UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
ESCOLA DE INFORMÁTICA APLICADA

UMA LINGUAGEM PARA DESCRIÇÃO DE CONTEÚDOS DE TRILHAS DE APRENDIZAGEM COM APLICAÇÃO AO ENSINO DE COMPUTAÇÃO

Luísa Caetano Correia

Orientador: Prof. Jobson Luiz Massollar da Silva

Rio de Janeiro, RJ - Brasil
Janeiro de 2024

Catalogação informatizada pelo(a) autor(a)

C824 Caetano Correia, Luísa
Uma linguagem para descrição de conteúdos de trilhas de aprendizagem com aplicação ao ensino de computação / Luísa Caetano Correia. -- Rio de Janeiro, 2024.
123 f

Orientador: Jobson Luiz Massollar da Silva.
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal do Estado do Rio de Janeiro, Graduação em Sistemas de Informação, 2024.

1. Trilha de Aprendizagem. 2. Compilador. 3. DSL. I. Massollar da Silva, Jobson Luiz, orient. II. Título.

UMA LINGUAGEM PARA DESCRIÇÃO DE CONTEÚDOS DE TRILHAS DE APRENDIZAGEM COM APLICAÇÃO AO ENSINO DE COMPUTAÇÃO

Luísa Caetano Correia

Trabalho de Conclusão de Curso da Universidade Federal do Estado do Rio de Janeiro como requisito parcial para obtenção do título de bacharel em Sistemas de Informação.

Aprovado por:

_____

Prof. Jobson Luiz Massollar da Silva (UNIRIO)

_____

Prof. Paulo Sérgio Medeiros dos Santos (UNIRIO)

_____

Profª Geiza Maria Hamazaki da Silva (UNIRIO)

AGRADECIMENTOS

Agradeço aos meus pais, José Fernandes e Lucimeri, que me apoiaram imensamente no decorrer do curso. Também agradeço ao meu orientador, Jobson Luiz Massollar da Silva, por sua inspiração e dedicação como professor, e a minha equipe do estágio - Heron, Diobert, Ana, Higor, Rodrigo e Rômulo - que realizei durante o último ano e me permitiu adquirir a experiência em programação que eu precisava.

RESUMO

Profissionais da área de TI estão, cada vez mais, procurando adquirir conhecimento de forma autodidata. Isso significa que há uma grande demanda por materiais de aprendizado, com os quais o interessado possa aprender sobre os conceitos de determinado assunto, fazer exercícios de fixação e adquirir mais conhecimento que o ajudará no mercado de trabalho. Para suprir essa demanda, existem pesquisas que estudam como as trilhas de aprendizagem podem ser organizadas e construídas de forma simples e eficiente. Neste trabalho foi realizada uma pesquisa relacionada a modelos de trilhas de aprendizagem, documentação do desenvolvedor e linguagens específicas de domínio para propor uma forma de criar conteúdos de aprendizagem. Como resultado, esse trabalho apresenta uma DSL (*Domain Specific Language*), denominada LCML (*Learning Content Markup Language*), cujo objetivo é descrever conteúdos de trilhas de aprendizagem, e também um compilador que traduz esses conteúdos para HTML. Como prova de conceito foram desenvolvidos, na linguagem LCML, seis conteúdos de uma trilha de aprendizagem sobre TDD (*Test Driven Development*). Além do sucesso na elaboração dos conteúdos e na sua conversão para HTML, percebeu-se também que a utilização da LCML pode reduzir o esforço no desenvolvimento dos conteúdos de aprendizagem.

Palavras-chave: trilha de aprendizagem, conteúdo, DSL, TDD, compilador

ABSTRACT

IT professionals are increasingly seeking to acquire self-taught knowledge. This means that there is a great demand for learning materials, in which interested parties can learn about the concepts of a given subject, do exercises and acquire more knowledge that will help them in the job market. To meet this demand, there is research that studies how learning paths can be organized and constructed in a simple and efficient way. In this work, research was carried out related to learning path models, developer documentation and domain-specific languages to propose a way of creating learning contents. As a result, this work presents a DSL (Domain Specific Language), called LCML (Learning Content Markup Language), whose objective is to describe learning path content, and also a compiler that translates this content into HTML. As a proof of concept, six contents of a learning path about TDD (Test Driven Development) were developed in the LCML language. In addition to the success in preparing the content and converting it to HTML, it was also noticed that the use of LCML can decrease the effort in developing learning content.

Keywords: learning path, content, DSL, TDD, compiler

LISTA DE FIGURAS

Figura 1: Modelo de trilha de aprendizagem ……..... 15
Figura 2: Relacionamento entre os 3 componentes do conhecimento de uma API ……... 17
Figura 3: Funcionamento do TDD Traffic Light …… 18
Figura 4: Exemplos de cabeçalhos em LCML ……... 21
Figura 5: Exemplos de códigos em LCML …… 21
Figura 6: Exemplos de listas em LCML …… 22
Figura 7: Exemplos de conceitos em LCML ……. 22
Figura 8: Exemplos de tabelas em LCML ……. 23
Figura 9: Exemplos de imagens em LCML ……... 24
Figura 10: Exemplos de vídeos em LCML …… 24
Figura 11: Exemplos de cenários de uso em LCML ……... 24
Figura 12: Exemplos de questões em LCML ……. 25
Figura 13: Exemplos de parágrafos em LCML ……... 26
Figura 14: Exemplos de formatação de textos em LCML ……. 27
Figura 15:Arquitetura do compilador ……. 28
Figura 16: Tokens gerados para o comando #img …….. 30
Figura 17: Tokens gerados para a resposta de uma #question …… 31
Figura 18: Código exemplo em LCML …….. 32
Figura 19:Lista sintática gerada pelo compilador para o código da Figura 16 …… 33
Figura 20: Código LCML dos conceitos do TDD …….. 50
Figura 21: Página HTML gerada para os conceitos do TDD …… 51
Figura 22: Código LCML dos exercícios de fixação dos conceitos do TDD …… 52
Figura 23: Página HTML gerada para os exercícios de fixação dos conceitos do TDD 53
Figura 24: Código LCML das funções básicas do JUnit …… 54
Figura 25: Página HTML gerada para as funções básicas do JUnit ……... 56
Figura 26: Código LCML dos exercícios de fixação do JUnit ……... 57
Figura 27: Página HTML gerada para os exercícios de fixação do JUnit …….. 62
Figura 28: Código LCML da aplicação prática do TDD …… 63
Figura 29: Página HTML gerada para aplicação prática do TDD ……. 70
Figura 30: Código LCML da avaliação final do aprendizado …… 71
Figura 31: Página HTML gerada para a avaliação final do aprendizado ……... 76

LISTA DE TABELAS

Tabela 1: Comandos da LCML ……..……... 20
Tabela 2: Marcações de texto definidas na LCML ……. 27
Tabela 3: Tabela de Contextos da LCML …….……. 32
Tabela 4: Classes CSS e IDs atribuídos às tags HTML ……..……. 36
Tabela 5: Exemplos de conversão de cabeçalhos ……..…… 36
Tabela 6: Exemplos de conversão de códigos……..…..……. 37
Tabela 7: Exemplos de conversão de listas……..…..……. 38
Tabela 8: Exemplos de conversão de conceitos…….……. 39
Tabela 9: Exemplos de conversão de tabelas……..…..…….…… 40
Tabela 10: Exemplos de conversão de imagens……..…..…… 41
Tabela 11: Exemplos de conversão de vídeos……..…….…….. 42
Tabela 12: Exemplos de conversão de cenários de uso……..…….…… 43
Tabela 13: Exemplos de conversão de questões…….……. 45
Tabela 14: Exemplos de conversão de parágrafos……..……..…… 46
Tabela 15: Exemplos de conversão de formatação de texto……..…….. 47
Tabela 16: Quantidade de caracteres dos conteúdos em LCML x conteúdos em HTML .. 77

SUMÁRIO

1. INTRODUÇÃO..... 11
1.1 MOTIVAÇÃO..... 11
1.2 OBJETIVOS..... 12
1.3 ORGANIZAÇÃO DO TEXTO..... 13
2. FUNDAMENTAÇÃO TEÓRICA..... 14
2.1 MODELOS DE TRILHAS DE APRENDIZAGEM..... 14
2.2 DOCUMENTAÇÃO DO DESENVOLVEDOR..... 16
2.3 TEST DRIVEN DEVELOPMENT..... 17
2.4 LINGUAGENS ESPECÍFICAS DE DOMÍNIO..... 19
3. LINGUAGEM PARA DEFINIÇÃO DE CONTEÚDO..... 20
3.1 CABEÇALHOS..... 21
3.2 CÓDIGOS..... 21
3.3 LISTAS..... 22
3.4 CONCEITOS..... 22
3.5 TABELAS..... 23
3.6 IMAGENS..... 23
3.7 VÍDEOS..... 24
3.8 CENÁRIOS DE USO..... 24
3.9 QUESTÕES..... 25
3.10 PARÁGRAFOS..... 26
3.11 ELEMENTOS DE FORMATAÇÃO DE TEXTO..... 26
4. IMPLEMENTAÇÃO DO COMPILADOR LCML..... 28
4.1 ARQUITETURA DO COMPILADOR..... 28
4.2 READER..... 29
4.3 ANALISADOR LÉXICO..... 29
4.4 ANALISADOR SINTÁTICO..... 31
4.5 ANALISADOR SEMÂNTICO..... 34
4.6 GERADOR..... 35
4.7 GERADOR DE HTML..... 35
4.7.1 Estilização..... 35
4.7.2 Cabeçalhos..... 36
4.7.3 Códigos..... 37
4.7.4 Listas..... 38
4.7.5 Conceitos..... 38
4.7.6 Tabelas..... 40
4.7.7 Imagens..... 41
4.7.8 Vídeos..... 42
4.7.9 Cenários de Uso..... 43

4.7.10 Questões.....44
4.7.11 Parágrafos.....46
4.7.12 Formatação de Textos.....47
5. CONTEÚDO DE APRENDIZAGEM PARA TDD.....49
5.1 TRILHA DE INTRODUÇÃO AO TDD.....49
5.1.1 Principais Conceitos do TDD.....50
5.1.2 JUnit.....54
5.1.3 Aplicação Prática do TDD.....63
5.1.4 Avaliação Final do Aprendizado.....71
5.2 CONCLUSÃO DA PROVA DE CONCEITO.....77
6. CONCLUSÃO.....78
REFERÊNCIAS BIBLIOGRÁFICAS.....80
APÊNDICE A - DEFINIÇÃO FORMAL DA LCML.....81
APÊNDICE B - PÁGINAS HTML DOS CONTEÚDOS DE TDD.....83

1. INTRODUÇÃO

1.1 MOTIVAÇÃO

Com o avanço da tecnologia, as oportunidades de emprego relacionadas à área de Tecnologia da Informação têm aumentado e, com isso, também vem crescendo a necessidade de especialização da mão-de-obra. Nesse cenário, a internet vem exercendo um papel essencial como um meio de oferta de cursos e materiais - gratuitos ou pagos - que atendam a essa demanda. Segundo uma pesquisa feita pelo Stack Overflow (2023), 80% dos desenvolvedores optam por adquirir conhecimento por meio de fontes online, como vídeos, blogs e fóruns, e esse valor é 10% maior que o do ano passado.

Há alguns anos vem surgindo, de forma cada vez mais frequente, plataformas de aprendizado com o intuito de facilitar a educação autodidata para capacitação profissional. Nesse contexto, um projeto de pesquisa na UNIRIO propôs a plataforma LearningCurve (SILVA, 2022), que visa “apoiar a elaboração de documentações de desenvolvedor com maior qualidade e de forma colaborativa e auxiliar no aprendizado de estudantes de TI através da documentação de software”. No contexto dessa mesma pesquisa, Campos (2023) define o conceito de trilhas de aprendizagem e um modelo para definição dessas trilhas, cujo objetivo é organizar um conjunto de recursos de aprendizagem em uma certa sequência que surge através do planejamento de como o aprendizado do estudante irá ocorrer (YANG, 2013).

Um dos elementos definidos no modelo de trilha de aprendizagem proposto por Campos (2023) são os conteúdos. Conteúdos definem os elementos básicos de estruturação da informação que será consumida pelos desenvolvedores ao percorrerem uma trilha de aprendizagem. Thayer (2021) propõe a categorização dos conteúdos em três tipos: conceitos de domínio, fatos de execução e padrões de uso. O primeiro representa as ideias e os conceitos relacionados ao contexto da aprendizagem, o segundo define as regras que explicam comportamentos da tecnologia e suas consequências e o terceiro representa o passo a passo de ações para realizar uma determinada tarefa naquela tecnologia. Apesar do trabalho de Thayer discutir a organização desses conteúdos no contexto do aprendizado de APIs, entende-se que ele pode ser aplicado ao aprendizado de outras tecnologias da área de TI, como uma linguagem de programação, por exemplo.

A partir desse cenário, a oportunidade de pesquisa que surge está na criação dos conteúdos de aprendizagem, de forma que os autores das trilhas de aprendizagem disponham de um instrumento para descrição desses conteúdos, de forma que essa descrição não esteja associada a nenhuma tecnologia específica de apresentação como HTML[1], CSS[2], Markdown[3] ou LaTeX[4]. Dessa forma, o conteudista poderá focar seus esforços na definição e criação do conteúdo propriamente dito, sem se importar com a tecnologia que será efetivamente usada na apresentação final desse conteúdo ao desenvolvedor.

1.2 OBJETIVOS

O objetivo desse trabalho pode ser resumido como: desenvolver um mecanismo para criação de conteúdos de aprendizagem. Esse objetivo pode ser dividido nos seguintes tópicos:

1. Definir uma DSL (*Domain Specific Language*) para descrição de conteúdos, segundo a categorização de Thayer (2021).
2. Implementar um compilador que traduza os conteúdos descritos nessa linguagem para HTML, de forma que o conteúdo criado possa ser visualizado em um navegador.
3. Elaborar uma prova de conceito para o tema TDD (*Test Driven Development*), ou seja, usando a DSL proposta serão criados conteúdos focados no aprendizado dessa tecnologia e tais conteúdos serão compilados para páginas HTML.

