Uma Experiência Lúdica no Ensino de Programação Orientada a Objetos

Adilson Vahldick[1]

[1]Departamento de Sistemas e Computação
Universidade Regional de Blumenau (FURB) Blumenau, SC Brasil
adilsonv77@gmail.com

Resumo. *Este artigo relata a condução da disciplina de Programação Orientada a Objetos com Java num curso de Computação utilizando um ambiente de desenvolvimento lúdico em que se podem programar classes e associá-las com figuras, inserindo posteriormente os objetos num mundo gerenciado pelo ambiente. Esses objetos podem se movimentar na tela e reagir de acordo com o pressionamento de teclas. Esses recursos oferecem a possibilidade de desenvolvimento de jogos como exercícios de prática da disciplina.*

1. Introdução

São conhecidos pelos professores das disciplinas de programação as dificuldades que os alunos enfrentam para passar pela barreira inicial da compreensão do que é construir um programa. Rapkiewicz (2006, p. 1-2) comenta que a falta de motivação e a dificuldade de desenvolver o raciocínio lógico necessário para a construção dos algoritmos são os fatores principais da desistência dos ingressantes em programação.

Vários trabalhos tem sido desenvolvidos (GALHARDO; ZAINA, 2004; PEREIRA JR, 2004; RAABE, 2005; THEISS, 2006; MATTOS; FUCHS, 2007) para auxiliar alunos e professores a superarem as dificuldades e traumas iniciais no ensino de programação. Depois que o aluno alcançou o mínimo de conhecimento necessário para a produção de um programa, ou seja, o domínio de estruturas condicionais, de repetição, atribuição de variáveis e modularização, ele enfrenta um novo problema nos semestres seguintes: a quebra de paradigma da programação estruturada para a programação orientada a objetos.

Segundo Börstler, Bruce e Michiels (2003, p. 84), a Programação Orientada a Objetos (POO) é, e deve ser, um tópico principal na ciência da computação visto que é o paradigma mais evidente nas linguagens e ambientes de programação na atualidade (Java, .NET, C++, Ruby, Python, PHP 5). Inúmeros aspectos devem ser considerados para ajudar o aluno a pensar orientado a objeto, como classes, encapsulamento, objetos, associação, herança e polimorfismo, os quais são os conceitos essenciais.

Esse artigo é o relato de uma experiência do autor como professor da disciplina de Programação Orientada a Objetos em Java em que foi evidenciado o uso de metáforas, da interação com objetos e na escolha de um ambiente de programação que favoreça o lúdico no aprendizado.

2. Fase Introdutória

As primeiras aulas de POO são consideradas as mais importantes de todo o semestre. Assim como os primeiros anos de vida de uma criança são as determinantes na personalidade do adulto, as primeiras semanas de POO são fundamentais para a compreensão do conceito mais básico e a essência de todo o paradigma que é a utilização de objetos e a definição das classes.

Na FURB, onde aconteceu essa experiência, a disciplina de POO era ministrada no terceiro semestre. Os alunos já haviam passado por 180 horas de programação usando Portugol e C. Logo, as sintaxes de atribuição de variáveis, das estruturas de condição, repetição, passagem de parâmetros e retornos de funções consomem pouco tempo dessa disciplina.

Para a introdução do conceito de objetos, foram utilizadas caixas de sapato confeccionadas com abas externas para fixação de tiras de papel para anotação das características dos objetos. A estrutura da caixa está ilustrada na figura 1. Na tampa da caixa são anotados o nome do objeto e o tipo do objeto (classe). Na folha da lateral são anotados o que se pode fazer com o objeto, ou seja, a lista de métodos. Dentro da caixa existe outra tira de papel onde os alunos anotavam as variáveis e valores que precisam ser guardados para esse objeto, ou seja, os atributos.

