Resumo por Kaléu Caminha

Diretrizes para o Projeto e Ensino Efetivo de Cursos de Programação Orientada a Objetos

Stelios Xinogalos

Este artigo apresenta uma revisão das principais pesquisas em ensino de Orientação a Objetos na medida em que demonstra como diversos resultados destas pesquisas foram utilizadas em quatro anos de aplicação de um curso de Orientação a Objetos (OO) com Java.

O artigo torna clara a complementariedade das diferentes abordagens de ensino de OO, de modo que muitas das diversas abordagens para ensino de OO são importantes, porém, nenhuma solução independente atende a todas as dificuldades típicas dos estudantes.

1 Abordagens de Ensino

A estratégia “Objetos Primeiro”, na qual o primeiro contato que os estudantes tem com programação é a instanciação de objetos e chamada de métodos destes objetos é amplamente aceita pelos instrutores desta área.

Esta estratégia gera dificuldades extras ao ensino de OO, já que a maioria dos ambientes e linguagens de programação não tem essa preocupação.

Para suportar esta estratégia e também para trabalhar outras dificuldades dos diversas abordagens foram desenvolvidas. As mais importantes são apresentadas abaixo:

- **Exemplos projetados para evitar as principais dificuldades** (Holland et al, 1997): Nesta técnica, os exemplos e atividades são projetados de modo a evitar que dificuldades comuns já observadas em cursos anteriores se repitam. Um exemplo é utilizar projetos onde são necessárias várias instâncias de uma classe para evitar que se entenda “classe” e “objeto” como a mesma coisa;
- **Uso de Interfaces Gráficas de Usuário** (Proulx et al, 2002): Proulx et al defendem que os conceitos de OO devem ser ensinados através do uso softwares com Interfaces Gráficas de Usuário (GUI) previamente desenvolvidas. Nestes exemplos, os alunos manipulam pequenos softwares com interfaces gráficas e os conceitos de OO vão sendo apresentados através da execução e manipulação do programa, onde, por exemplo, cada botão equivale a um método de uma classe. Eles desenvolveram um modelo de um curso com 4 laboratórios onde os estudantes:
  - no primeiro, estudam o comportamento dos objetos sem nenhuma referência ao código fonte, apenas manipulando a GUI;
  - no segundo laboratório, observam a mudança no estado dos objetos e a resposta à ações invocadas por chamadas de métodos;
  - no terceiro, exploram parte do código do projeto e fazem pequenas modificações;
  - no quarto, estendem uma classe existente.
- **Uso de estudos de caso** (Nevison e Wells, 2003): Nevison e Wells acreditam que o uso de estudos de caso permite trabalhar diversos níveis de complexidade,

ressaltam contudo, que a escolha de quais estudos de caso serão utilizados deve ser feita co cuidado;

- **Uso de ambientes educacionais de programação**: O uso de ambientes profissionais de desenvolvimento de software são complexos e não recomendados para estudantes iniciantes. Neste contexto diversas iniciativas surgiram com a intenção de criar ambientes de programação que favorecessem o ensino.

  A ferramenta com maior destaque até o momento é o BlueJ, desenvolvido por Kolling et al (2003) que permite o uso da estratégia "objetos primeiro" e do ensino baseado em projetos.

  Nele os estudantes iniciam por criar objetos e executar métodos de classes já existentes de um determinado projeto, sem conhecer o código fonte, apenas pela interface gráfica do programa, depois, visualizam os códigos e fazem pequenas modificações em métodos existentes, adicionam novos métodos e por fim, passam a construir suas próprias classes;
- **Programação de micro-mundos**: Ambientes de programação de micro-mundos são sofwares que permitem que um estudante interaja com objetos - geralmente caracterizados como objetos do mundo real como um animal ou um robô -de forma a compreender os principais conceitos da programação OO.

  Por exemplo, um estudante pode ser estimulado a criar instâncias de uma classe Robô previamente elaborada e executar métodos destas instâncias como *moverParaDireita* ou *moverParaEsquerda*.

  Muitos destes sofwares utilizam uma linguagem de programação educacional.