Ao cumprir esses objetivos, entende-se que os principais resultados deste trabalho são a definição da linguagem para criação de conteúdos de aprendizagem e seu compilador. Esses resultados contribuem para facilitar o processo de criação de conteúdos, uma vez que permite que o conteudista se concentre na definição do conteúdo propriamente dito, e sua disponibilização em um formato amplamente acessível, já que esse conteúdo pode ser acessado via navegador.

É importante também ressaltar que não está no escopo deste trabalho a integração do compilador aqui desenvolvido com as demais ferramentas criadas para a plataforma LearningCurve.

[1] https://www.w3.org/html/
[2] https://www.w3.org/Style/CSS/
[3] https://www.markdownguide.org/
[4] https://www.latex-project.org/

1.3 ORGANIZAÇÃO DO TEXTO

Além desta introdução, este trabalho está organizado nos seguintes capítulos:

- Capítulo 2: Fundamentação teórica - Apresenta os conceitos fundamentais sobre modelos de trilhas de aprendizagem, documentação do desenvolvedor, TDD e DSL.
- Capítulo 3: Linguagem para definição de conteúdo - Apresenta a DSL desenvolvida neste trabalho para descrição de conteúdos de trilhas de aprendizagem.
- Capítulo 4: Implementação do compilador - Apresenta a arquitetura do compilador que processa a DSL proposta, seus componentes e o conversor para HTML.
- Capítulo 5: Conteúdo de aprendizagem para TDD - Apresenta uma prova de conceito através da criação de uma trilha de aprendizagem para TDD utilizando a DSL desenvolvida neste trabalho.
- Capítulo 6: Conclusão - Apresenta as conclusões finais deste trabalho.

2. FUNDAMENTAÇÃO TEÓRICA

2.1 MODELOS DE TRILHAS DE APRENDIZAGEM

De acordo com Lopes e Lima (2019), a definição de trilha de aprendizagem não pode se resumir apenas a um conjunto de passos, procedimentos ou prescrições, porque isso engessaria muito o processo de ensino-aprendizagem. Por isso, também é necessário desenvolver uma estrutura de organização para ela. Avaliando diversos conceitos de trilha, as autoras chegaram a três possíveis categorias:

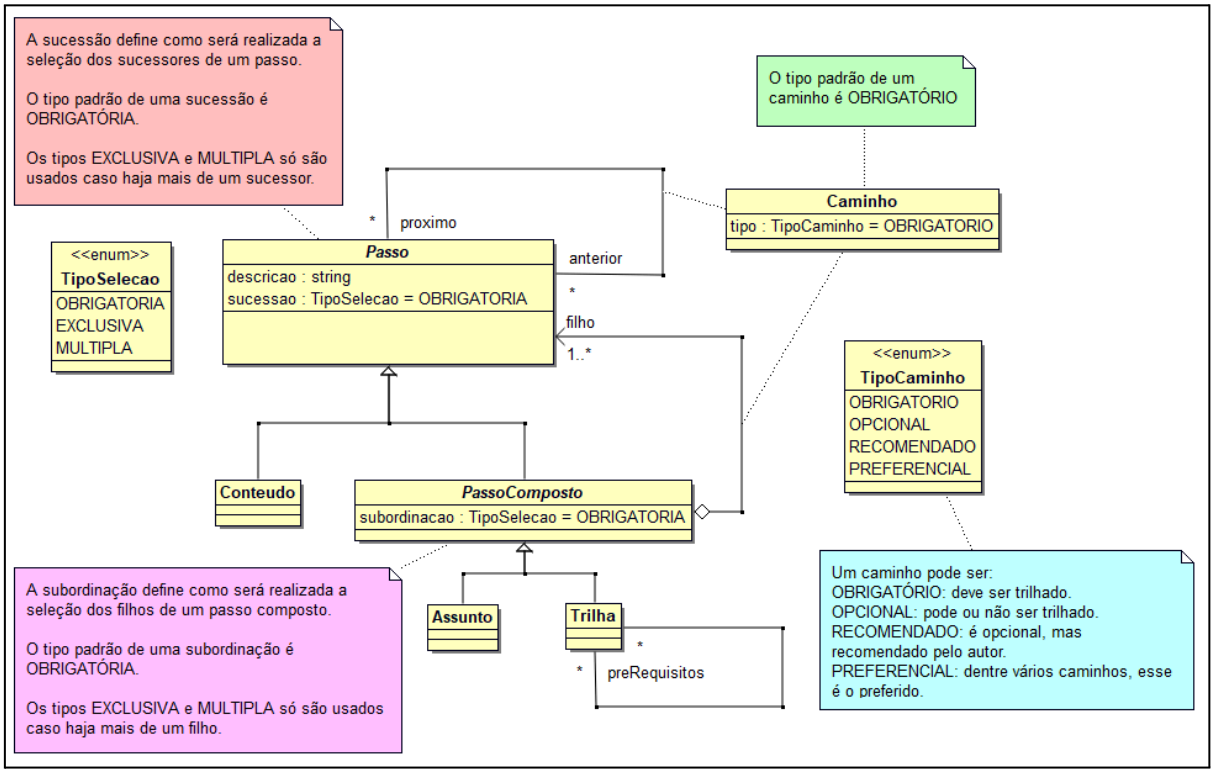
- Primeira Categoria: a trilha se trata de um modelo a ser seguido.
- Segunda Categoria: a trilha é conceituada como uma ação, ato ou efeito de percorrer um caminho.
- Terceira categoria: a trilha equivale ou representa o registro de caminhos percorridos.

As trilhas que estão na primeira categoria são comparadas em diversos estudos, por Lopes e Lima (2019), com um mapa conceitual, sendo a diferença que no caso de uma trilha haveria uma organização de sequência de aprendizagem, definindo como o aluno aprenderia o tema. Uma trilha que representa a organização da sequência necessária para aprender um determinado tema pode entrar na primeira categoria, considerando que um caminho determinado pode ser considerado um modelo. Porém, também pode entrar na segunda categoria, sendo um ato de trilhar um percurso. Nesse caso, estariam classificadas as definições cujos percursos são passíveis de escolhas por parte dos alunos, devido ao seu foco nas ações feitas durante o percurso da trilha. Levando em conta todas essas definições, uma trilha de aprendizagem pode ser considerada como caminhos que podem ser escolhidos pelo aluno para que ele adquira conhecimento gradativamente, já que a segunda categoria considera flexibilidade para tomada de decisões no processo de aprendizado. Como a terceira categoria se trata do registro de caminhos percorridos, a representação acaba sendo fundamental para esse modelo, que geralmente é feita utilizando grafos.

Depois de levantar todos esses pontos Lopes e Lima (2019) chegaram a conclusão de que uma trilha de aprendizagem seria um conjunto sistemático e multimodal de unidades de aprendizagem com diversas formas de navegação, podendo ser personalizados de acordo com os seus objetivos, perfil do aluno ou até características de aprendizagem. A definição de trilha

de aprendizagem de Yang *et al.* (2014) é um conjunto de objetos que são responsáveis por definir como o curso de um estudo se desenvolve, utilizando passos que um aluno deve fazer para aprender o conteúdo. Em cada um deles, o estudante realiza ações de aprendizagem que seguem a pedagogia adequada para aprender determinado assunto. Essa definição reforça a definição de Lopes e Lima (2019). Além disso, esses autores consideram necessário haver alguma forma de avaliação na trilha de aprendizagem, já que é esperado que o aluno chegue a um determinado nível de conhecimento sobre o assunto conforme vai estudando-o.

Figura 1: Modelo de trilha de aprendizagem



Fonte: Campos (2023)

A Figura 1 apresenta um modelo de trilha de aprendizagem desenvolvido por Campos (2023), que se baseou nos conceitos de trilha de aprendizagem mencionados anteriormente. É possível perceber que o diagrama segue o padrão de projeto *Composite* (GAMMA et al., 1995), que é comum em estruturas de árvores. Nele, as trilhas de aprendizagem podem ser compostas por outras trilhas (nesse modelo pré-requisitos são considerados como trilhas) ou assuntos, que possuem caráter estruturante para organizar os temas abordados na trilha e também podem ser compostos por outros assuntos ou conteúdos, que representam os elementos básicos ou ações de aprendizado a serem executadas pelo aluno ao consumir a trilha. Além desses elementos, o modelo também define regras de como consumir esses conteúdos para concluir a ação de aprendizagem. É importante destacar que o objetivo desse

modelo é sugerir a estruturação de uma trilha de aprendizagem, mas ele não contempla a descrição dos conteúdos.

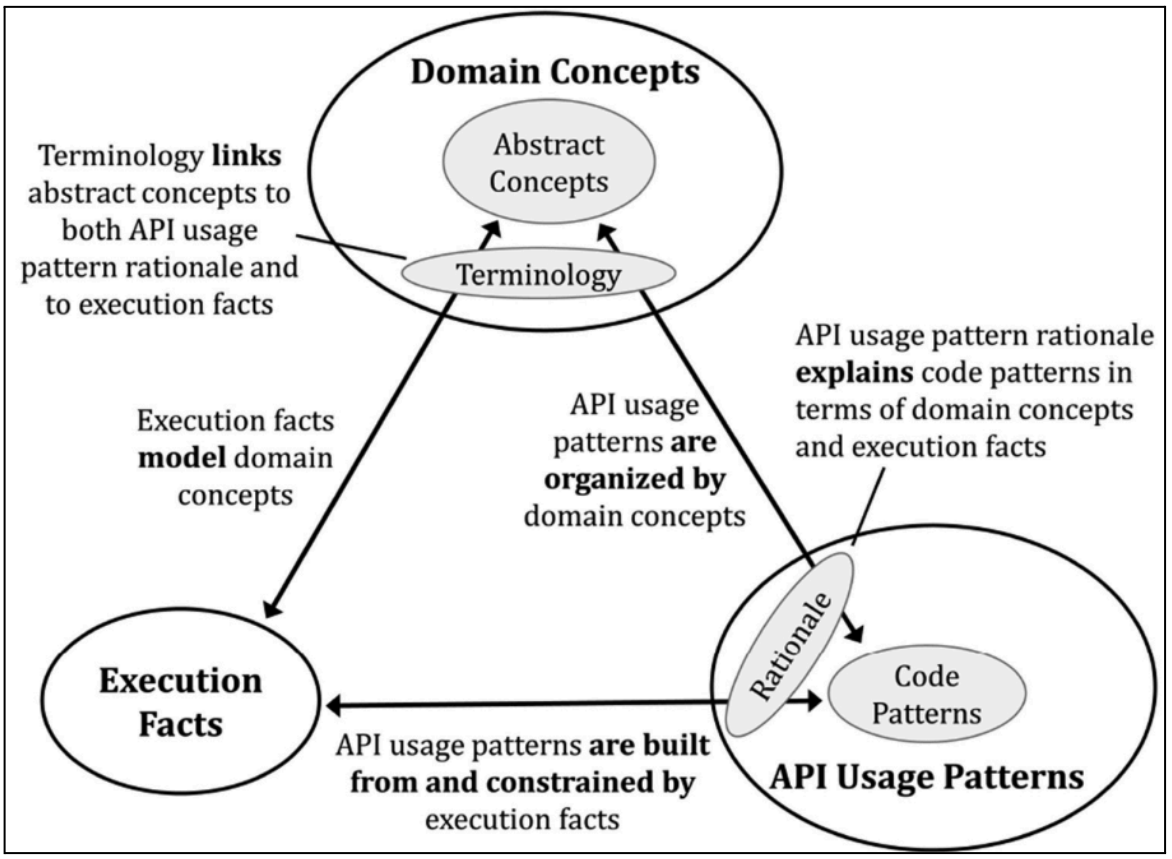
2.2 DOCUMENTAÇÃO DO DESENVOLVEDOR

Nos dias atuais, o desenvolvimento de softwares, dos mais simples aos mais complexos, geralmente demandam o uso de um conjunto variado de tecnologias. Essas tecnologias normalmente se apresentam sob a forma de APIs (*Application Program Interface*), *frameworks*, linguagens de programação, serviços remotos, práticas de desenvolvimento, dentre outros. Assim, é essencial que os desenvolvedores tenham acesso à documentação dessas tecnologias para que possam estudá-las, entender os detalhes do seu funcionamento e acessar exemplos de seu uso. Por isso, é importante que essa documentação tenha uma qualidade adequada, mas o que significa uma documentação de qualidade? Thayer *et al.* (2021) definem que o conhecimento robusto de uma API requer três tipos de conhecimento: conceitos de domínio, fatos de execução e cenários de uso (Figura 2). O primeiro diz respeito ao mundo real fora da API que ela tenta modelar e a terminologia adotada na API. O segundo define o que cada parte da API faz, por exemplo: tratando-se de uma função quais seriam os seus parâmetros de entrada, os valores de saída e quais os seus efeitos colaterais. O último define a correlação das suas partes, ou seja, como as funções de uma API podem ser utilizadas em conjunto para resolver algum problema específico.

Esses três aspectos do conhecimento são centrais para que um desenvolvedor entenda o funcionamento de uma API e consiga desenvolver código utilizando essa API. Pode haver outros conhecimentos específicos que serão necessários, principalmente do ponto de vista de execução do código, já que para isso, talvez seja necessário conhecer o sistema operacional, outras plataformas e configurações. Contudo, apenas do ponto de vista de compreensão e capacidade de escrever um código coeso utilizando uma API, esses três elementos são suficientes, segundo Thayer *et al.* (2021). Também é possível desenvolver sem necessariamente ter esses três tipos de conhecimento, mas é preciso ter um conhecimento mínimo sobre eles para conseguir ter sucesso no desenvolvimento.

Apesar deste trabalho não ter como foco o aprendizado de APIs, a organização de conteúdos de aprendizagem para outras tecnologias como *frameworks*, linguagens de programação e até mesmo práticas de desenvolvimento, pode explorar as diretrizes propostas por Thayer *et al*. Assim, alguns comandos da linguagem de descrição de conteúdos de aprendizagem apresentada neste trabalho foram diretamente influenciados por essa proposta.

