UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA–INFORMÁTICA APLICADA
Técnicas de Construção de Programas – 2025/1
Professora Karina Kohl –Turma A

Trabalho Prático - Etapa 0: Especificação do Aplicativo

Adriel de Souza, Ana Sanfelice, Aline Fraga, Eduardo Veiga, Stefany Nascimento

1.2 Mudanças em relação à Etapa anterior

Colocamos todo conteúdo previsto do diagrama de classes em código implementado, porém com os nomes ajustados em inglês com a adição da classe Office.

Na superclasse Task (tarefa), as alterações foram a retirada de disciplina dos atributos, visto que a classe tarefa está dentro de uma disciplina e não precisaria ter essa informação dentro de tarefa, além disso, foi criado método getStatus e foi removido o método visualizarTarefa, pois isso se dá diretamente na interface visual. Na subclasse Work foi ajustado o único atributo para toDo, visto que é importante ao ter um trabalho extenso, poder colocar o que precisa ser feito para concluí-lo. Implementamos todo o conteúdo previsto no diagrama de classes em código, ajustando os nomes para inglês.

No package de Courses (Disciplinas), foram inseridos as classes de Course (Disciplina), Teacher (Professor), Office (Sala) e Schedule (Horario_Disciplina). Dessa maneira foi criado uma classe extra para ajustar a sala de um professor e tornar o código mais legível e evitar o code smell de obsessão primitiva. Além disso foram feitas as seguintes alterações em relação ao diagrama da etapa anterior:

- Adicionado getters e setters para todos atributos da Classe Courses (Disciplina)
- Métodos de visualizar horários e tarefas podem ser resumidos ao seus getters, a visualização gráfica cabe a interface gráfica.
- O atributo Teacher (Professor) se tornou uma lista e consequentemente foi criado o método add Teacher.
- Adicionado polimorfismo aos construtores da classe Course, podendo criar um objeto com professores e horários instanciados ou não.
- Criada uma classe para a sala do professor.

1.3 Implementação

Na etapa de implementação, o foco foi a modelagem das classes do domínio conforme o projeto elaborado anteriormente, priorizando a correta representação dos conceitos centrais do sistema acadêmico.

A classe Task é a superclasse de Exam, Lab e Work, usando a ideia de herança, criamos atributos e métodos que são comuns a todas as tasks. Além disso, Task se relaciona com Course através de uma agregação Os atributos comuns são name, description, date e status. Status informa se foi concluído ou não. Além do construtor com todos os atributos, temos os métodos getters e setters de todos os atributos.

A classe Work é subclasse de Task, como atributo exclusivamente dela temos toDo, que vai uma explicação do que precisa ser feito para o trabalho e, junto disso, terá o getter e setter desse atributo, o construtor usando o “super” para os atributos da classe-mãe. A classe Lab é também subclasse de Task, como atributo criado nela temos room e nos métodos o getter e setter de room, além do construto. Na subclasse Exam temos dois atributos room e content, e seus getters e setters e o construtor.

A classe Course foi implementada para reunir as principais informações de uma disciplina, como código, nome, créditos, professores, horários, tarefas e o link do Moodle. Os atributos são tipados de acordo com sua função, utilizando listas para professores, horários e tarefas, e tipos simples para os demais dados. O construtor principal permite criar cursos já com listas preenchidas, enquanto construtores alternativos facilitam a criação de objetos vazios ou parciais. O encapsulamento é garantido por atributos privados e acesso via getters e setters, que também validam os dados. Os métodos de adição (addTeacher, addSchedule, addTask) asseguram que apenas objetos válidos sejam inseridos nas listas, lançando exceções para valores nulos, e o método setLinkMoodle permite atualizar o link do Moodle com validação.

A classe Teacher representa um professor, com atributos para nome, e-mail e um objeto Office (sala). Todos os atributos são privados e acessados por getters e setters, que também validam os dados. A relação entre Course e Teacher é de associação simples: um curso pode ter vários professores, e cada professor pode estar associado a um curso e também a um escritório (Office). A classe Office encapsula informações sobre sala e prédio, com métodos para acessar e modificar esses valores, além de um método para retornar uma descrição textual do escritório. Assim, a relação entre Teacher e Office é de composição simples também, pois o escritório faz parte da identidade do professor.

1.4 Teste

Para a superclasse Task, casos de teste: Construtor e getters é testado os valores atribuídos ao criar o objeto pelo construtor e o de Setters, onde busca modificar os atributos após a criação.

Para a subclasse Work, casos de teste: Construtor é testado a herança dos atributos e o toDo e o de getters e setters para ver se os valores estão sendo atualizados corretamente. Os testes da subclasse Exam são semelhantes ao anterior, primeiro faz-se um teste para verificar a herança e os novos atributos da subclasse e depois testa se getters e setters modificam valores corretamente e o mesmo ocorre nos testes da subclasse Lab.

Para a classe Course, os testes garantem que a inicialização dos objetos e o funcionamento dos métodos de acesso e modificação estejam corretos. São verificados os valores dos atributos após a criação do objeto, bem como a atualização das listas internas e demais campos via setters, assegurando integridade e encapsulamento.

Para a classe Teacher, os testes verificam se os atributos nome, e-mail e escritório são corretamente atribuídos e recuperados, além de testar a modificação desses dados. Isso garante que a associação com o objeto Office funciona e que os dados do professor podem ser alterados com segurança.

Para a classe Office, os testes cobrem a criação do objeto, a alteração dos atributos e a geração da descrição textual do escritório. Assim, assegura-se que os métodos de acesso e modificação funcionam corretamente e que a representação textual reflete os dados do objeto.

A utilização de testes unitários durante a implementação foi fundamental para garantir a qualidade e a robustez do código. Os testes permitiram identificar rapidamente erros de lógica, problemas de encapsulamento e inconsistências nos métodos de acesso e modificação dos objetos. Além disso, a escrita dos testes antes ou durante a implementação ajudou a definir claramente o comportamento esperado das classes, servindo como documentação viva do sistema.

O grupo percebeu que, ao adotar testes unitários, o processo de refatoração e evolução do código tornou-se mais seguro e eficiente, pois qualquer alteração que introduzisse um erro era imediatamente detectada pelos testes. Isso proporcionou maior confiança no funcionamento correto das classes do domínio e facilitou a integração entre diferentes partes do sistema. Em resumo, a experiência com testes unitários foi extremamente positiva e reforçou a importância dessa prática no desenvolvimento de software orientado a objetos.

1.5 Executável (aplicação, interface)

https://docs.google.com/document/d/1ukUF26PJ2a4sb4GtQD0WUc_8o_9RjMRGHkeSIW-Ocq0/edit?usp=sharing