  Um dos ambientes que permitem a programação de micro-mundos é o objectKarel (Buck & Stucki, 2000), baseado na metáfora de robôs que realizam tarefas em um mundo formado por ruas horizontais e verticais, obstáculos e objetos que emitem o som de um *beep*.
- **Abordagem "modelo primeiro" e uso de padrões de código**: (Bennedsen & Caspersen, 2004): Nesta forma de trabalho, o modelo do projeto vem em primeiro lugar, servindo como base para trabalhar os conceitos de Orientação a Objetos. Após o modelo pronto, padrões de codificação são utilizados para transformar os diagramas UML construídos em código.

2 Avaliação "de longa duração" de um curso Orientação a Objetos

O curso estudado neste artigo é oferecido como uma disciplina de "Projeto e Programação Orientada a Objetos" com duração de 13 semanas no terceiro semestre de uma formação no "Technology Management Departament". Cada semana possui 2 horas/aula de aula teórica e 2 horas/aula de laboratório. Destas 13 semanas, duas são dedicadas para atividades de avaliação, restando 11 semanas para o trabalho direto de conteúdos.

2.1 Avaliação do curso utilizando um ambiente educacional

A primeira aplicação do curso foi realizada utilizando o BlueJ como ferramenta de ensino acompanhada do livro-texto *"Objetos Primeiro com Java: Uma introdução prática usando o BlueJ"*.

As 11 aulas foram trabalhas na seguinte sequência: 1

1. Objetos e Classes;
2. Entendendo definições de classes;
3. Interação entre objetos;
4. Coleções de objetos flexíveis e não flexíveis;
5. Uso de Bibliotecas de classe como HashMap;
6. Teste e “Debugging”;
7. Projeto de Classes;
8. Melhorando a estrutura com Herança;
9. Polimorfismo e Sobrecarga;
10. Classes Abstratas e Interfaces;
11. Métodos estáticos – o método main.

Ao final do curso, verificou-se que as principais dificuldades relacionadas diretamente com Orientação a Objetos foram:

1. Construtores;
2. Instanciação de objetos;
3. métodos “set”;
4. métodos “get”;
5. Chamada de métodos;
6. Modificadores de Acesso;
7. Coleções de objetos;
8. Herança;
9. Métodos e classes abstratas e interfaces.

Foram verificadas três dificuldades relevantes especificamente relacionadas à determinadas características do BlueJ, são elas:

1. **Ênfase nas técnicas de visualização e manipulação direta do BlueJ**: São atribuídas a esta característica as dificuldades com instanciação de objetos e chamadas de métodos. Isso porque no BlueJ, a diferença entre a forma de manter objetos em variáveis e chamar métodos destas variáveis (Ilustração 1) é muito diferente da forma como a mesma ação é feita no BlueJ (Ilustração 2).

   Outra crítica é a que métodos que possuem retorno são chamados da mesma forma que métodos sem retorno. Esta questão causa confusão ao tentar passar para os estudantes que métodos com retorno devem ser sempre atribuídos a uma variável.

```
BallDemo bola1 = new BolaDemo();
bola1.bounce();
```

Ilustração 1: Instanciação de objeto e chamada de método em código

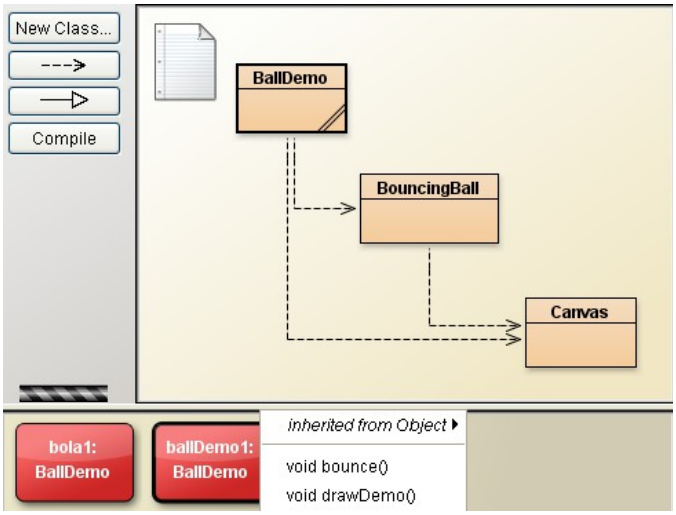

Ilustração 2: Objetos na bancada de objetos (em vermelho) e chamada de método de um objeto

2. **Ênfase em projetos existentes**: Em função da utilização de projetos com uma boa quantidade de código pronto, a dificuldade em iniciar projetos do zero se mostrou muito maior.