Figura 2: Relacionamento entre os 3 componentes do conhecimento de uma API



Fonte: Thayer *et al.* (2021)

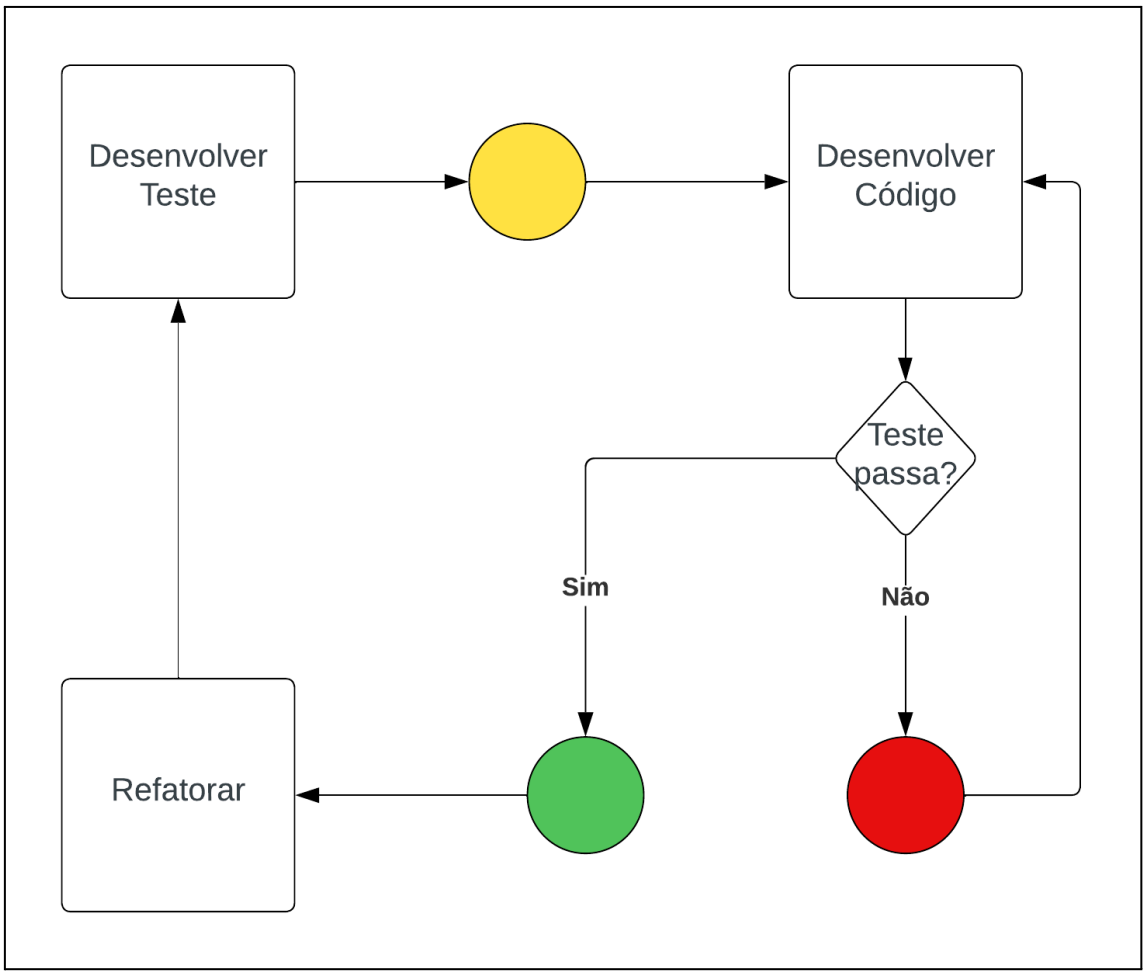
2.3 TEST DRIVEN DEVELOPMENT

O *Test Driven Development* (TDD), de acordo com Astels (2003), é uma técnica de desenvolvimento em que os testes são desenvolvidos antes do código. Ele afirma ainda que se deve utilizar uma lógica de semáforo chamada *TDD Traffic Light*, desenvolvida por William Wake, que considera erros de compilação como a luz amarela, falha em testes como a luz vermelha e sucesso nos testes como a luz verde. Um elemento extremamente fundamental para que TDD seja possível é a constante refatoração. Normalmente, desenvolve-se o mínimo necessário para que os testes passem e depois realizam-se várias refatorações até que o código possua a lógica de programação desejada.

Para desenvolver uma determinada funcionalidade utilizando TDD, o programador deve pensar nos testes mais simples possíveis para testar cada aspecto da funcionalidade. Quando tentar criar o primeiro teste, o *TDD Traffic Light* fica com a luz amarela, já que uma classe ou função que ainda não existe está sendo utilizada. Então, codifica-se o mínimo possível para resolver os problemas de compilação. Normalmente, nesse momento a luz muda para a vermelha, já que o código não tem nada implementado até então, apenas as estruturas

mais básicas foram criadas. Então, é necessário corrigir o código para que o teste passe, também da forma mais simples possível. Depois que o teste passar, verifica-se se há duplicação no código. Se sim, deve-se refatorá-lo, se não o próximo teste deve ser desenvolvido e o processo se repete até chegar ao fim do desenvolvimento da funcionalidade. A Figura 3 representa o fluxo do TDD descrito anteriormente:

Figura 3: Funcionamento do TDD Traffic Light



Fonte: Elaborada pela autora

Existem casos em que a luz verde pode vir logo após uma amarela, normalmente quando a linguagem de programação é fortemente tipada, como Java. Suponha que uma função que deva retornar um valor inteiro esteja sendo desenvolvida em TDD e que o primeiro teste verifica se o valor retornado é 0. Para resolver os erros de compilação, não basta apenas criar a função com os parâmetros necessários, mas também atribuir algum valor de retorno, que sendo o mais simples possível, deve ser retornado, nesse caso específico, o valor 0. Como o teste esperava que o valor fosse zero, o teste resultará em uma luz verde, o que acaba entrando no caso de exceção do *TDD Traffic Light*.

Algumas das vantagens do uso do TDD são: os testes são desenvolvidos mais rapidamente e a identificação de defeitos torna-se mais fácil. Geralmente os testes são feitos depois que o código está pronto, o que pode demorar mais tempo, já que o programador tem que relembrar o que fez ou a pessoa responsável pelos testes tem que parar para entender o código e depois testá-lo. Ademais, como cada trecho de código exige um teste relacionado a ele, é mais fácil identificar defeitos, porque se um teste não passar, o problema estará no código recém escrito, permitindo que ele seja corrigido imediatamente.

2.4 LINGUAGENS ESPECÍFICAS DE DOMÍNIO

Uma DSL (*Domain Specific Language*) (FOWLER, 2010) é uma linguagem projetada para cumprir um objetivo específico e, por conta disso, geralmente está associada a um domínio específico. Por isso, não é possível criar um sistema completo com ela, apenas uma parte dele, o que significa que ela deve ser combinada com outras linguagens. DSLs normalmente possuem comandos simples e focados, o que facilita seu uso pelos desenvolvedores e seu entendimento até mesmo por outros membros da equipe de desenvolvimento.

Segundo Fowler (2010), existem dois tipos de DSLs: externa e interna. A primeira é construída do zero em uma linguagem separada e possui um interpretador/compilador próprio. Muitas DSLs externas trabalham em conjunto com a principal linguagem de programação utilizada em um sistema, oferecendo formas mais adequadas de codificar certas partes desse sistema. Alguns exemplos de DSLs externas são: XML, SQL e HTML. Já as DSL 's externas são construídas a partir de uma linguagem de programação já existente, usando a sintaxe dessa linguagem, mas com o código escrito de forma diferente da forma usual, geralmente usando um formato especial de API denominada interface fluente (FOWLER, 2010). Um bom exemplo é o LINQ (*Language Integrated Query*), usada para escrever códigos de consulta a dados no C#.

A partir desses conceitos, o próximo capítulo irá detalhar a DSL proposta para definição de conteúdos de aprendizagem, seus comandos e exemplos de uso.

3. LINGUAGEM PARA DEFINIÇÃO DE CONTEÚDO

A DSL desenvolvida neste trabalho recebeu o nome de *Learning Content Markup Language* ou LCML, que tem como objetivo descrever conteúdos de uma trilha de aprendizagem. A ideia geral da linguagem é permitir a descrição de conteúdos usando uma sintaxe simples e leve (sem muitos símbolos ou palavras reservadas) e o principal marcador dos comandos da linguagem é o símbolo # (hashtag) no início de uma linha. A Tabela 1 apresenta os comandos da LCML que serão detalhados nas seções a seguir. No Apêndice A encontra-se a definição formal da LCML em BNF estendida. A BNF estendida adotada neste trabalho utiliza a sintaxe proposta pelo W3C (*World Wide Web Consortium*) e que foi usada na definição da sintaxe da linguagem XML (W3C, 2024). Por ser uma definição formal, as regras sintáticas da LCML podem ser facilmente compreendidas a partir das regras definidas no Apêndice A.

Tabela 1: Comandos da LCML

Comando	Descrição
#1, #2, #3	Cabeçalhos de nível 1, 2 e 3. Permitem criar seções nos conteúdos para organizar os diversos assuntos.
#code	Código: permite inserir códigos-fonte no conteúdo.
#list	Lista: permite criar uma lista de itens.
#concepts	Conceitos: permite inserir uma lista de conceitos e suas respectivas definições Thayer *et al.* (2021).
#table	Tabela: permite criar uma tabela com cabeçalho.
#image	Imagem: permite inserir uma imagem no conteúdo.
#video	Vídeo: permite inserir um vídeo no conteúdo.
#scenario	Cenário: permite definir um cenário de uso com o seu passo a passo e os respectivos detalhamentos Thayer *et al.* (2021).
#question	Questão: permite definir questões de múltipla escolha.

Fonte: elaborada pela autora

A linguagem LCML foi inspirada no Markdown[5] e nos conceitos de DSL externa de Fowler (2010). O objetivo era tornar a linguagem o mais simples possível de forma a facilitar a sua aprendizagem e uso e, por isso, nos casos em que há comandos de abertura e encerramento, a mesma expressão foi utilizada tanto para abrir quanto para fechar o comando.

[5] https://www.markdownguide.org/

Dentre os comandos citados na Tabela 1, dois cobrem a proposta de Thayer: #scenario (cenários de uso) e #concepts (conceitos de domínio). Não foi criado nenhum comando específico para fatos de execução, porque considerou-se que no contexto da trilha de aprendizagem eles seriam tratados como conceitos técnicos, ou seja, conceitos de alguma ferramenta, framework, dentre outros. Sendo assim, julgou-se que, para a definição dos conteúdos, os conceitos de domínio e os fatos de execução são semelhantes e, portanto, os fatos de execução pode ser representados como conceitos de domínio (#concepts).

3.1 CABEÇALHOS

É possível definir cabeçalhos de nível 1, 2 e 3 com os comandos **#1**, **#2** e **#3**, respectivamente (Figura 4). O texto do cabeçalho deve vir após o comando e deve ocupar apenas a própria linha do comando (não é possível definir cabeçalhos que ocupem mais de uma linha). Esse comando não gera nenhum tipo de numeração, logo esses devem ser fornecidos pelo conteudista, se assim ele quiser.

Figura 4: Exemplos de cabeçalhos em LCML

```
#1 1. TDD

#2 1.1 Conceitos de TDD

#3 1.1.1 Teste Unitário
```

Fonte: elaborada pela autora

3.2 CÓDIGOS

O comando **#code** (Figura 5) define o início e o fim de uma área de código-fonte e tudo que é escrito nessa área deve ser apresentado ao usuário exatamente da mesma forma como está escrito, ou seja, com quebras de linha, espaços, tabs, recuos, etc.

Figura 5: Exemplos de códigos em LCML

```
#code
x = x + 2;
#code

#code
public static String getString(String key) {
   return rb != null ? rb.getString(key) : key;
}
#code
```

Fonte: elaborada pela autora

3.3 LISTAS

O comando **#list** (Figura 6) define uma área de itens de uma lista. Cada linha dentro dessa área representa um item da lista, portanto, um item não pode ocupar duas ou mais linhas.

Figura 6: Exemplos de listas em LCML

```
#list
JUnit
Jest
#list


#list
Item número um da lista
Esse é o item dois da lista
Terceiro item da lista
#list
```

Fonte: elaborada pela autora

3.4 CONCEITOS

O comando **#concepts** (Figura 7) define uma área onde são descritos os conceitos relacionados a um domínio, tecnologia, framework, etc. Esse comando foi definido para que a LCML esteja aderente aos elementos propostos por Thayer *et al.* (2021), apresentados no capítulo 2. Cada conceito é descrito em uma linha iniciada por um **-** (hífen) seguido do nome do conceito. As linhas seguintes devem ser usadas para explicar ou detalhar o conceito e podem conter parágrafos, imagens, vídeos, listas, códigos ou tabelas.

Figura 7: Exemplos de conceitos em LCML

```
#concepts
- Conceito UM
Definição do conceito UM

- Conceito DOIS
Definição do conceito DOIS.
Continuação da definição do conceito DOIS

#concepts

#concepts
- TDD
Test Driven Development (Desenvolvimento Dirigido por Teste)

- JUnit
Framework para implementação de scripts de teste em Java

- Teste Unitário
Teste realizado em uma única unidade.
```

```
A unidade deve estar isolada das demais.

#concepts
```

Fonte: elaborada pela autora

3.5 TABELAS

O comando **#table** (Figura 8) define uma tabela e, opcionalmente, pode ser seguido do modificador **border** para indicar que a tabela deverá ter borda. Cada linha dentro da área da tabela representa uma linha da tabela e a primeira linha representa, obrigatoriamente, o cabeçalho da tabela (não é possível definir uma tabela sem cabeçalho). Em cada linha, as colunas são separadas pelo símbolo | (pipe). Opcionalmente, os textos dos cabeçalhos podem ser precedidos dos símbolos <, = ou > que indicam, respectivamente, alinhamento à esquerda (*default*), centralizado ou alinhamento à direita para os dados daquela coluna. Na versão atual, as células das tabelas só podem conter textos.