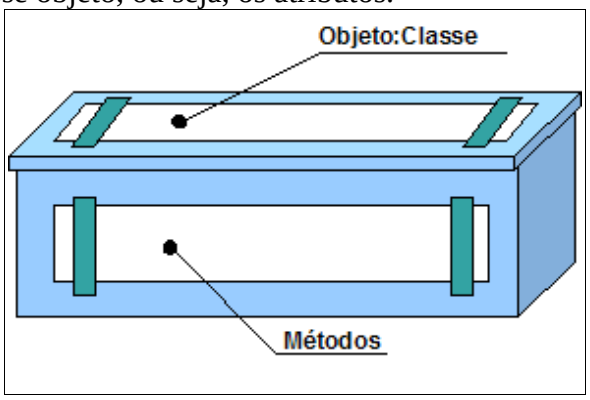

Figura 1. Caixa de sapato estruturado para ser um objeto

Os exercícios iniciais dos alunos foram leituras de código fonte. Essa estratégia é chamada de *Read Before Write* (BERGIN, 2002) que consiste em primeiro apresentar ao aluno exemplos e artefatos que ele consegue entender, para depois exigir dele a produção de artefatos de mesma complexidade. Não se pode exigir do aluno a criação de classes se ele jamais viu antes um código fonte que mostre como é uma classe.

Nesse processo de interpretação, os alunos tinham que identificar o nome da classe, os atributos e as assinaturas dos métodos e anotar essas informações nas tiras de papel de acordo com as regras expostas nos parágrafos anteriores. Os exercícios sempre constavam de uma classe com um método `main()`, a qual foi batizada de classe de sistema . É nesse método que acontece a instanciação dos objetos e neste momento entravam em cena as caixas. De acordo com o código do método `main()`, os alunos interagiam com essas caixas. Essa ferramenta tornou-se muito útil para as aulas em sala onde se podia exercitar a criação e o uso de objetos.

Notou-se que essa abordagem foi muito útil para aqueles que tinham dificuldades de abstração. Para outros, que tinham mais facilidade, tornou-se enfadonho.

Essa diferenciação foi observada quando eles não contavam mais com as caixas para executar os exercícios.

No laboratório necessitou-se de uma ferramenta que fosse uma extensão das aulas em sala, ou seja, um ambiente que mostrasse como e quando objetos fossem instanciados, métodos utilizados e atributos acessados. (SOUSA, 2002, p. 10). Já havia um ambiente adotado pelos professores anteriores, que era o BlueJ, e que se encaixava no método que se estava usando em sala. O BlueJ tende a facilitar o processo de aprendizado de POO através de recursos visuais interativos, como por exemplo classes e relacionamento que podem ser definidos visualmente. Araújo e Daibert (2006, p. 37) afirmam que no BlueJ é possível verificar o comportamento dos objetos em memória durante a execução. Os objetos são representados como caixas, que com cliques é possível inspecionar os valores dos seus atributos e invocar seus métodos.

O ambiente não possui recursos de autocompletar ou assistentes que facilitam o trabalho do programador, como em muitos ambientes profissionais, como o NetBeans ou o Eclipse. Entretanto, o recurso de criar objetos e manipulá-los sem necessidade de escrever código para isso, ou seja testar diretamente as classes, é que torna essa ferramenta atraente para os iniciantes de POO.

A figura 2 mostra como isso acontece no BlueJ. A área numerada com (1) é onde são visualizadas e adicionadas as classes. Para editar o código de uma classe basta fazer um duplo clique sobre o retângulo da classe. Com o botão direito sobre esse retângulo é possível criar um objeto. O objeto criado é apresentado na área (2) em cor vermelha. Com o clique do botão direito sobre esse objeto é apresentado um menu com todos os métodos que podem ser invocados. Um duplo clique sobre o objeto abre uma nova janela mostrando os valores contidos nos atributos.

