presidente Prudente, 2001.

[UNP 2002] UNIVERSIDADE FEDERAL DE PERNAMBUCO. **Introdução à Programação**. Cin. Ufpe. Disponível em: <http:// www.cin.ufpe.br/~graduacao/reforma/p_introducao_programacao.html>. Acesso em: 07 out. 2002.

[USP 2001] USP. DEPARTAMENTO DE FÍSICA E INFORMÁTICA. **Introdução à prática de programação**. Disponível em: <http://www.ffclrp.usp.br/site/home/dic-bio/5910310.htm>. Acesso em: 18 jul. 2001.

[VAN 99] VANHELSUWÉ, V. **The Best Java Books For Beginners**. Disponível em: < http://www.cnn.com/TECH/computing/9901/22/javabks.idg/>. 22 jan. 1999.