   Além disso, métodos get e set, bem como construtores, são geralmente fornecidos pelos projetos e os alunos raramente os implementam.

   A maior dificuldade foi ainda com coleções de objetos, apesar dos estudantes entenderem o conceito, possuem muita dificuldade em compreender como manipular as listas.

3. **Ensino posterior do método *main***: O ensino apenas no final do curso do método *main* exige que os alunos utilizem apenas o BlueJ para testar código, potencializando as dificuldades comentadas no item 1.

2.2 Reformulação do curso com uso de uma combinação de um ambiente educacional com um ambiente profissional de desenvolvimento

Baseado nas observações da versão anterior do curso este foi redesenhado e as seguintes mudanças foram feitas:

- O método main passou a ser apresentado na 4º aula e os estudantes passam a realizar instanciação de objetos e chamada de métodos diretamente a partir do método main.
- Pouco após a 4ª aula, o ambiente de programação JCreator é apresentado aos alunos e eles próprios podem escolher qual ambiente melhor atende às suas necessidades;
- Projetos do zero são desenvolvidos muito mais cedo;
- A aula sobre *"debugging"* e testes foi substituída por uma segunda aula sobre coleções de objetos em função desta ter sido a principal dificuldade percebida;

Estas modificações sanaram as principais dificuldades atribuídas às características do BlueJ e o uso de projetos desenvolvidos do zero ajudou a diminuir as dificuldades com os métodos get e set, construtores e modificadores de acesso das variáveis de instância.

O curso seguiu neste formato por dois anos, contando apenas com pequenas modificações.

2.3 Exposição gradual dos conceitos de POO com micro-mundos, ambientes educacionais e ambientes profissionais de desenvolvimento.

Após a observação de 2 anos do curso, verificou-se que as maiores dificuldades dos alunos era a compreensão de conceitos mais avançados de Programação Orientada a Objetos como herança, sobrescrita e polimorfismo.

Estimulados por um estudo sobre o potencial da programação de micro-mundos para o ensino de conceitos de Orientação a Objetos foi realizado um estudo comparando o desempenho de estudantes em um curso com BlueJ e outro com objectKarel.

O resultado demonstrou maior facilidade com os conceitos de OO no curso realizado com objectKarel.

A partir deste resultado o curso foi novamente remodelado para que as duas primeiras lições fossem ministradas através de exemplos com o objectKarel.

Além disto, outras alterações foram realizadas no curso:

1. Melhor compreensão da estrutura de ArrayLists através do uso de simples programas e de seus respectivos diagramas de objetos;
2. A técnica pedagógica “preenchendo nos brancos” foi utilizada para que os estudantes completassem trechos de código propositalmente incompletos ou com erros para trabalhar conceitos específicos e erros comuns no uso de ArrayLists;
3. Desenvolvimento de projetos que utilizam ArrayList do zero.

Com estas modificações foi verificado que houve um aumento da compreensão dos conceitos de OO conforme esperado. A maior dúvida era se haveria dificuldade em transmitir os exemplos mostrados no objectKarel para o Java. Em questionários aplicados em 2 versões do curso, foi obtido respectivamente 69 e 91 porcento de estudantes que responderam não terem tido nenhuma dificuldade extra em transportar os conceitos do objectKarel para o Java.

3. Diretrizes para o Projeto e Ensino de cursos de programação Orientada a Objetos

Ao fim de 4 anos de avaliações e modificações o curso passou a utilizar três ambientes: objectKarel, BlueJ e Jcreator. O material didático utilizado é baseado no material que acompanha o objectKarel e o livro-texto *“Objetos Primeiro com Java: Uma introdução prática usando o BlueJ”.* Ao invés de 11 lições, o curso passou a ter 12 aulas divididas na seguinte estrutura:

Aula	**Temas trabalhados**	**Ambiente**
1	*Objetos, classes, herança*: objeto, construção e inicialização de um objeto, mensagens/métodos, atributos e comportamentos de um objeto, herança, superclasse, subclasse, hierarquia de herança, diagrama de classe UML	*objectKarel*
2	*Herança em vários níveis, polimorfismo e sobrescrita*.	*objectKarel*
3	*Definição de Classe*: campos, construtores, métodos de	*BlueJ*

	acesso, métodos modificadores, estrutura de retorno, parâmetros, escopo de variáveis, estruturas condicionais.	
4	*Interação com objeto*: abstração, modularização, objetos criando objetos, múltiplos construtores, diagrama de classe, diagrama de objetos, tipos primitivos, tipos de objetos, chamadas de métodos internos e externos.	*BlueJ*
5	*Métodos estáticos*: métodos de instância vs métodos de classe, o método main, execução sem o BlueJ, byte code, Java Virtual Machine.	*BlueJ, JCreator*
6, 7	*Agrupando objetos em coleções*: Coleções de tamanho flexível (ArrayList), coleções de tamanho fixo (arrays), classes genéricas, iteradores, loops (while, for, for-each).	*BlueJ, JCreator*
8	*Usando bibliotecas de classe*: Classes padrão do Java, leitura de documentação, interface vs implementação de uma classe, explorando e usando classes (HashMap, Random, HashSet), modificadores de acesso (public, private), informações escondidas, variáveis de classe (estáticas).	*BlueJ, JCreator*
9	*Projeto de Classes*: acoplamento, coesão, encapsulamento, duplicação de código, projeto dirigido à responsabilidade, reuso de código.	*BlueJ, JCreator*
10	*Melhorando a estrutura com herança*: hierarquia de classes, superclasse, subclasse, construtor da superclasse, *"subtyping"*, substituição, *"autoboxing"*.	*BlueJ, JCreator*
11	*Polimorfismo, sobrescrita:* tipos dinâmicos e estáticos, busca dinâmica de método, super, acesso protegido.	*BlueJ, JCreator*
12	*Classes abstratas e Interface*.	*BlueJ, JCreator*

3.1 Uso de ambientes de programação

A escolha do ambiente de programação que será utilizado é difícil e importante. Ambientes profissionais são inadequados para trabalhar aspectos introdutórios, ambientes de micro-mundos são interessantes para trabalhar conceitos mas não uma linguagem específica, já ambientes educacionais são interessantes para transferir os conceitos para uma determinada linguagem, porém, sem a transição para uma IDE profissional os estudantes ficam condicionados às características específicas da ferramenta.

A dinâmica de 4 anos no curso apresentado mostrou que mais de uma ferramenta é necessária para que os diversos aspectos cognitivos possam ser trabalhados. Esta dinâmica gerou as seguintes diretrizes sobre o uso de ambientes de programação:

- **Use um ambiente de micro-mundo para uma introdução curta dos principais aspectos de programação orientada a objetos**: o ambiente escolhido deve suportar o ensino dos principais conceitos de OO sem preocupações com a sintaxe. O conceito de objeto deixa de ser abstrato pois agora os objetos podem ser manipulados, os estudantes também ganham confiança na sua capacidade de programação;
- **Use um ambiente educacional para apresentar os conceitos vistos no micro-mundo para a sua implementação em Java**: a transição para um ambiente educacional é mais suave do que para um ambiente profissional pois ambientes

específicos para o ensino eliminam a complexidade permitindo que os estuantes possam se dedicar a sintaxe dos conceitos fundamentais. É essencial que estes ambientes tenham ferramentas de visualização e manipulação direta de classes e objetos e permitam a invocação dinâmica de métodos. O tempo com o um ambiente educacional não deve se estender muito para evitar falsas percepções, principalmente sobre os aspectos dinâmicos da Orientação a Objetos.

- **Use um ambiente profissional para prepará-los para o futuro:** É inevitável que esta transição aconteça, a maior preocupação deve ser quando realizá-la. Neste curso foi decidido realizar esta transição na quinta lição dando ênfase para as características destes ambientes que poderiam ajudar os estudantes como 1) Visão das classes em árvore; 2) Navegação rápida dentro de uma classe; 3) auto-completar de código; 4) o destaque de sintaxe. Ao realizar esta transição os estudantes se sentem mais confiantes com o seu conhecimento.

3.2 Diretrizes para o Ensino das lições propostas

As lições são baseadas como já mencionado no livro-texto que acompanha o BlueJ. As principais diretrizes desta metodologia são:

- *"Objetos Primeiro"*: estudantes iniciam criando objetos e chamando métodos destes objetos desde a primeira aula;
- Abordagem Iterativa: Todos os conceitos chave são apresentados logo na primeira lição e revisitados ao longo do curso;
- Foco nos conceitos de OO e não nos detalhes de sintaxe do Java;
- As aulas são organizadas com foco nas tarefas base de Orientação a Objetos e não nos recursos d Java;
- Abordagem guiada a projetos.