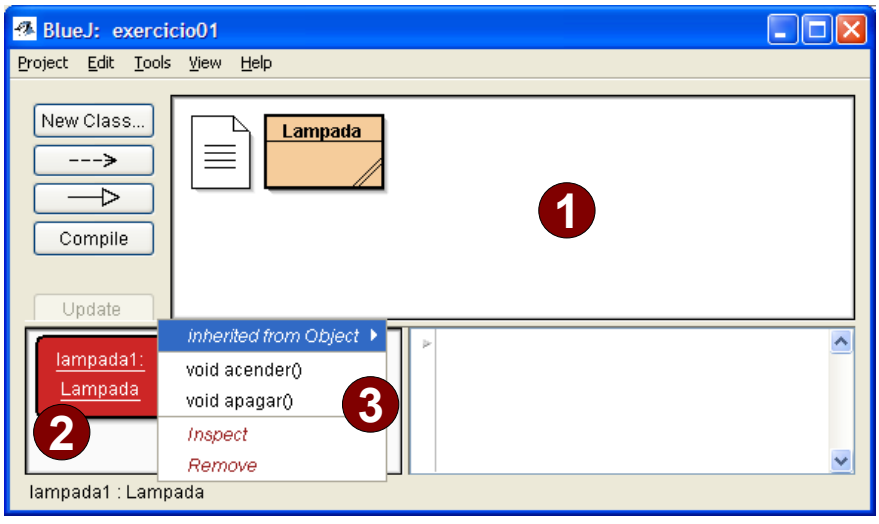

Figura 2. Ambiente BlueJ

Essa ferramenta foi utilizada durante toda a primeira unidade da disciplina que aborda os assuntos de classe, objetos, métodos e atributos. A dificuldade em não ter tarefas tão automatizadas na edição de código, com o autocompletar e o assistentes, exigem mais atenção e compreensão na produção da classe. O sucesso maior era alcançado por aqueles que seguiam uma recomendação do professor: *não inicie a*

codificação de um novo método enquanto o método atual não fizer a classe ser compilável.

3. O Ambiente Lúdico

Um terço da disciplina é o estudo de conceitos básicos: classes, objetos, métodos, atributos e encapsulamento. O assunto seguinte é o mecanismo de herança. Existe um discussão sobre o que estudar primeiro: herança ou associação? O autor desse artigo acredita que a herança deva vir antes da associação, pois concentra o aluno a enxergar a solução dos problemas em somente uma classe. Isso lhe dá a oportunidade de passar por mais algumas experiências na produção de classes sem pensar numa estrutura envolvendo outras classes para resolver os objetivos de algum método, o que exige um poder de abstração que a maioria a essa altura do semestre não alcançou ainda.

Existe um ambiente que utiliza o BlueJ como ferramenta de edição de código, o qual chama-se Greenfoot. Esse ambiente oferece um mundo para representar visualmente os objetos, ou seja, o aluno, ao implementar uma classe, precisa definir uma figura que será utilizada quando for instanciar um objeto nesse mundo. O Greenfoot é um framework e um ambiente para criar aplicações interativas e simulações num plano bidimensional (HENRIKSEN e KÖLLING, 2004).

O framework consiste de somente quatro classes: `World`, `Actor`, `Greenfoot` e `GreenfootImage`. As duas primeiras são abstratas. Essas quatro classes oferecem possibilidades abrangentes de exercícios e o tamanho limitado do framework favorece a compreensão do todo por parte dos alunos. As classes que compõe o framework são `World`, representa o mundo onde os objetos serão visualizados; `Actor` são os próprios objetos; `GreenfootImage` é a representação gráfica dos atores, com recursos para desenhar, onde se pode carregar uma imagem, escrever textos, desenhar linhas, elipses e retângulos e configurar cores; e `Greenfoot` é uma classe utilitária, que possui métodos estáticos para controlar a execução da simulação, interceptar teclas e um método para sortear números.

O ambiente Greenfoot é apresentado na figura 3. Na área (1) estão representadas as classes de mundo. Os alunos precisam implementar uma classe descendente de `World`. Ao implementar essa descendente, o aluno informa o tamanho de cada célula, a quantidade de células verticais e horizontais, os quais caracterizarão o mundo. A instanciação desse mundo acontece automaticamente quando essa classe é compilada.

Na área (2) estão as classes descendentes de `Actor`. Como `Actor` é uma classe abstrata, não é possível instanciá-la, logo a necessidade de estendê-la. São essas as classes centrais dos exercícios e que demandam o esforço por parte dos alunos. Por exemplo, a área (2) da figura contém duas classes descendentes de `Actor`. Ao clicar com o botão direito sobre uma delas, é possível criar um novo objeto, que em seguida ele é posicionado em alguma célula no mundo.