Figura 8: Exemplos de tabelas em LCML

```
#table
Cabeçalho da Coluna 1 | Cabeçalho da Coluna 2
Linha 1, coluna 1 | Linha 1, coluna 2
Linha 2, coluna 1 | Linha 2, coluna 2
#table

#table border
<Produto | =Categoria | >Preço
Mouse | A | 32,30
Notebook | B | 2727,48
Teclado | C | 128,65
#table
```

Fonte: elaborada pela autora

3.6 IMAGENS

O comando **#img** (Figura 9) define uma imagem e deve ter, como primeiro parâmetro, a URL da imagem. Opcionalmente, é possível definir um texto alternativo para a imagem ou sua largura e altura.

Figura 9: Exemplos de imagens em LCML

```
#img http://site.com/foto1.jpg

#img http://site.com/foto2.jpg 500 300

#img http://site.com/foto3.jpg "texto alternativo"

#img http://site.com/foto4.jpg "texto alternativo" 500 300
```

Fonte: elaborada pela autora

3.7 VÍDEOS

O comando **#video** (Figura 10) define um vídeo e deve ter, como primeiro parâmetro, a URL do vídeo. Opcionalmente, é possível definir a largura e a altura da área para apresentação do vídeo. Os controles comuns de *start*, *stop* e *pause* são sempre apresentados juntamente com o vídeo.

Figura 10: Exemplos de vídeos em LCML

```
#video https://www.site.com/movie.mp4 500 300

#video https://www.youtube.com/embed/tgbNymZ7vqY

#video https://www.youtube.com/embed/tgbNymZ7vqY 500 300
```

Fonte: elaborada pela autora

3.8 CENÁRIOS DE USO

O comando **#scenario** (Figura 11) define uma área onde são descritos os passos de um cenário de uso que ilustra o uso de uma tecnologia, API, framework, etc. Esse comando foi definido para que a LCML esteja aderente aos elementos propostos por Thayer *et al.* (2021), apresentados no capítulo 2. Cada passo do cenário de uso é descrito em uma linha iniciada por um **-** (hífen). As linhas seguintes devem ser usadas para explicar ou detalhar aquele passo e podem conter parágrafos, imagens, vídeos, listas, códigos ou tabelas.

Figura 11: Exemplos de cenários de uso em LCML

```
#scenario
- Passo um
Texto do passo um

- Passo dois
- Passo três
#scenario

#scenario
- Passo um
```

```
Parágrafo do passo um

- Passo dois
Parágrafo do passo dois

Outro parágrafo do passo dois

- Passo três
#img http://www.unirio.br/++theme++unirio-teste/img/logo_unirio.png
#scenario
```

Fonte: elaborada pela autora

3.9 QUESTÕES

O comando **#question** (Figura 12) define uma questão de múltipla escolha que pode ser usada para avaliar o progresso da aprendizagem, por exemplo. O enunciado da questão pode conter parágrafos, imagens, vídeos, listas, códigos ou tabelas. As respostas da questão de múltipla escolha iniciam com a combinação **()** (abre e fecha parênteses) e uma dela deve estar marcada com um ***** (asterisco) para indicar a resposta correta. Cada resposta também pode ser detalhada com parágrafos, imagens, vídeos, listas, códigos ou tabelas.

Opcionalmente esse comando pode vir seguido do nome de uma função que será executada quando o usuário selecionar uma das respostas. Esse parâmetro foi criado especificamente para a tradução da LCML em HTML, onde a correção da resposta pode ser realizada através da execução de um código Javascript. Essa discussão será realizada com mais detalhes no Capítulo 4.

Figura 12: Exemplos de questões em LCML

```
#question
Quanto é 2 x 3?

() Cinco
(*) Seis
() Oito
#question

#question assess
Primeiro parágrafo do enunciado

Segundo parágrafo do enunciado

() Primeira resposta
Parágrafo da primeira resposta

(*) Segunda resposta
#img http://site.com.br/image.jpg
() Terceira resposta
#code
```

```
if (x > y)
  x++;
#code
#question
```

Fonte: elaborada pela autora

3.10 PARÁGRAFOS

Parágrafos não são comandos propriamente ditos, mas qualquer texto não iniciado com # (hashtag), - (hífen), \ (contrabarra) ou ((abre parênteses) é tratado como um parágrafo (Figura 13). Como não existe um comando para determinar formalmente o início e o fim de um parágrafo, seu término é definido por uma linha vazia (linha em branco). Dessa forma, ao iniciar um parágrafo, todas as linhas subsequentes vão automaticamente fazer parte desse parágrafo até que seja encontrada uma linha vazia. Por isso, o fato de existirem duas linhas de texto distintas no código LCML não significa que o usuário verá exatamente duas linhas na apresentação final do conteúdo, pois ambas linhas fazem parte do mesmo parágrafo. Para forçar uma quebra de linha em um parágrafo deve-se acrescentar uma \ (contrabarra) como último caracter da linha. Da mesma forma, para forçar uma linha em branco no meio de um parágrafo deve-se acrescentar uma linha contendo somente uma \ (contrabarra).

Figura 13: Exemplos de parágrafos em LCML

```
Essa é a linha inicial de um parágrafo.
Essa frase vai continuar a linha anterior.
Fim do parágrafo.

Segundo parágrafo.
Essa frase continua a linha anterior.
Ao fim dessa linha tem uma quebra.\
Nova linha no mesmo parágrafo com uma quebra no final.\
\
Última linha do parágrafo.
```

Fonte: elaborada pela autora

3.11 ELEMENTOS DE FORMATAÇÃO DE TEXTO

Nos textos dos cabeçalhos, listas, conceitos, tabelas, cenários, questões e parágrafos é possível inserir marcações em LCML para definir links, negrito, itálico, sublinhado, tachado, fonte aumentada, subscrito e sobrescrito. A Tabela 2 apresenta essas marcações, que foram inspiradas no Markdown, e a Figura 14 mostra exemplos de uso dessas marcações

Tabela 2: Marcações de texto definidas na LCML

Marcação	LCML
Link	`[texto do link](url destino)`
Negrito	`**texto em negrito**`
Itálico	`*texto em itálico*`
Sublinhado	`__texto sublinhado__`
Tachado	`~~texto tachado~~`
Fonte aumentada	`++texto com fonte aumentada++`
Subscrito	`~texto subscrito~`
Sobrescrito	`^texto sobrescrito^`

Fonte: elaborada pela autora

Figura 14: Exemplos de formatação de textos em LCML

```
Clique [aqui](http://google.com) para acessar o Google!

Esse texto vai ser apresentado em **negrito**

Esse texto vai ser apresentado em *itálico*

Esse texto vai ser apresentado __sublinhado__

Esse texto vai ser apresentado ~~tachado~~

Esse texto vai ser apresentado com ++Fonte Aumentada++

H~2~O

X^i+1^
```

Fonte: elaborada pela autora

Este capítulo apresentou a linguagem LCML e seus comandos, com exemplos. No próximo capítulo serão apresentados o compilador LCML e sua arquitetura, assim como os detalhes da conversão do conteúdo LCML para HTML.

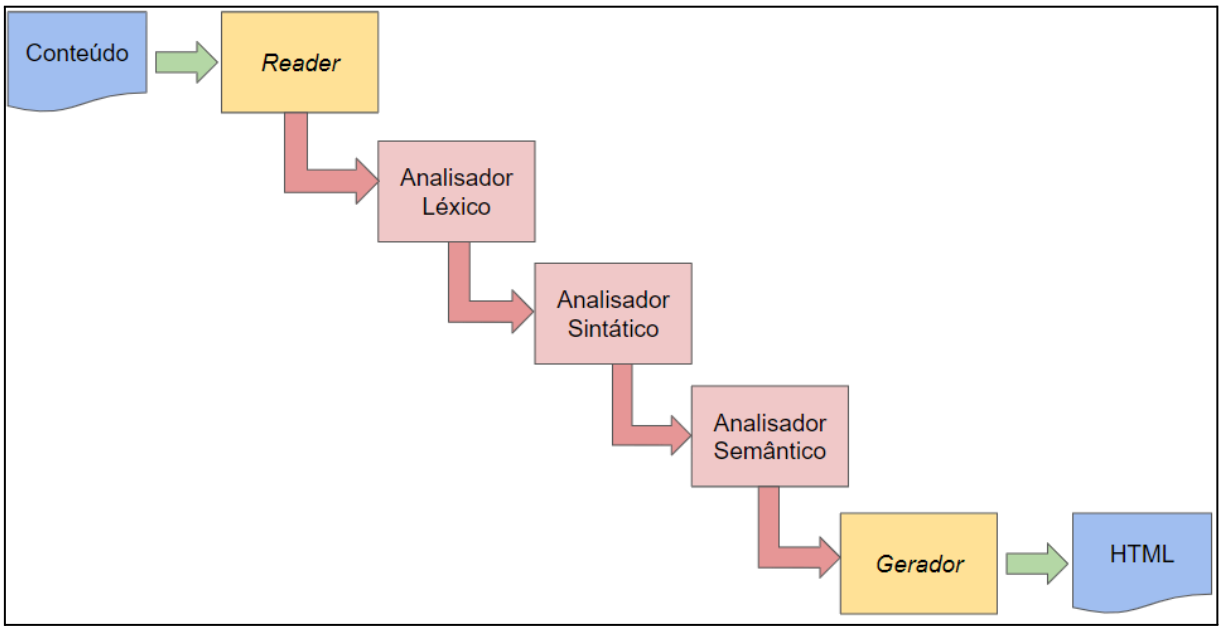
4. IMPLEMENTAÇÃO DO COMPILADOR LCML

A literatura técnica geralmente utiliza o termo "compilador" para programas que traduzem um código-fonte em alto nível para código de máquina (p. ex., compilador C/C++) ou para código intermediário de uma máquina virtual (p. ex., compilador Java). Entretanto, a palavra "compilador" também pode ser usada para programas que traduzem um código-fonte em alto nível para outro código em alto nível. Muitas vezes, esse último tem sido chamado de "transpilador", porém um transpilador é um tipo de compilador. Por isso, nesse TCC será adotado o termo "compilador".

4.1 ARQUITETURA DO COMPILADOR

A Figura 15 apresenta a arquitetura do compilador LCML, desenvolvido na linguagem Java (versão 20). Nela é possível observar os principais elementos e o fluxo entre eles: Reader, Analisador Léxico, Analisador Sintático, Analisador Semântico e Gerador. A solução foi implementada pensando na possibilidade de substituição de dois desses componentes (Reader e Gerador), de forma que o compilador possa ser flexível tanto na leitura dos dados de entrada quanto na geração do resultado final. Nas seções a seguir serão detalhados cada um desses componentes.

Figura 15: Arquitetura do compilador



Fonte: elaborada pela autora

O código do compilador pode ser acessado em um repositório do Github através do link https://github.com/LuisaCCorreia/Learning_Curve_LCML_Compiler.

4.2 READER

É o componente responsável pela leitura do código LCML. Essa leitura é feita linha a linha e cada linha lida é transferida para o Analisador Léxico, uma por vez. O Reader implementado neste trabalho lê o conteúdo de um arquivo em disco (a partir do nome desse arquivo). A arquitetura do compilador permite que esse componente seja substituído por outro componente sem maiores dificuldades. Dessa forma, outro Reader poderá ler o código LCML a partir de uma cadeia de caracteres ou stream.

4.3 ANALISADOR LÉXICO

O Analisador Léxico é responsável por analisar as linhas da LCML enviadas pelo Reader e quebrar essa linha em *tokens*. O *token* identifica o elemento léxico presente na linha do conteúdo. A implementação desse processo, chamado de *tokenização*, é feita linha a linha e adota expressões regulares para verificar se o comando está escrito corretamente e extrair os *tokens* correspondentes.

Considere, por exemplo, que o compilador está analisando um conteúdo que contém uma imagem, representada da seguinte forma:

```
#img http://www.unirio.br/++theme++unirio-teste/img/logo_unirio.png "texto alternativo" 100
50
```

Essa linha será enviada para o Analisador Léxico e ele vai comparar com as expressões regulares de uma lista da classe *Command*, que possui os atributos relacionados à regex, ao *token* e aos contextos em que aquele *token* é permitido. A regex que combina com a linha mostrada anteriormente é:

```
^#img[\\s]+(?<url>[^\\s]+)(?:[\\s]+(?<alt>\\\".+\\\"))?(?:[\\s]+(?<width>\\
d+)[\\s]+(?<height>\\d+))?[\\s]*$
```

Com a combinação bem sucedida será instanciado um objeto da classe *Pair*, que possui o tipo de *token* no lado esquerdo e uma estrutura de dados do tipo *Map* que possui o nome do grupo do regex como chave, tendo nesse caso 4 chaves com seus respectivos valores:

- **url**: http://www.unirio.br/++theme++unirio-teste/img/logo_unirio.png
- **alt**: “texto alternativo”
- **width**: 100
- **height**: 50

A Figura 16 representa os valores mencionados anteriormente, que é exibida no terminal ao utilizar o compilador:

Figura 16: Tokens gerados para o comando #img

```
Type: IMAGE
      width: [100] [posicao 88]
      alt: ["texto alternativo"] [posicao 68]
      url: [http://www.unirio.br/++theme++unirio-teste/img/logo_unirio.png] [posicao
5]
      height: [50] [posicao 92]
```

Fonte: Elaborada pela autora

Em um segundo exemplo, considere que o compilador está analisando um conteúdo que contém a seguinte questão:

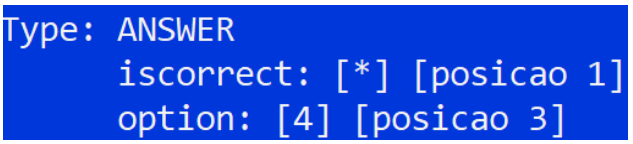
```
#question
Qual é o resultado da soma 2 + 2?
(*)4
()6
()8
#question
```

Ao analisar a terceira linha desse comando, ela irá combinar com a seguinte expressão regular:

```
^(?:\\()(?<iscorrect>\\*?)(?:\\))(?<option>.+)$
```

Note que essa expressão regular só faz sentido dentro do comando **#question**, por isso, algumas expressões regulares só são aplicadas em determinados contextos. Com a combinação bem sucedida ocorre a instanciação da classe *Pair*, que possui o tipo de token no lado esquerdo e uma estrutura de dados do tipo *Map*, que possui o nome do grupo do regex como chave, tendo nesse caso 2 chaves: **iscorrect** e **option**, com os respectivos valores ***** e **4**. A Figura 17 mostra como essa estrutura é exibida pelo compilador:

Figura 17: Tokens gerados para a resposta de uma #question



Fonte: Elaborada pela autora

Dessa forma, para cada comando encontrado no conteúdo, será buscada uma regex correspondente dentro do contexto em que se encontra o comando. Se o comando não combinar com nenhuma das expressões regulares é gerado o *token* INVALID e, consequentemente, um erro de compilação.