Concomitantemente, é utilizada a abordagem "modelo primeiro" de Bennedsen e Caspersen (2004) que consiste em:

1. O problema é descrito de modo que exige a construção do modelo de um sistema;
2. Conceitos básicos do domínio do problema são apresentados;
3. Um modelo cognitivo do problema é construído, reconhecendo os objetos que realizam as atividades envolvidas no problema;
4. O modelo de domínio é apresentado com um diagrama de classes UML;
5. As classes são construídas, introduzindo os novos conceitos e estruturas de OO;
6. Padrões de código são utilizados para problemas similares.

Abaixo, seguem as diretrizes relacionadas a cada lição trabalhada. As lições utilizam projetos do livro-texto que acompanha o BlueJ e seus respectivos nomes são informados em parênteses.

3.2.1 Lição 1: Objetos, classes e herança no objectKarel

Na primeira lição, são apresentados os conceitos básicos de classe, objeto e método. Os estudantes iniciam criando um novo objeto da classe "Robo" com a ajuda da interface Gráfica (Ilustração 3), depois, devem utilizar os métodos do robô (como "mover", "virar para a esquerda", etc) para realizar tarefas específicas como completar um determinado percurso.

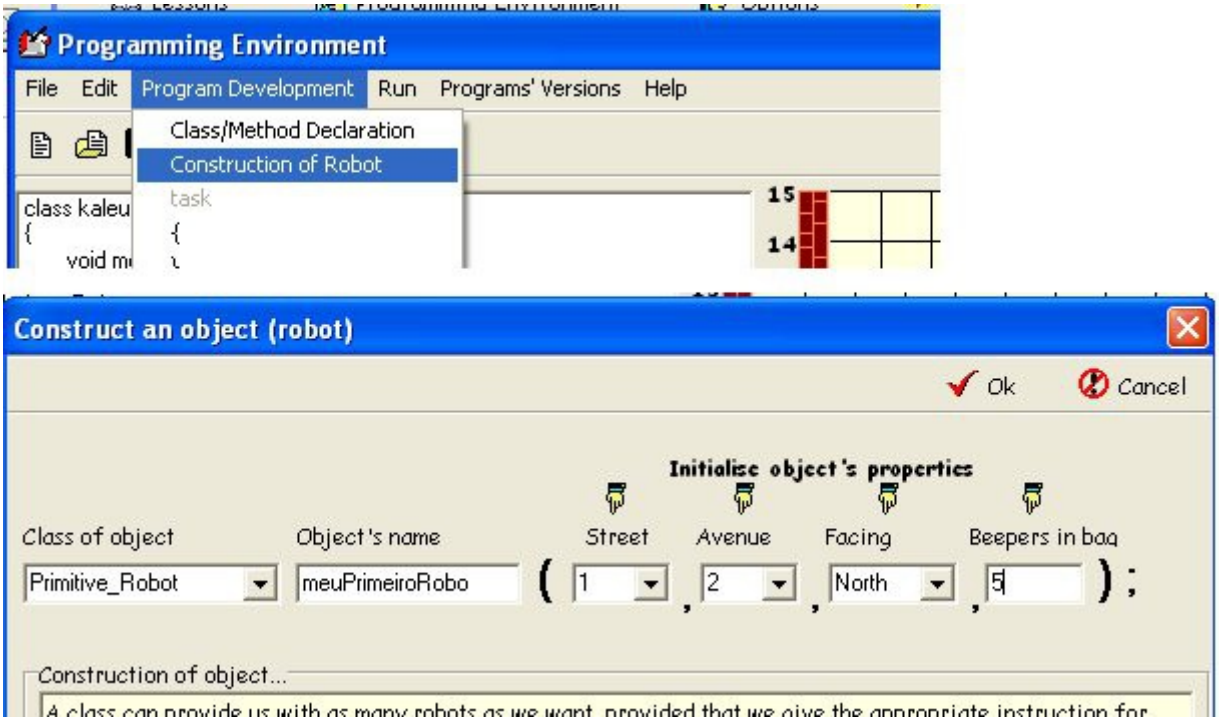

Ilustração 3: Instanciação de um objeto no objectKarel com a ajuda da interface gráfica

O programa vai sendo construído a partir de botões onde o estudante seleciona os métodos que irá executar, criando uma sequencia que no objectKarel é chamada de tarefa (Ilustração 4).