O mundo é apresentado na área (3). É ali que os atores vivem . É possível movimentá-los, mudar a direção deles, trocar a imagem, entre outros comportamentos.

A área (4) contém o controle da simulação: botões para executar a simulação, configurar a velocidade da execução, reiniciar o mundo ou executar passo-a-passo.

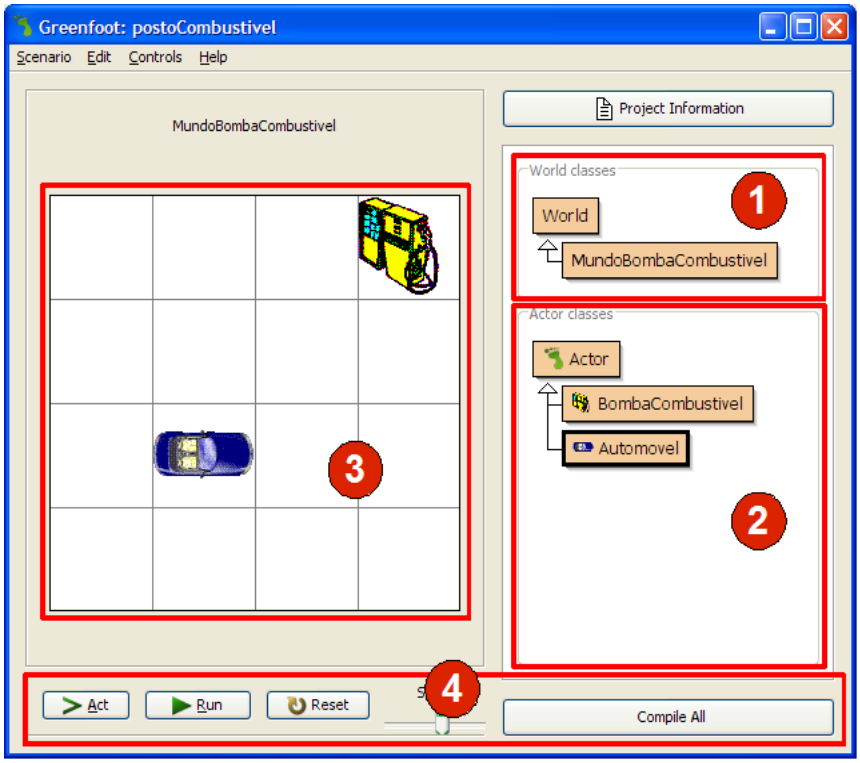

Figura 3. Ambiente Greenfoot

A classe Actor possui um método `act()`, o qual é chamado a cada ciclo da simulação. A lógica do ator, ou seja, sua interação com o mundo, precisa ser implementada nesse método. Na classe `Actor` já existem métodos para, por exemplo, posicionar o objeto numa célula do mundo, descobrir se existe outro ator numa célula específica, ou ao seu redor, e trocar a imagem dele.

Inspirado na palestra Princípios Básicos de Desenvolvimento de Jogos proferida em 11/04/2007 pelo sr. Dennis Kerr Coelho no auditório do Campus IV da FURB, a lógica do método `act()` foi dividida em três métodos protegidos (`protected`): um para processar as teclas, outro para executar a simulação e um terceiro para atualizar o estado do ator no mundo. Basicamente o método `act()` deveria manter o código como mostrado na figura 4.

```
public void act() {
      processarTeclas();
      simular();
      desenhar();
}
```

Figura 4. Método act() de todos os atores

No método `processarTeclas()` é solicitada a última tecla pressionada, e de acordo com o resultado é atualizado um atributo com o significado dessa tecla. Por exemplo, um atributo `direcao` do tipo `int`, que recebe `1` se foi pressionada a tecla direcional para cima. A classe `Greenfoot` possue dois métodos para interceptar pressionamento de teclas.