4.4 ANALISADOR SINTÁTICO

O Analisador Sintático é responsável por avaliar se a combinação dos *tokens* enviados pelo Analisador Léxico faz sentido. Para isso ele verifica se determinado *token* é válido no contexto em que ele foi encontrado e, se não for, é gerado um erro de compilação. Um exemplo: dentro do contexto de um comando **#code** não pode haver nenhum outro comando, como **#table** ou **#list**. Outro exemplo: no contexto de um comando **#scenario** não pode haver um comando **#question**. Se o Analisador Sintático não detectar nenhum problema, ele cria uma estrutura chamada lista sintática, que representa o código LCML como uma lista de *tokens*.

A Tabela 3 mostra os comandos e o contexto permitido em cada comando, marcados com a cor verde. Na coluna de comandos também constam os comandos internos, como o item de uma lista, por exemplo. O contexto “DEFAULT” seria o contexto mais básico. Os contextos são gerenciados em uma pilha, ou seja, quando é encontrada a abertura de um comando, seu contexto é colocado na pilha, e quando é encontrado o fechamento do comando, esse contexto é retirado da pilha. Assim, o contexto atual do analisar é sempre o que está no topo da pilha.

Tabela 3: Tabela de Contextos da LCML

COMANDOS	CONTEXTO							
	DEFAULT	PARAGRAPH	CODE	LIST	CONCEPTS	TABLE	SCENARIO	QUESTION
HEADER1								
HEADER2								
HEADER3								
IMAGE								
VIDEO								
BEGIN_CODE								
END_CODE								
SOURCE_CODE								
BEGIN_LIST								
END_LIST								
LIST_ITEM								
BEGIN_CONCEPTS								
END_CONCEPTS								
CONCEPT								
BEGIN_TABLE								
END_TABLE								
TABLE_ROW								
BEGIN_SCENARIO								
END_SCENARIO								
STEP								
BEGIN_QUESTION								
END_QUESTION								
ANSWER								
TEXT								
BREAK								
EMPTY_LINE								
EOF			Erro	Erro	Erro	Erro	Erro	Erro

Fonte: Criada pela autora

A Figura 18 apresenta um exemplo de código em LCML e a Figura 19 mostra a lista sintática gerada para esse código. Nela é possível perceber os tokens gerados pelos comandos e os fragmentos capturados pelo Analisador Léxico.

Figura 18: Código exemplo em LCML

```
#1 Título de Nível 1

Este é um parágrafo.
Parágrafos não são definidos por nenhum comando específico.
São simplesmente textos escritos de forma livre.

#2 Seção 1.1

#3 Subseção 1.1.1

#concepts
- TDD
Test Driven Development (Desenvolvimento Dirigido por Teste)
```

```
- JUnit
Framework para implementação de scripts de teste em Java

- Teste Unitário
Teste realizado em uma única unidade.
A unidade deve estar isolada das demais.

#concepts

#list
JUnit
Mockito
Jest
NUnit
#list

#code
int x = 0, y = 1;

if (x > y)
  x++;
else
  y++;
#code

#img http://www.unirio.br/++theme++unirio-teste/img/logo_unirio.png
"Logotipo da UNIRIO" 100 50

#video https://www.youtube.com/embed/tgbNymZ7vqY 500 300
```

Fonte: elaborada pela autora

Figura 19: Lista sintática gerada pelo compilador para o código da Figura 16

```
Token: HEADER1
       title: [Título de Nível 1] [position 3]
Token: BEGIN_PARAGRAPH
Token: TEXT
       paragraph: [Este é um parágrafo.] [position 0]
Token: TEXT
       paragraph: [Parágrafos não são definidos por nenhum comando
específico.] [position 0]
Token: TEXT
       paragraph: [São simplesmente textos escritos de forma livre.]
[position 0]
Token: END_PARAGRAPH
Token: HEADER2
       title: [Seção 1.1] [position 3]
Token: HEADER3
       title: [Subseção 1.1.1] [position 3]
Token: BEGIN_CONCEPTS
Token: CONCEPT
       concept: [TDD] [position 2]
Token: BEGIN_PARAGRAPH
Token: TEXT
       paragraph: [Test Driven Development (Desenvolvimento Dirigido por
Teste)] [position 0]
Token: END_PARAGRAPH
Token: CONCEPT
```

```
       concept: [JUnit] [position 2]
Token: BEGIN_PARAGRAPH
Token: TEXT
       paragraph: [Framework para implementação de scripts de teste em
Java] [position 0]
Token: END_PARAGRAPH
Token: CONCEPT
       concept: [Teste Unitário] [position 2]
Token: BEGIN_PARAGRAPH
Token: TEXT
       paragraph: [Teste realizado em uma única unidade. A unidade deve
estar isolada das demais.] [position 0]
Token: END_PARAGRAPH
Token: END_CONCEPTS
Token: BEGIN_LIST
Token: LIST_ITEM
       item: [JUnit] [position 0]
Token: LIST_ITEM
       item: [Mockito] [position 0]
Token: LIST_ITEM
       item: [Jest] [position 0]
Token: LIST_ITEM
       item: [NUnit] [position 0]
Token: END_LIST
Token: BEGIN_SOURCE
Token: SOURCE_CODE
       source: [int x = 0, y = 1;] [position 0]
Token: SOURCE_CODE
       source: [] [position 0]
Token: SOURCE_CODE
       source: [if (x > y)] [position 0]
Token: SOURCE_CODE
       source: [  x++;] [position 0]
Token: SOURCE_CODE
       source: [else] [position 0]
Token: SOURCE_CODE
       source: [  y++;] [position 0]
Token: END_SOURCE
Token: IMAGE
       width: [100] [position 89]
       alt: ["Logotipo da UNIRIO"] [position 68]
       url:
[http://www.unirio.br/++theme++unirio-teste/img/logo_unirio.png] [position
5]
       height: [50] [position 93]
Token: VIDEO
       width: [500] [position 49]
       url: [https://www.youtube.com/embed/tgbNymZ7vqY] [position 7]
       height: [300] [position 53]
```

Fonte: elaborada pela autora

4.5 ANALISADOR SEMÂNTICO

O Analisador Semântico é responsável por avaliar se a estrutura gerada faz sentido, pois mesmo que os *tokens* sejam válidos e o contexto esteja correto, a sua combinação pode ser semanticamente inválida. Na versão atual da LCML, dois casos são avaliados pelo Analisador Semântico, ambos associados ao comando **#question**. Um deles verifica se há

uma e apenas uma resposta marcada como correta na questão, ou seja, não é permitido que haja mais de uma resposta correta ou nenhuma resposta correta. O outro caso verifica se há pelo menos duas respostas para a questão, já que se há apenas uma ela será, necessariamente, a resposta correta e isso fará a questão perder o sentido. Se algum desses casos for identificado, será apresentada uma mensagem de erro.

4.6 GERADOR

Esse componente é responsável pela conversão da lista sintática gerada pelo Analisador Sintático (Figura 16) para o formato alvo desejado. O Gerador implementado neste trabalho converte os *tokens* dessa estrutura em um arquivo HTML gravado em disco. Entretanto, a arquitetura do compilador permite que esse gerador seja facilmente substituído por outro, possibilitando que a geração seja feita em outros formatos, como JSON ou LaTeX.

4.7 GERADOR DE HTML

Essa seção descreve o gerador HTML implementado neste trabalho e detalha como cada comando da LCML foi mapeado para o HTML. Essa descrição pode servir de ponto de partida para implementação de outros geradores.

4.7.1 Estilização

Como visto no Capítulo 3, a LCML não possui comandos para estilização do conteúdo, ou seja, não existem comandos para definição de cores, fontes ou outros estilos. Isso acontece porque o foco da LCML é a descrição dos elementos semânticos que compõem o conteúdo, porém, a estilização é um fator relevante para a apresentação do conteúdo para o usuário final. Para resolver esse problema, o gerador de HTML gera referências para classes CSS (*Cascade Stylesheet*) nos comandos HTML para permitir que o conteudista crie essas classes em um arquivo CSS e defina o estilo desejado para a apresentação. Além disso, cada tag HTML associada a um comando recebe um identificador único que também permite aplicar estilos a um comando específico (Tabela 4).

Tabela 4: Classes CSS e IDs atribuídos às tags HTML

LCML	Classe CSS	ID
#1	content_h1	header1, header2, header3, ...
#2	content_h2	
#3	content_h3	
#code	content_code	code1, code2, code3, ...
#list	content_list	list1, list2, list3, …
#concepts	content_concepts	concepts1, concepts2, concepts3, …
#table	content_table	table1, table2, table3, …
#img	content_img	img1, img2, img3, …
#video	content_video	video1, video2, video3, …
#scenario	content_scenario	scenario1, scenario2, scenario3, …
#question	content_question	question1, question2, question3, …
parágrafo	content_p	p1, p2, p3, …

Fonte:elaborada pela autora

4.7.2 Cabeçalhos

Os comandos **#1**, **#2** e **#3** são traduzidos diretamente para os comandos **h1**, **h2** e **h3** do HTML (Tabela 5). Conforme definido na Tabela 4, são geradas referências para as classes CSS dos cabeçalhos e um **id** para cada cabeçalho com uma numeração sequencial.

Tabela 5: Exemplos de conversão de cabeçalhos

LCML

```
#1 1. TDD
```

HTML

```
<h1 class="content_h1" id="header1">
1. TDD
</h1>
```

LCML

```
#2 1.1 Conceitos de TDD
```

HTML

```
<h2 class="content_h2" id="header2">
1.1 Conceitos de TDD
</h2>
```

LCML

```
#3 1.1.1 Teste Unitário
```

```
HTML

<h3 class="content_h3" id="header3">
1.1.1 Teste Unitário
</h3>
```

Fonte: elaborada pela autora

4.7.3 Códigos

O comando **#code** é traduzido utilizando as tags ** ** e **<code>** (Tabela 6). A tag ** ** é usada para definir uma área onde o código será apresentado, enquanto a tag **<code>** formata o conteúdo com uma fonte monoespaçada, que é mais adequada para apresentação de códigos. Além disso, para que o usuário visualize o código da mesma forma em que ele é escrito, é necessário substituir cada espaço em branco por **&nbps;** (tabulações são substituídas por quatro ** **) e acrescentar a tag **
** ao final de cada linha para forçar a quebra de linha.

Tabela 6: Exemplos de conversão de códigos

```
LCML

#code
x = x + 2;
#code
```

```
HTML

<div class="content_code" id="code1">
<code>
x = x + 2;<br>
</code>
</div>
```

```
LCML

#code
public static String getString(String key) {
   return rb != null ? rb.getString(key) : key;
}
#code
```

```
HTML

<div class="content_code" id="code2">
<code>
public static String getString(String key) {<br>
   return rb != null ? rb.getString(
```

```
key) : key;<br>
}<br>
</code>
</div>
```

Fonte: elaborada pela autora

4.7.4 Listas

O comando **#list** é traduzido utilizando as tags **<ul>** e **<li>** que montam listas não numeradas no HTML (Tabela 7). O marcador de cada item da lista é o marcador padrão usado pelo navegador.

Tabela 7: Exemplos de conversão de listas

LCML

```
#list
JUnit
Jest
#list
```

HTML

```
<ul  class="content_list" id="list1">
  <li>JUnit</li>
  <li>Jest</li>
</ul>
```

LCML

```
#list
Item número um da lista
Esse é o item dois da lista
Terceiro item da lista
#list
```

HTML

```
<ul  class="content_list" id="list2">
  <li>Item número um da lista</li>
  <li>Esse é o item dois da lista</li>
  <li>Terceiro item da lista</li>
</ul>
```

Fonte: elaborada pela autora

4.7.5 Conceitos

Os conceitos apresentados no comando **#concept** são traduzidos como parágrafos no HTML, ou seja, cada conceito é apresentado em uma tag **<p>** (Tabela 8). A explicação ou detalhamento de cada conceito é traduzido de acordo com o comando usado para esse detalhamento.

Tabela 8: Exemplos de conversão de conceitos

LCML

```
#concepts
- Conceito UM
Definição do conceito UM

- Conceito DOIS
Definição do conceito DOIS.
Continuação da definição do conceito DOIS

#concepts
```

HTML

```
<div class="content_concepts" id="concepts1">
  <p>Conceito UM</p>
  <p class="content_p" id="p1">Definição do conceito UM</p>
  <p>Conceito DOIS</p>
  <p class="content_p" id="p2">Definição do conceito DOIS.Continuação da
definição do conceito DOIS</p>
</div>
```

LCML

```
#concepts
- TDD
Test Driven Development (Desenvolvimento Dirigido por Teste)

- JUnit
Framework para implementação de scripts de teste em Java

- Teste Unitário
Teste realizado em uma única unidade.
A unidade deve estar isolada das demais.

#concepts
```

HTML

```
<div class="content_concepts" id="concepts2">
  <p>TDD</p>
  <p class="content_p" id="p3">Test Driven Development (Desenvolvimento
Dirigido por Teste)</p>
  <p>JUnit</p>
  <p class="content_p" id="p4">Framework para implementação de scripts de
teste em Java</p>
  <p>Teste Unitário</p>
```

```
  <p class="content_p" id="p5">Teste realizado em uma única unidade.A
unidade deve estar isolada das demais.</p>
</div>
```

Fonte: elaborada pela autora

4.7.6 Tabelas

O comando **#table** é traduzido utilizando as tags **<table>**, **<th>**, **<tr>** e **<td>** (Tabela 9). Caso a tabela tenha uma borda, será criado um estilo CSS a ser aplicado exclusivamente nessa tabela (usando o ID da tabela). Para o alinhamento das colunas será usado o atributo **align** da tag **<td>**.