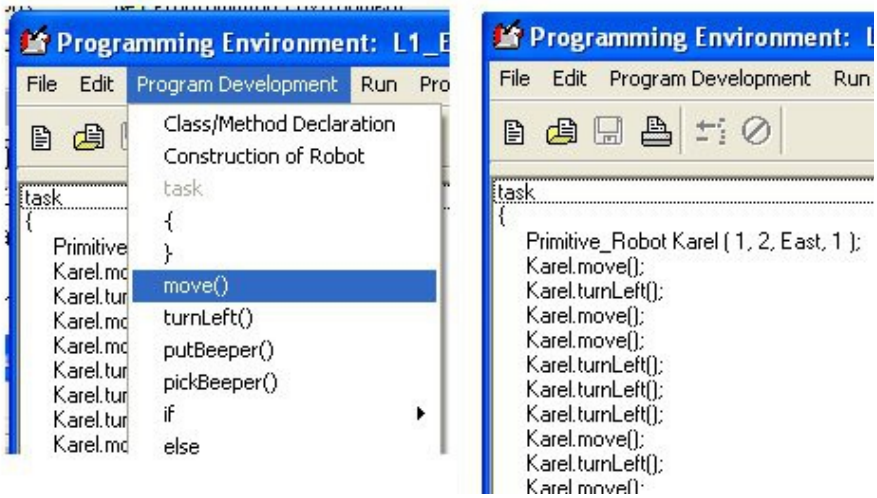

Ilustração 4: Criação de uma tarefa a partir dos métodos de um robô

Após, podem executar o programa e observar qual o comportamento do robô para cada método e o que isso significa na linguagem dos robôs (por exemplo, virar para a esquerda significa mudar a frente do robô em 90 graus negativos.), estas informações são fornecidas na forma de texto no rodapé do ambiente (Ilustração 5).

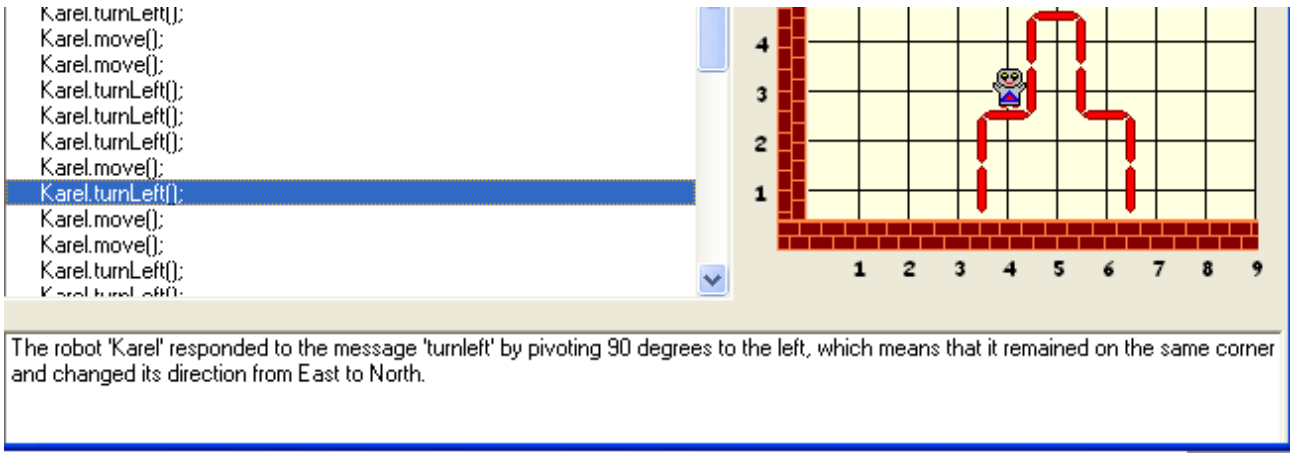

Ilustração 5: Execução de um programa. à direita a visualização do robô, à esquerda a lista de tarefas e abaixo a descrição do que ocorre no interior do método em execução.

Em uma segunda atividade, problemas mais complexos são apresentados de modo que é necessário utilizar herança para a criação de uma nova classe com métodos que irão facilitar a realização da tarefa.