O método `simular()` define uma nova ação utilizando as teclas pressionadas, os valores dos atributos de estado ou os dados vindos do mundo. Por exemplo, se o atributo `barraEspacoPressionado` for igual a `true`, então o método `simular()` decide que vai ser necessário criar um novo ator `Projetil` e adicioná-lo ao mundo. O `simular()` não pode executar ações que reflitam visualmente no mundo. Por exemplo, o `simular()` não pode mudar a posição do ator, trocar a imagem, ou adicionar um novo ator no mundo. Nesse caso, ele simplesmente pode decidir se é ou não necessário mudar a posição do ator, e quais são essas coordenadas, ou que precisa adicionar um novo ator.

É no método `desenhar()` que acontece a atualização no mundo, de acordo com as decisões do `simular()`. Seguindo o exemplo do parágrafo anterior, no método `simular()` foi decidido que o ator precisa ir para as coordenadas x = 10 e y = 0. O posicionamento do ator no mundo, com essas coordenadas, é então feito no método `desenhar()`.

A adoção dessa divisão constitui numa prática dos alunos em criar métodos simples e que eles atendam a somente um objetivo, mantendo o estado do objeto em atributos.

4. Experiência com o Greenfoot

Os projetos do Greenfoot eram disponibilizados num arquivo compactado. No projeto era fornecido o enunciado acessível através do botão [Project Information] (localize na figura 2) e mais figuras que podiam ser associadas com os atores, e em alguns desses projetos eram adicionadas algumas classes de ator prontas e na maioria das vezes a classe de mundo.

Nas primeiras aulas foi apresentado o ambiente e adaptado um exercício da primeira unidade, migrando as classes básicas feitas no BlueJ para atores do Greenfoot. Nas aulas seguintes foram entregues dois novos enunciados também baseados em classes que os alunos já desenvolveram na unidade anterior. O objetivo desses exercícios eram a ambientação com a ferramenta e com o framework.

Somente após o terceiro exercício é que foi explicado para obedecerem a divisão do método `act()` explanada na seção anterior. À medida que os exercícios avançavam, menos recursos prontos (classes de ator ou mundo, e figuras) estavam disponíveis nos projetos. A certa altura, os projetos continham somente o enunciado.

Para a avaliação da aprendizagem, foram feitas duas verificações (provas) em laboratório: uma para a unidade dois (herança) e outra para a unidade três (associação). Também foi solicitado um trabalho final onde os alunos precisavam implementar o jogo Pacman (Come-come). O Greenfoot permite o uso de arquivos de áudio e imagens animadas (formato GIF). Esses recursos foram utilizados por cinco alunos.

5. Conclusões

Ao utilizar formas diferenciadas de implementar os exercícios, com o objetivo de motivar os alunos através da implementação de jogos, o autor desse artigo acreditava numa aprovação da disciplina de 60 a 70% da turma. O histórico da disciplina apresenta um índice de aprovação entre 30% a 40%.

A estatística da experiência é apresentada na tabela 1. O universo foi de 56 alunos distribuídos em duas turmas: no matutino com 11 alunos e no noturno com 45 alunos.

Tabela 1. Estatística da Turma

Quesito	Quantidade	Percentual (%)
Universo	56	---
Primeira Prova (Nota >= 6)	27	48,21
Segunda Prova (fizeram)	35	62,50
Segunda Prova (Nota >= 6)	15	26,79
Trabalho Final (fizeram)	25	44,64
Trabalho Final (Nota >= 6)	21	37,50
Aprovados (Nota >= 6)	**19**	**33,93**

Num primeiro momento os números mostram que a situação não melhorou com o uso do Greenfoot. É possível observar que na primeira prova, metade da sala não conseguiu alcançar os conceitos fundamentais da orientação a objetos, o que compromete o progresso no restante do semestre. Observa-se que 40% da turma desistiu antes da segunda prova. A tabela 2 apresenta os números ajustados no universo de 35 alunos baseado naqueles que fizeram a segunda prova.