Tabela 9: Exemplos de conversão de tabelas

LCML

```
#table
Cabeçalho da Coluna 1 | Cabeçalho da Coluna 2
Linha 1, coluna 1 | Linha 1, coluna 2
Linha 2, coluna 1 | Linha 2, coluna 2
#table
```

HTML

```
<table class='content_table' id='table1'>
  <tr>
    <th align='center'>Cabeçalho da Coluna 1</th>
    <th align='center'>Cabeçalho da Coluna 2</th>
  </tr>
  <tr>
    <td align='left'>Linha 1, coluna 1</td>
    <td align='left'>Linha 1, coluna 2</td>
  </tr>
  <tr>
    <td align='left'>Linha 2, coluna 1</td>
    <td align='left'>Linha 2, coluna 2</td>
  </tr>
</table>
```

LCML

```
#table border
<Produto | =Categoria | >Preço
Mouse | A | 32,30
Notebook | B | 2727,48
Teclado | C | 128,65
#table
```

HTML

```
<style>
#table2 { border: 1px solid black;
          border-collapse: collapse;
          padding:3px; }
```

```
</style>
<table class='content_table' id='table2'>
  <tr>
    <th align='center'>Produto</th>
    <th align='center'>Categoria</th>
    <th align='center'>Preço</th>
  </tr>
  <tr>
    <td align='left'>Mouse</td>
    <td align='center'>A</td>
    <td align='right'>32,30</td>
  </tr>
  <tr>
    <td align='left'>Notebook</td>
    <td align='center'>B</td>
    <td align='right'>2727,48</td>
  </tr>
  <tr>
    <td align='left'>Teclado</td>
    <td align='center'>C</td>
    <td align='right'>128,65</td>
  </tr>
</table>
```

Fonte: elaborada pela autora

4.7.7 Imagens

O comando **#img** é traduzido usando as tags ** ** e **<img>** (Tabela 10). A tag ** ** é usada para definir uma área para a apresentação da imagem, de forma que duas imagens consecutivas não fiquem uma ao lado da outra, e sim uma abaixo da outra. A tag **<img>** vai criar a imagem propriamente dita. Para o texto alternativo é usado o parâmetro **alt** e para as dimensões são usados estilos CSS.

Tabela 10: Exemplos de conversão de imagens

LCML

```
#img http://site.com/foto1.jpg
```

HTML

```
<div class="content_img" id="img1">
   <img src="http://site.com/foto1.jpg">
</div>
```

LCML

```
#img http://site.com/foto2.jpg 500 300
```

HTML

```
<div class="content_img" id="img2">
   <img src="http://site.com/foto2.jpg"
        style="width:500px;height:300px;">
</div>
```

LCML

```
#img http://site.com/foto3.jpg "texto alternativo"
```

HTML

```
<div class="content_img" id="img3">
   <img src="http://site.com/foto3.jpg"
        alt="texto alternativo">
</div>
```

LCML

```
#img http://site.com/foto4.jpg "texto alternativo" 500 300
```

HTML

```
<div class="content_img" id="img4">
   <img src="http://site.com/foto4.jpg"
        alt="texto alternativo"
        style="width:500px;height:300px;">
</div>
```

Fonte: elaborada pela autora

4.7.8 Vídeos

O comando **#video** é traduzido usando as tags ** **, **<video>** e **<iframe>** (Tabela 11). A tag ** ** é usada para definir uma área para a apresentação do vídeo, de forma que dois vídeos consecutivos não fiquem um ao lado do outro, e sim um abaixo do outro. Se o vídeo for do Youtube, a tradução utilizará a tag <**iframe**>, já que o servidor aceita fazer conexão apenas com ela, mas se for qualquer outro tipo de vídeo a tag utilizada será <**video**>.

Tabela 11: Exemplos de conversão de vídeos

LCML

```
#video https://www.site.com/movie.mp4 500 300
```

HTML

```
<div class="content_video" id="video1">
  <video controls width="500" height="300">
    <source src="https://www.site.com/movie.mp4"/>
  </video>
</div>
```

LCML

```
#video https://www.youtube.com/embed/tgbNymZ7vqY
```

HTML

```
<div class="content_video" id="video2">
  <iframe src="https://www.youtube.com/embed/tgbNymZ7vqY"></iframe>
</div>
```

LCML

```
#video https://www.youtube.com/embed/tgbNymZ7vqY 500 300
```

HTML

```
<div class="content_video" id="video3">
  <iframe width="500"
          height="300"
          src="https://www.youtube.com/embed/tgbNymZ7vqY">
  </iframe>
</div>
```

Fonte:elaborada pela autora

4.7.9 Cenários de Uso

O comando **#scenario** é traduzido usando os comandos ** ** e **<h3>** (Tabela 12). A tag ** ** cria uma área para apresentação do cenário, enquanto a tag **<h3>** representa cada passo do cenário de uso. Além disso, como o cenário representa um passo a passo para executar uma tarefa, uma numeração sequencial 1, 2, 3, etc. é acrescentada em cada passo do cenário. A explicação ou detalhamento de cada passo é traduzido de acordo com o comando usado para esse detalhamento.

Tabela 12: Exemplos de conversão de cenários de uso

LCML

```
#scenario
- Passo um
Texto do passo um

- Passo dois
- Passo três
#scenario
```

HTML

```
<div class="content_scenario" id="scenario1">
```

```
  <h3>1.Passo um\</h3>
  <p class="content_p" id="p6">Texto do passo um</p>
  <h3>2.Passo dois</h3>
  <h3>3.Passo três</h3>
</div>
```

LCML

```
#scenario
- Passo um
Parágrafo do passo um

- Passo dois
Parágrafo do passo dois

Outro parágrafo do passo dois

- Passo três
#img http://www.unirio.br/++theme++unirio-teste/img/logo_unirio.png
#scenario
```

HTML

```
<div class="content_scenario" id="scenario2">
  <h3>1.Passo um</h3>
  <p class="content_p" id="p7">Parágrafo do passo um</p>
  <h3>2.Passo dois</h3>
  <p class="content_p" id="p8">Parágrafo do passo dois</p>
  <p class="content_p" id="p9">Outro parágrafo do passo dois</p>
  <h3>3.Passo três</h3>
</div>
```

Fonte: elaborada pela autora

4.7.10 Questões

O comando **#question** é traduzido usando as tags ** **, **<input>**, **<label>** e **<button>** (Tabela 13). A tag ** ** cria uma área para apresentação da questão. Cada resposta da questão é traduzida usando a tag **<input>** para criar o botão de rádio relativo aquela resposta e a tag **<label>** é usada para criar o texto da resposta. O botão de rádio de cada resposta possui um **id** único e o atributo **value** é usado para marcar a resposta correta. Por fim, a tag **<button>** é usada para criar um botão que o usuário poderá acionar para corrigir a questão. Se a questão não possuir uma função de verificação, então será gerado um código Javascript que será executado pelo botão de correção da questão. Esse código simplesmente avisa ao usuário se ele acertou ou não a resposta. Se a questão possuir uma função de verificação, então essa função será chamada quando o usuário pressionar o botão de

correção da questão. Nesse caso, fica a cargo do conteudista implementar o código Javascript da função e disponibilizá-lo no HTML.

Tabela 13: Exemplos de conversão de questões

LCML

```
#question
Quanto é 2 x 3?

() Cinco
(*) Seis
() Oito
#question
```

HTML

```
<p class="content_p" id="p10">Quanto é 2 x 3?</p>
<input type='radio' id='q1_a1' name='q1' value=' '>
<label for='q1_a1'> Cinco</label><br>
<input type='radio' id='q1_a2' name='q1' value='*'>
<label for='q1_a2'> Seis</label><br>
<input type='radio' id='q1_a3' name='q1' value=' '>
<label for='q1_a3'> Oito</label><br>
<br>
<button>Verificar</button>
<br>
```

LCML

```
#question assess
Primeiro parágrafo do enunciado

Segundo parágrafo do enunciado

() Primeira resposta
Parágrafo da primeira resposta

(*) Segunda resposta
#img http://site.com.br/image.jpg
() Terceira resposta
#code
if (x > y)
  x++;
#code
#question
```

HTML

```
<p class="content_p" id="p11">Primeiro parágrafo do enunciado</p>
<p class="content_p" id="p12">Segundo parágrafo do enunciado</p>
<input type='radio' id='q2_a1' name='q2' value=' '>
<label for='q2_a1'> Primeira resposta</label>
```

```
<br>
<p class="content_p" id="p13">Parágrafo da primeira resposta</p>
<input type='radio' id='q2_a2' name='q2' value='*'>
<label for='q2_a2'> Segunda resposta</label><br>
<div class="content_img" id="img5">
<img src="http://site.com.br/image.jpg">
</div>
<br>
<input type='radio' id='q2_a3' name='q2' value=' '>
<label for='q2_a3'> Terceira resposta</label>
<br>
<div class="content_code" id="code3">
<code>
if (x > y)<br>
  x++;<br>
</code>
</div>
<br>
<button onclick='assess(q2)'>Verificar</button>
<br>
```

Fonte: elaborada pela autora

4.7.11 Parágrafos

Parágrafos são traduzidos usando a tag **<p>** (Tabela 14). Caso haja quebras de linha, elas serão implementadas com a tag **
**.

Tabela 14: Exemplos de conversão de parágrafos

LCML

```
Essa é a linha inicial de um parágrafo.
Essa frase vai continuar a linha anterior.
Fim do parágrafo.
```

HTML

```
<p class="content_p" id="p14">Essa é a linha inicial de um parágrafo.Essa
frase vai continuar a linha anterior.Fim do parágrafo.</p>
```

LCML

```
Segundo parágrafo.
Essa frase continua a linha anterior.
Ao fim dessa linha tem uma quebra.\
Nova linha no mesmo parágrafo com uma quebra no final.\
\
Última linha do parágrafo.
```

HTML

```
<p class="content_p" id="p15">Segundo parágrafo.Essa frase continua a linha
anterior.Ao fim dessa linha tem uma quebra.<br>Nova linha no mesmo parágrafo
com uma quebra no final.<br><br>
```

```
Última linha do parágrafo.</p>
```

Fonte: elaborada pela autora

4.7.12 Formatação de Textos

Os diversos comandos de formatação de texto da LCML (link, negrito, itálico, sublinhado, tachado, fonte aumentada, subscrito e sobrescrito) são traduzidos usando tags básicas do HTML como **<a>**, **<b>**, **<i>**, **<u>**, **<del>**, **<sub>** e **<sup>**. Para a fonte aumentada é usado um estilo CSS (Tabela 15).

Tabela 15: Exemplos de conversão de formatação de textos

LCML
`Clique [aqui](http://google.com) para acessar o Google!`
HTML
`Clique <a href='http://google.com'>aqui</a> para acessar o Google!`
LCML
`Esse texto vai ser apresentado em **negrito**`
HTML
`Esse texto vai ser apresentado em <b>negrito</b>`
LCML
`Esse texto vai ser apresentado em *itálico*`
HTML
`Esse texto vai ser apresentado em <i>itálico</i>`
LCML
`Esse texto vai ser apresentado __sublinhado__`
HTML
`Esse texto vai ser apresentado <u>sublinhado</u>`
LCML
`Esse texto vai ser apresentado ~~tachado~~`
HTML

Esse texto vai ser apresentado **~~**tachado**~~**
LCML Esse texto vai ser apresentado com **++**Fonte Aumentada**++**
HTML Esse texto vai ser apresentado com **<span** style='font-size:larger;'**>** Fonte Aumentada****
LCML H**~**2**~**O
HTML H**_{**2**}**O
LCML X**^**i+1**^**
HTML X**^{**i+1**}**

Fonte: elaborada pela autora

No próximo capítulo serão apresentados conteúdos de uma trilha de aprendizagem criados com a LCML, que serão utilizados como prova de conceito do projeto dessa linguagem.

5. CONTEÚDO DE APRENDIZAGEM PARA TDD

TDD é um assunto bastante extenso e não faz parte do escopo deste trabalho definir conteúdos que explorem todo esse tema. O objetivo é mostrar a utilidade da LCML como linguagem de definição de conteúdos de aprendizagem e comparar o esforço de desenvolver esses conteúdos em LCML e em HTML. Assim, foi desenvolvida uma trilha de aprendizagem de TDD abordando os seus conceitos básicos e os conteúdos dessa trilha foram escritos em LCML pela autora deste trabalho. Ao final, serão feitas algumas comparações entre esse código LCML e o código HTML gerado pelo compilador.

5.1 TRILHA DE INTRODUÇÃO AO TDD

A trilha de aprendizagem de TDD está dividida em 3 assuntos e os assuntos estão organizados em um ou dois conteúdos, seguindo o modelo de trilha de aprendizagem proposto por Campos (2023):

- Principais Conceitos do TDD
    - Conteúdo 1.1: conceitos básicos do TDD
    - Conteúdo 1.2: exercícios de fixação
- JUnit
    - Conteúdo 2.1: funções básicas do JUnit
    - Conteúdo 2.2: exercícios de fixação
- Aplicação Prática do TDD
    - Conteúdo 3: cenário de uso do TDD
- Avaliação Final do Aprendizado
    - Conteúdo 4: exercícios de avaliação da prática de TDD

Nessa divisão é possível enxergar os três tipos de conhecimento da documentação do desenvolvedor, ou seja, conceitos de domínio, fatos de execução e cenários de uso. O primeiro assunto equivale aos conceitos de domínio relacionados a TDD, o segundo equivale aos fatos de execução do *framework* de testes JUnit e o terceiro assunto aborda um cenário de utilização do TDD. As próximas seções mostram como ficaram os conteúdos de cada um desses conteúdos na linguagem LCML e uma visão das páginas HTML geradas. Os códigos HTML gerados pelo compilador para cada um dos conteúdos estão listados no Apêndice B.