O autor ressalta que é essencial durante o processo explicar que cada método muda o estado interno do objeto (mesmo que os alunos não possam ver as variáveis de instância). Também é importante utilizar exemplos que utilizem vários objetos da mesma classe para evitar que os estudantes considerem os dois conceitos como a mesma coisa.

3.2.2 Lição 2: Herança em vários níveis, polimorfismo e sobrescrita.

A didática se mantém a mesma e agora os problemas apresentados exigem a construção de uma hierarquia de classes. Também são utilizados exemplos que trabalhem a necessidade da sobrescrita de métodos como por exemplo, robôs mais velozes que no método “move”, andam mais rápido que os robôs utilizados nos exercícios anteriores.

O polimorfismo, também é trabalhado com cuidado por se uma das principais dificuldades observadas nos estudantes.

3.2.3 Lição 3: Definição de classe

A definição de uma classe é apresentada aos alunos utilizando o exemplo de uma máquina de vender bilhetes (TicketMachine no livro texto que acompanha o BlueJ). Os estudantes criam instâncias desta classe, chamam seus métodos, observam a implementação em Java da classe e por fim, tem a oportunidade de realizar pequenas modificações nela.

No primeiro ano do curso foram identificadas as dificuldades mais comuns referentes ao ensino do paradigma de POO, é importante que o instrutor esteja atento a elas desde o início do ensino para reforçar o entendimento destes tópicos. Segue abaixo as principais dificuldades encontradas:

- A compreensão e uso de múltiplos construtores;
- O uso do *this* é de difícil compreensão e em geral os estudantes o evitam;
- A inicialização dos campos no construtor;
- Costumam declarar o tipo de retorno de métodos “set”;

- Tentam acessar valores de campos privados diretamente, sem o uso de métodos *get*;

Outras dificuldades são atribuídas mais especificamente ao uso dos recursos do BlueJ, são elas:

- **Não declaração do tipo das variáveis que armazenam os objetos**. Isso pode ser trabalhado dando ênfase na janela de instanciação de um objeto à variável onde o objeto será armazenado e o tipo desta variável. O BlueJ já apresenta estas informações, porém, comumente passam despercebidas.
- **Chamada de métodos**. Frequentemente métodos com retorno são chamados da mesma maneira que métodos void. No BlueJ, é possível chamar ambos os tipos de métodos da mesma forma, de modo qque o resultado de métodos com retorno frequentemente é interpretado como “impressão do valor na tela”;

Por fim, é essencial verificar se os conceitos aprendidos no objectKarel foram efetivamente transportados para o Java. Isso pode ser feito solicitando que os estudantes representem em Java os modelos dos robôs criados nas últimas duas lições.

3.2.4 Lição 4: Interação entre objetos

Estudantes são apresentados aos conceitos de modularização, abstração e interação entre objetos utilizando o exemplo de um relógio digital com duas classes, uma representando um relógio digital e outra que representa um mostrador de dois números. O Relógio é composto por dois mostradores, um para as horas e outro para os minutos.

As dificuldades observadas nesta etapa foram:

- Os estudantes não tem certeza onde devem inicializar os campos;
- Não sabem como manipular este novo tipo;
- Tem dificuldades com a notação de ponto.

Para estas dificuldades é recomendado o uso de diagramas de objetos e atividades onde os estudantes identifiquem chamadas de método internas e externas;

3.2.5 Liçao 5: O método main

nesta aula é apresentado como executar código Java sem o BlueJ. Uma introdução sobre uso de métodos estáticos versus métodos de instância é feita utilizando o BlueJ.

Após, os estudantes criam seu próprio método main no Jcreator para um projeto que eles já tenham conhecimento como o TicketMachine.

Esta lição também é aproveitada para falar sobre compilação, “byte code” e máquina virtual.

As principais dificuldades encontradas foram:

- Uso de parâmetros formais da chamado do construtor (Ex: **new Classe(Type param);**);
- O tipo das variáveis dos objetos são esquecidos;
- Esquecem argumentos em chamadas de métodos;
- Esquecem os parênteses em chamadas de métodos sem argumentos;

Para lidar com estas questões é sugerido:

- Atividades que são de alguma forma “silenciadas” pelo BlueJ como o tipo das

variáveis que armazenam os objetos são demonstrados utilizando as caixas de diálogo do BlueJ e seu paralelo em sintaxe Java;

- Exemplos e tarefas são realizados em cima de projetos já conhecidos;
- Um programa contendo erros comuns é apresentado e os estudantes são ensinados a reconhecer os erros e tratá-los.