Tabela 2. Estatística da Turma (Novo Universo)

Quesito	Quantidade	Percentual (%)
Universo	35	62,50
Segunda Prova (Nota >= 6)	15	42,86
Trabalho Final (fizeram)	25	71,43
Trabalho Final (Nota >= 6)	21	60,00
Aprovados (Nota >= 6)	**19**	**54,29**

Nesse novo universo se pode observar que metade daqueles que fizeram a segunda prova foram aprovados na disciplina. Essa proporção ainda não é agradável. Entretanto, a quantidade de aprovados é maior que aqueles que obtiveram nota 6,0 ou superior na segunda prova, e os aprovados estão muito próximos aqueles que chegaram ao trabalho final.

Analisando a primeira tabela verifica-se que boa parte da turma não assimilou os conceitos básicos, o que de certa forma, compromete o rendimento no restante do semestre. O uso das caixas ajudou na compreensão de uso de um objeto, mas a tática não contribuiu na produção de uma classe nova.

Os números não foram motivadores. Existe uma avaliação que não é possível representar em números: comparando com turmas anteriores da mesma disciplina, observou-se que a qualidade do código final dos que aprovaram nesse semestre é superior ao das turmas anteriores. Isso acontece porque os alunos que tendenciosamente dedicam-se a resolver os exercícios tem um retorno imediato do sucesso ou fracasso de suas classes e são motivados a implementar o próximo método ou regra no ator. Além disso, existe o desafio de implementar uma simulação com mais recursos, com imagens melhores, com recursos de multimídia e com um mundo mais complexo.

Referências

ARAÚJO, Marcos Antonio Pereira; DAIBERT, Marcelo Santos. Introdução ao BlueJ: aprenda visualmente programação OO e Java. Java Magazine, São Paulo, n 37, p. 62-67, 2006.

BERGIN, Joseph. Some Pedagogical Patterns. New York, 2002. Disponível em: <http://www.csis.pace.edu/~bergin/patterns/fewpedpats.html#rbw> Acesso em: 04 out. 2007.

BÖRSTLER, Jürgen; BRUCE, Kim; MICHIELS, Isabel. Sixth workshop on pedagogies and tools for learning object oriented concepts. In: ECOOP, 17., 2003, Darmstadt. Anais... [Darnastadt]: [s.n.], [2003]. p. 84-87.

GALHARDO, Mariane Forgaça; ZAINA, Luciana A. Martinez. Simulação para ensino de conceitos da orientação a objetos. In: XIII Seminário de Computação - SEMINCO, Blumenau, 2004. Anais... Blumenau: Furb, 2004. p. 109-116.

HENRIKSEN, Poul; KÖLLING, Michael. greenfoot: Combining Object Visualisation with Interaction. In Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications (OOPSLA), pages 73-82, Vancouver, BC, CANADA, November 2004.

MATTOS, Mauro Marcelo. FUCHS, Jean Fábio. Qualifica: Uma Ferramenta para Apoio a Construção de Algoritmos Estruturados. In: XVI Seminário de Computação - SEMINCO, Blumenau, 2007. Anais... Blumenau: Furb, 2007. p. 75-87.

PEREIRA JR., José. C. R. et al. Ensino de algoritmos e programação: uma experiência no nível médio. São Leopoldo, 2004. Disponível em: <http://www.unisinos.br/_diversos/congresso/sbc2005/_dados/anais/pdf/arq0033.pdf > Acesso em: 04 out. 2007.

RAABE, André Luís Alice. Uma proposta de arquitetura de Sistema Tutor Inteligente baseada na teoria das experiências de aprendizagem mediadas. 2005. 152f. Tese (Doutorado) Programa de Pós-Graduação em Informática na Educação, Universidade Federal do Rio Grande do Sul, Porto Alegre.

RAPKIEWICZ, Clevi E. et al. Estratégicas pedagógicas no ensino de algoritmos e programação associadas ao uso de jogos educacionais. Rio Grande do Sul, 2006. Disponível em: <http://www.cinted.ufrgs.br/renote/dez2006/artigosrenote/25157.pdf > Acesso em: 03 out. 2007.

SOUSA, Fábio Cordova de. Utilização da reflexão computacional para implementação de um monitor de software orientado a objetos em Java, 2002. 53f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

THEISS, Fabrício. Linguagem Visual Orientada por figuras geométricas voltada para o ensino de programação: versão 2.0. 2006. 83 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.