5.1.1 Principais Conceitos do TDD

Esse assunto trata dos principais conceitos referentes ao TDD e, para ele, foram definidos dois conteúdos: um para apresentar esses conceitos (usando o comando **#concepts** da LCML) e o outro para definir alguns exercícios de fixação sobre esses conceitos (usando o comando **#question**). A Figura 20 apresenta o primeiro conteúdo em LCML e a Figura 21 apresenta a página HTML gerada para esse conteúdo. Da mesma forma, a Figura 22 apresenta o LCML do conteúdo dois e a Figura 23 apresenta a página HTML gerada para esse conteúdo.

Figura 20: Código LCML dos conceitos do TDD

```
#1 1. Principais Conceitos do TDD

#2 1.1 Conceitos Básicos do TDD
#concepts
- ++TDD++
*Test Driven Development (Desenvolvimento Dirigido por Teste)* é um modo de
desenvolvimento em que os testes são desenvolvidos antes do código e são
feitas constantes refatorações até que a lógica seja totalmente
implementada.\
Dessa forma, há uma suíte de testes exaustiva e ela determina como o código
deve ser escrito. Geralmente praticado no XP (Extreme Programming).

- ++Testes de programador++
São os tipos de testes realizados em TDD. Eles são testes que definem como
o código deve funcionar. Diferente de testes de unidade, que verificam se o
código está funcionando como deveria.

- ++TDD Traffic Light++
Metáfora criada por William Wake. Ela define 3 situações durante a
utilização de TDD:

#list
**Luz amarela**: erros de compilação
**Luz vermelha**: falha nos testes
**Luz verde**: sucesso nos testes
#list
- ++Duplicação++
Um dos principais problemas do código. Pode ser explícita ou sutil. Alguns
exemplos são:

#list
Conjunto de linhas repetidas na mesma ordem em diferentes partes do código.
Solução: colocar as linhas que se repetem em uma função e substituir as
demais ocorrências pela chamada da função.
Retorno constante em uma função. Forma sutil de duplicação muito comum em
TDD, já que ao implementar o mínimo possível, muitas vezes será atribuído
um retorno constante para que o teste passe. Solução: implementar lógica de
programação.
Mesma informação mantida em locais diferentes. Solução: selecionar a melhor
implementação e remover a outra ocorrência.
#list
- ++Refatoração++
É uma forma de alterar a lógica do código sem alterar a sua saída com o
objetivo de melhorar a sua estrutura interna. Deve ser feita nas seguintes
```

```
circunstâncias:

#list
Duplicação
Código e/ou seu objetivo não está claro
Code Smells (indícios de que há algo errado no código)
#list
#concepts
```

Fonte: elaborada pela autora

Figura 21: Página HTML gerada para os conceitos do TDD

1. Principais Conceitos do TDD

1.1 Conceitos Básicos do TDD

TDD

Test Driven Development (Desenvolvimento Dirigido por Teste) é um modo de desenvolvimento em que os testes são desenvolvidos antes do código e são feitas constantes refatorações até que a lógica seja totalmente implementada.
Dessa forma, há uma suíte de testes exaustiva e ela determina como o código deve ser escrito. Geralmente praticado no XP (Extreme Programming).

Testes de programador

São os tipos de testes realizados em TDD. Eles são testes que definem como o código deve funcionar. Diferente de testes de unidade, que verificam se o código está funcionando como deveria.

TDD Traffic Light

Metáfora criada por William Wake. Ela define 3 situações durante a utilização de TDD:

- **Luz amarela**: erros de compilação
- **Luz vermelha**: falha nos testes
- **Luz verde**: sucesso nos testes

Duplicação

Um dos principais problemas do código. Pode ser explícita ou sutil. Alguns exemplos são:

- Conjunto de linhas repetidas na mesma ordem em diferentes partes do código. Solução: colocar as linhas que se repetem em uma função e substituir as demais ocorrências pela chamada da função.
- Retorno constante em uma função. Forma sutil de duplicação muito comum em TDD, já que ao implementar o mínimo possível, muitas vezes será atribuído um retorno constante para que o teste passe. Solução: implementar lógica de programação.
- Mesma informação mantida em locais diferentes. Solução: selecionar a melhor implementação e remover a outra ocorrência.

Refatoração

É uma forma de alterar a lógica do código sem alterar a sua saída com o objetivo de melhorar a sua estrutura interna. Deve ser feita nas seguintes circunstâncias:

- Duplicação
- Código e/ou seu objetivo não está claro
- Code Smells (indícios de que há algo errado no código)

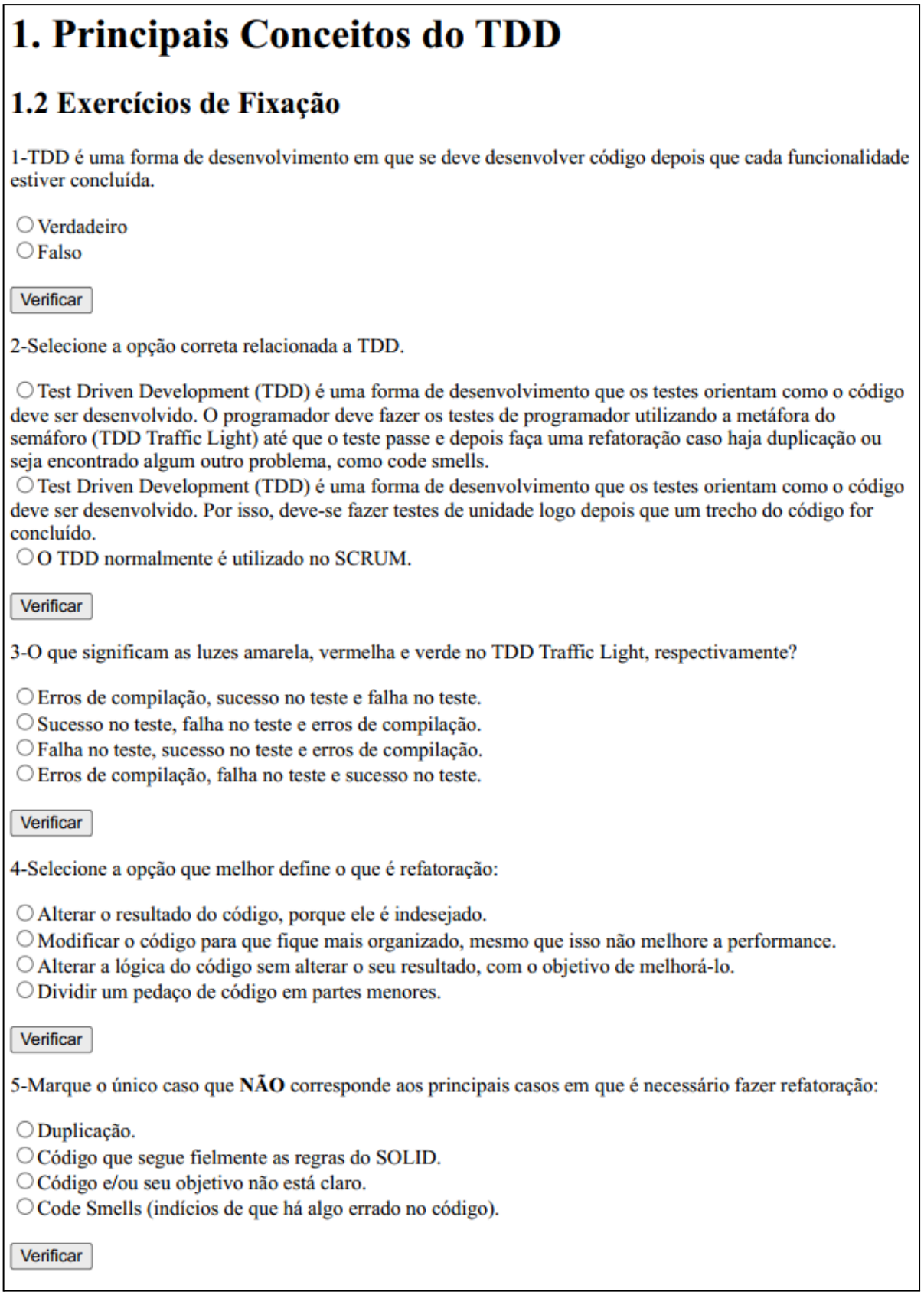
Fonte: elaborada pela autora

Figura 22: Código LCML dos exercícios de fixação dos conceitos do TDD

```
#1 1. Principais Conceitos do TDD
#2 1.2 Exercícios de Fixação
#question
1-TDD é uma forma de desenvolvimento em que se deve desenvolver código
depois que cada funcionalidade estiver concluída.

()Verdadeiro
(*)Falso
#question

#question
2-Selecione a opção correta relacionada a TDD.

(*)Test Driven Development (TDD) é uma forma de desenvolvimento que os
testes orientam como o código deve ser desenvolvido. O programador deve
fazer os testes de programador utilizando a metáfora do semáforo (TDD
Traffic Light) até que o teste passe e depois faça uma refatoração caso
haja duplicação ou seja encontrado algum outro problema, como code smells.
()Test Driven Development (TDD) é uma forma de desenvolvimento que os
testes orientam como o código deve ser desenvolvido. Por isso, deve-se
fazer testes de unidade logo depois que um trecho do código for concluído.
()O TDD normalmente é utilizado no SCRUM.
#question

#question
3-O que significam as luzes amarela, vermelha e verde no TDD Traffic Light,
respectivamente?

()Erros de compilação, sucesso no teste e falha no teste.
()Sucesso no teste, falha no teste e erros de compilação.
()Falha no teste, sucesso no teste e erros de compilação.
(*)Erros de compilação, falha no teste e sucesso no teste.
#question

#question
4-Selecione a opção que melhor define o que é refatoração:

()Alterar o resultado do código, porque ele é indesejado.
()Modificar o código para que fique mais organizado, mesmo que isso não
melhore a performance.
(*)Alterar a lógica do código sem alterar o seu resultado, com o objetivo
de melhorá-lo.
()Dividir um pedaço de código em partes menores.
#question

#question
5-Marque o único caso que **NÃO** corresponde aos principais casos em que é
necessário fazer refatoração:

()Duplicação.
(*)Código que segue fielmente as regras do SOLID.
()Código e/ou seu objetivo não está claro.
()Code Smells (indícios de que há algo errado no código).
#question
```

Fonte: elaborada pela autora

Figura 23: Página HTML gerada para os exercícios de fixação dos conceitos do TDD



1. Principais Conceitos do TDD

1.2 Exercícios de Fixação

1-TDD é uma forma de desenvolvimento em que se deve desenvolver código depois que cada funcionalidade estiver concluída.

- ○ Verdadeiro
- ○ Falso

Verificar

2-Selecione a opção correta relacionada a TDD.

- ○ Test Driven Development (TDD) é uma forma de desenvolvimento que os testes orientam como o código deve ser desenvolvido. O programador deve fazer os testes de programador utilizando a metáfora do semáforo (TDD Traffic Light) até que o teste passe e depois faça uma refatoração caso haja duplicação ou seja encontrado algum outro problema, como code smells.
- ○ Test Driven Development (TDD) é uma forma de desenvolvimento que os testes orientam como o código deve ser desenvolvido. Por isso, deve-se fazer testes de unidade logo depois que um trecho do código for concluído.
- ○ O TDD normalmente é utilizado no SCRUM.

Verificar

3-O que significam as luzes amarela, vermelha e verde no TDD Traffic Light, respectivamente?

- ○ Erros de compilação, sucesso no teste e falha no teste.
- ○ Sucesso no teste, falha no teste e erros de compilação.
- ○ Falha no teste, sucesso no teste e erros de compilação.
- ○ Erros de compilação, falha no teste e sucesso no teste.

Verificar

4-Selecione a opção que melhor define o que é refatoração:

- ○ Alterar o resultado do código, porque ele é indesejado.
- ○ Modificar o código para que fique mais organizado, mesmo que isso não melhore a performance.
- ○ Alterar a lógica do código sem alterar o seu resultado, com o objetivo de melhorá-lo.
- ○ Dividir um pedaço de código em partes menores.

Verificar

5-Marque o único caso que **NÃO** corresponde aos principais casos em que é necessário fazer refatoração:

- ○ Duplicação.
- ○ Código que segue fielmente as regras do SOLID.
- ○ Código e/ou seu objetivo não está claro.
- ○ Code Smells (indícios de que há algo errado no código).

Verificar

Fonte: elaborada pela autora

5.1.2 JUnit

Nesse assunto é abordado o uso do *framework* JUnit[6] e foram definidos dois conteúdos. Um deles usa o comando **#concepts** para mostrar as principais funções do *framework* com as definições dos parâmetros e qual o seu comportamento com exemplos. O outro aborda exercícios de fixação sobre esse tema, usando o comando **#question**. A Figura 24 apresenta o primeiro conteúdo em LCML e a Figura 25 apresenta parcialmente a página HTML gerada para esse conteúdo. Da mesma forma, a Figura 26 apresenta o LCML do conteúdo dos exercícios de fixação e a Figura 27 apresenta parcialmente a página HTML gerada para esse conteúdo.