3.2.6 Lição 6: Agrupando objetos em um ArrayList

Uso de coleções em programação é comum e necessário mesmo em programas de baixa complexidade. Alguns autores já colocam o uso de ArrayList como prioridade sobre Arrays por ser uma estrutura mais adequada para representar listas continuas.

Durante os quatro anos de curso os autores identificaram diversas dificuldades envolvendo o uso de ArrayList`s e chegaram às seguintes conclusões:

1. O uso dos diagramas de objetos são fundamentais desde o início do projeto. Os estudantes são estimulados a criar um diagrama de objetos para um dado programa e simular operações de remoção e inclusão no diagrama de objetos.
2. São oferecidos exercícios para com código incompleto para que os estudantes preencham os espaços vazios com determinados comandos.
3. Após, são construídos programas do zero.

3.2.7 Lição 7: Agrupando objetos em coleções (ArrayList vs Array)

O uso de Arrays é apresentado e o mesmo projeto no qual foi trabalhado o uso de ArrayList é agora modificado para funcionar com Arrays.

3.2.8 Lição 8: Utilizando bibliotecas de classe

Nesta aula os estudantes são apresentados à documentação das classes Java, com ênfase nas classes String e HashMap. A aula também apresenta a estrutura de pacotes.

3.2.9 Lição 9: Projeto de Classes

O projeto de classes é algo difícil de ser realizado e a simples exposição dos principais conceitos teóricos para uma solução bem projetada não causa um grande impacto no aprendizado. É importante que os estudantes entendam através de uma aplicação os problemas de um mau projeto de classes.

A maior dificuldade encontrada foi a limitação do tempo que impossibilita o desenvolvimento de uma solução completa, na qual, o entendimento dos principais conceitos de um bom projeto de classes seria trabalhado.

3.2.10 Lição 10: Melhorando a estrutura com Herança

O projeto desta lição é uma biblioteca multimídia de CD's e DVD`s. A primeira versão do projeto não usa herança, de modo que muito código acaba sendo duplicado.

Quando os alunos são perguntados sobre como resolver o problema, eles se recordam da apresentação feita com o objectKarel e facilmente compreendem o uso de herança.

A maior dificuldade percebida é a chamada do construtor da superclasse através da subclasse. Uma sugestão é utilizar o debugger para ver a execução do código passo a passo e os campos sendo preenchidos.

3.2.11 Lição 11: Polimorfismo e Sobrescrita

Através de uma continuação do exemplo visto anteriormente, é realizado um exemplo onde cada subclasse sobrescreve o comportamento de um método na classe pai.

3.2.12 Lição 12: Classes abstratas e Interface

Nesta aula o uso de classes abstratas e interface são utilizados para expandir a simulação de um ambiente com raposas e coelhos para conter diversos tipos de animais.

Percebeu-se que não é difícil para os estudantes compreenderem o conceito e o implementarem, porém, demonstram grande dificuldade em perceber quando uma classe deve ser declarada como concreta, abstrata ou interface.

4 Conclusão

As duas principais conclusões do autor do artigo são 1) a necessidade de avaliação continua dos cursos de modo a validar as pesquisas e dar continuidade aos processos de melhoria e que 2) mais de uma ferramenta é necessário para suportar as diferentes necessidades cognitivas no ensino de programação.

O artigo também clarifica como integrar os diferentes tipos de ambientes de ensino de programação desenvolvidos até o momento de modo a garantir um ensino eficaz ao mesmo tempo em que prepara os estudantes para o mercado.

É muito rica a descrição detalhada de todas as aulas realizadas e das principais dificuldades em cada tema.

Por fim, fica registradoa importância de, durante todas as atividades, coletar os feedbacks dos alunos de modo a buscar soluções e criar uma experiência de aprendizado realmente efetiva.

Stelios Xinogalos (2009). **Guidelines for Designing and Teaching an Effective Object-Oriented Design and Programming Course**, Advanced Learning, Raquel Hijn-Neira (Ed.), ISBN: 978-953-307-010-0, InTech, Available from: http://www.intechopen.com/books/advanced-learning/guidelines-for-designing-and-teaching-an-effective-object-oriented-design-and-programming-course