Figura 24: Código LCML das funções básicas do JUnit

```
#1 2. JUnit

#2 2.1 Funções Básicas do JUnit

Em todas as funções a seguir é possível colocar uma mensagem de erro no
primeiro parâmetro além dos demais parâmetros da função. Contudo, esse
parâmetro é opcional.

#concepts
- ++1) fail++
Gera um erro no teste imediatamente. Útil para demonstrar que algo não
ocorreu como o esperado.\
Como exemplo, considere um teste que deveria gerar uma exceção se houvesse
uma divisão por zero. Se ele não entrar no bloco catch, isso significa que
uma exceção não foi gerada.

#code
public void testDivisaoPorZero(){
  try{
    int x = 3/0;
    fail("A divisão por zero não gerou exceção.");
  } catch(ArithmeticException ex){

  }
}
#code
- ++2) assertTrue e assertFalse++
Se o método assertTrue for utilizado, o teste falhará se o resultado for
false, já o assertFalse falhará se o resultado for true.

#code
public void testListaVazia(){
  ArrayList<Integer> numeros = new ArrayList<Integer>();
  assertTrue(numeros.isEmpty());
  numeros.add(1);
  numeros.add(2);
  numeros.add(3);
  assertFalse("A lista está vazia", numeros.isEmpty());
```

[6] https://junit.org/

```
}
#code
- ++3) assertNull e assertNotNull++
Verificam se um valor é nulo ou não nulo.

#code
public void testValorNull(){
  ArrayList<Integer> numeros = new ArrayList<Integer>();
  String valorNulo = null;
  assertNull(valorNulo);
  assertNotNull("Uma lista vazia deve ter valor não nulo",
numeros.isEmpty());
}
#code
- ++4) assertSame e assertNotSame++
Verifica se os dois objetos colocados nos parâmetros se referem ao mesmo
objeto. Também pode receber uma mensagem de erro no primeiro parâmetro que
corresponde a mensagem de erro que será exibida se o teste falhar.

#code
public void testObjetosIguais(){
  Movie m1 = new Movie("Star Wars");
  Movie m2 = new Movie("Star Trek");
  ArrayList<Movie> movies = new ArrayList<Movie>();
  movies.add(m1);
  assertNotSame("Esses filmes são iguais.", m2, movies.get(0));
  assertSame(m1, movies.get(0));
}
#code
- ++5) assertEquals++
Verifica se os valores passados pelos parâmetros são iguais.

#code
public void testValoresIguais(){
  int x = 2+4;
  int y = 3+3;
  assertEquals("Os valores não são iguais", x, y);
}
#code
#concepts
```

Fonte: elaborada pela autora

Figura 25: Página HTML gerada para as funções básicas do JUnit

2. JUnit

2.1 Funções Básicas do JUnit

Em todas as funções a seguir é possível colocar uma mensagem de erro no primeiro parâmetro além dos demais parâmetros da função. Contudo, esse parâmetro é opcional.

1) fail

Gera um erro no teste imediatamente. Útil para demonstrar que algo não ocorreu como o esperado.
Como exemplo, considere um teste que deveria gerar uma exceção se houvesse uma divisão por zero. Se ele não entrar no bloco catch, isso significa que uma exceção não foi gerada.

```
public void testDivisaoPorZero(){
  try{
    int x = 3/0;
    fail("A divisão por zero não gerou exceção.");
  } catch(ArithmeticException ex){

  }
}
```

2) assertTrue e assertFalse

Se o método assertTrue for utilizado, o teste falhará se o resultado for false, já o assertFalse falhará se o resultado for true.

```
public void testListaVazia(){
  ArrayList numeros = new ArrayList();
  assertTrue(numeros.isEmpty());
  numeros.add(1);
  numeros.add(2);
  numeros.add(3);
  assertFalse("A lista está vazia", numeros.isEmpty());
}
```

3) assertNull e assertNotNull

Verificam se um valor é nulo ou não nulo.

```
public void testValorNull(){
  ArrayList numeros = new ArrayList();
  String valorNulo = null;
  assertNull(valorNulo);
  assertNotNull("Uma lista vazia deve ter valor não nulo", numeros.isEmpty());
}
```

4) assertSame e assertNotSame

Verifica se os dois objetos colocados nos parâmetros se referem ao mesmo objeto. Também pode receber uma mensagem de erro no primeiro parâmetro que corresponde a mensagem de erro que será exibida se o teste falhar.

```
public void testObjetosIguais(){
  Movie m1 = new Movie("Star Wars");
```

Fonte: elaborada pela autora

Figura 26: Código LCML dos exercícios de fixação de JUnit

```
#1 2. JUnit

#2 2.2 Exercícios de Fixação
#question
1-Em todas as funções vistas de JUnit, é possível escrever opcionalmente a
mensagem de erro que aparecerá se o teste falhar adicionando no primeiro
parâmetro, além dos parâmetros já existentes na própria função.

(*) Verdadeiro
() Falso
#question

2-Sobre a função fail responda:

#question
a) Marque a opção que melhor define o que essa função faz:

() Verifica se uma exceção foi lançada.
() Verifica se um resultado é diferente do esperado.
(*) Gera uma falha no teste imediatamente. Útil para indicar que a função
não deu erro onde deveria dar.
() Verifica se o retorno foi undefined.
#question

#question
b) Marque a opção que demonstra a função sendo utilizada da maneira
correta:

() Código 1:
#code
  public void testDivisao(){
    try{
      int x = dividirNumeros(5,0);
      fail("A divisão por zero deveria gerar exceção, mas isso não
ocorreu.", x);
    }
      catch(ArithmeticException ex){
    }
  }
#code
(*) Código 2:
#code
  public void testDivisao(){
    try{
      int x = dividirNumeros(5,0);
      fail("A divisão por zero deveria gerar exceção, mas isso não
ocorreu.");
    }
      catch(ArithmeticException ex){
    }
  }
#code
() Código 3:
#code
  public void testDivisao(){
    try{
```

```
      fail("A divisão por zero deveria gerar exceção, mas isso não
ocorreu.", dividirNumeros(5,0));
    }
    catch(ArithmeticException ex){
    }
  }
#code
() Código 4:
#code
  public void testDivisao(){
    try{
      int x = dividirNumeros(5,0);
      System.out.println(x);
    } catch(ArithmeticException ex){
      fail("A divisão por zero deveria gerar exceção, mas isso não
ocorreu.");
    }
  }
#code
#question

3-Sobre as funções assertTrue e assertFalse responda:

#question
a) Marque a opção que melhor define o que essas funções fazem:

(*) A função assertTrue verifica se um valor é true e a assertFalse
verifica se é false.
() A função assertTrue verifica se um valor é false e a assertFalse
verifica se é true.
() A função assertTrue verifica se um valor é diferente de undefined e a
assertFalse verifica se é undefined.
() A função assertTrue verifica se um valor é undefined e a assertFalse
verifica se é diferente de undefined.
#question

#question
b) Marque a opção que demonstra alguma das funções sendo utilizada da
maneira correta:

() Código 1:
#code
public void testListaVazia(){
  ArrayList<String> nomes = new ArrayList<String>();
  nomes.add("Ana");
  nomes.add("Beatriz");
  nomes.add("Carlos");
  assertTrue("A lista está vazia", nomes.isEmpty());
}
#code
() Código 2:
#code
public void testListaVazia(){
  ArrayList<String> nomes = new ArrayList<String>();
  assertFalse(nomes.isEmpty());
}
#code
() Código 3:
#code
public void testListaVazia(){
```

```
  ArrayList<String> nomes = new ArrayList<String>();
  assertFalse(nomes.isEmpty());
  nomes.add("Ana");
  nomes.add("Beatriz");
  nomes.add("Carlos");
  assertTrue("A lista está vazia", nomes.isEmpty());
}
#code
(*) Código 4:
#code
public void testListaVazia(){
  ArrayList<String> nomes = new ArrayList<String>();
  assertTrue(nomes.isEmpty());
}
#code
#question

3- Sobre as funções assertNull e assertNotNull responda:

#question
a) Marque a opção que melhor define o que essas funções fazem:

() AssertNull verifica se o valor é diferente de null e assertNotNull
verifica se é null.
(*) AssertNull verifica se o valor é null e assertNotNull verifica se é
diferente de null.
() AssertNull verifica se o valor é um número e assertNotNull verifica se é
uma string.
() AssertNull verifica se o valor é uma string e assertNotNull verifica se
é um número.
#question
#question
b) Marque a opção que demonstra alguma das funções sendo utilizadas da
maneira correta:

() Código 1:
#code
public void testValorNull(){
  String s = null;
  assertNotNull(s);
}
#code
() Código 2:
#code
public void testValorNotNull(){
  int num = 5;
  assertNull(num);
}
#code
(*) Código 3:
#code
public void testValorNotNull(){
  ArrayList<Integer> numeros = new ArrayList<Integer>();
  assertNotNull(numeros);
}
#code
() Código 4:
#code
public void testValorNotNull(){
  ArrayList<Integer> numeros = new ArrayList<Integer>();
```

```
  assertNotNull(numeros, "Uma lista vazia não pode ter valor null");
}
#code
#question

4-Sobre as funções assertSame e assertNotSame responda:

#question
a) Marque a opção que melhor define o que essa função faz:

() Verifica se os valores colocados nos parâmetros são iguais.
() Verifica se os dois arrays colocados nos parâmetros são iguais.
()  Verifica se os dois números colocados nos parâmetros são iguais.
(*) Verifica se os dois objetos colocados nos parâmetros são o mesmo
objeto.
#question

#question
b) Marque a opção que demonstra a função sendo utilizada da maneira
correta:

(*) Código 1:
#code
public void testObjetosIguais(){
  Book b1 = new Book();
  Book b2 = new Book();
  ArrayList<Book> books = new ArrayList<Book>();
  books.add(b1);
  assertNotSame("Esses livros são iguais.", b2, books.get(0));
  assertSame(b1, books.get(0));
}
#code
() Código 2:
#code
public void testValoresIguais(){
  Pessoa p1 = new Pessoa();
  Pessoa p2 = new Pessoa();
  p1.setName("Ana");
  p1.setLastName("Smith");
  p2.setName("Ana");
  p2.setLastName("Torres");

  assertNotSame(p1.getLastName(), p2.getLastName());
  assertSame(p1.getName(), p2.getName());
}
#code
() Código 3:
#code
public void testValoresIguais(){
  int soma = 3+3;

  assertSame(soma, 6);
}
#code
() Código 4:
#code
public void testObjetosIguais(){
  ArrayList<Integer> numeros = new ArrayList<Integer>();
  ArrayList<String> nomes = new ArrayList<String>();
  numeros.add(1);
```

```
  numeros.add(2);

  nomes.add("Ana");
  nomes.add("Alexandre");

  assertNotSame(numeros, nomes);
}
#code
#question

5- Sobre a função assertEquals responda:

#question
a) Marque a opção que melhor define o que essa função faz:

() Verifica se o valor é ímpar.
(*) Verifica se dois valores são iguais.
() Verifica se o valor é par.
() Verifica se o valor é null.
#question

#question
b) Marque a opção que demonstra a função sendo utilizada da maneira
correta:

() Código 1:
#code
public void testValoresIguais(){
  assertEquals(4);
}
#code
() Código 2:
#code
public void testValoresIguais(){
  assertEquals(3);
}
#code
() Código 3:
#code
public void testValoresIguais(){
  int soma = 1+3;

  assertEquals(soma, 4, "Os valores são diferentes.");
}
#code
(*) Código 4:
#code
public void testValoresIguais(){
  int soma = 1+3;

  assertEquals(soma, 4);
}
#code
#question
```

Fonte: elaborada pela autora

Figura 27: Página HTML gerada para os exercícios de fixação do JUnit

2. JUnit

2.2 Exercícios de Fixação

1-Em todas as funções vistas de JUnit, é possível escrever opcionalmente a mensagem de erro que aparecerá se o teste falhar adicionando no primeiro parâmetro, além dos parâmetros já existentes na própria função.

- ○ Verdadeiro
- ○ Falso

Verificar

2-Sobre a função fail responda:

a) Marque a opção que melhor define o que essa função faz:

- ○ Verifica se uma exceção foi lançada.
- ○ Verifica se um resultado é diferente do esperado.
- ○ Gera uma falha no teste imediatamente. Útil para indicar que a função não deu erro onde deveria dar.
- ○ Verifica se o retorno foi undefined.

Verificar

b) Marque a opção que demonstra a função sendo utilizada da maneira correta:

○ Código 1:

```
public void testDivisao(){
  try{
    int x = dividirNumeros(5,0);
    fail("A divisão por zero deveria gerar exceção, mas isso não ocorreu.", x);
  }
 catch(ArithmeticException ex){
  }
}
```

○ Código 2:

```
public void testDivisao(){
  try{
    int x = dividirNumeros(5,0);
    fail("A divisão por zero deveria gerar exceção, mas isso não ocorreu.");
  }
 catch(ArithmeticException ex){
  }
}
```

○ Código 3:

```
public void testDivisao(){
  try{
    fail("A divisão por zero deveria gerar exceção, mas isso não ocorreu.", dividirNumeros(5,0));
  }
  catch(ArithmeticException ex){
  }
}
```

○ Código 4:

```
public void testDivisao(){
  try{
```

Fonte: elaborada pela autora

5.1.3 Aplicação Prática do TDD

O terceiro assunto da trilha trata da demonstração da aplicação prática do TDD em um cenário de uso, que é formado por apenas um conteúdo. O principal comando desse conteúdo é o **#scenario**, já que o objetivo é apresentar um passo a passo do uso dessa metodologia de desenvolvimento. A Figura 28 apresenta o conteúdo em LCML e a Figura 29 apresenta parcialmente a página HTML gerada para esse conteúdo.

Figura 28: Código LCML da aplicação prática do TDD

```
#1 3. Aplicação Prática do TDD

#scenario
Esse exemplo consiste em desenvolver algo que armazene uma lista de filmes
e permita que se adicione filmes. Dessa forma, pode-se considerar os
seguintes testes iniciais:

#list
Teste 1: Uma lista vazia deve ter tamanho zero.
Teste 2: Adicionar um item em uma lista deve tornar o seu tamanho 1.
Teste 3: Adicionar dois itens em uma lista deve tornar o seu tamanho 2.
Teste 4: Verificar se um determinado filme está na lista de filmes, dando
uma resposta positiva se ele estiver e negativa se não estiver.
#list
- Criação do teste 1
O teste de verificação de uma lista vazia com tamanho zero fica da seguinte
forma:

#code
public void testListaVazia(){
  MovieList listaVazia = new MovieList();
  assertEquals("O tamanho da lista deve ser 0", 0, listaVazia.size());
}
#code
Esse teste gerará uma luz amarela, já que a classe MovieList. Então,
deve-se resolver isso da maneira mais simples possível, criando uma classe
e adicionando a função size com retorno zero:

#code
public class MovieList {
  public int size(){
    return 0;
  }
}
#code
Agora ficamos com um caso de luz verde. Em TDD, normalmente depois da luz
amarela, vem uma luz vermelha e depois a verde.\
Nesse caso, isso não ocorreu porque java é uma linguagem fortemente tipada,
então para resolver os erros de compilação uma função com tipo de retorno
int deve retornar algo.\
Em linguagens fracamente tipadas, seria possível não colocar retorno, o
teste falhar para depois atribuir o retorno correto.\
No caso do java, foi colocado o tipo mais simples de retorno no caso de um
número inteiro (zero), que acabou sendo o valor desejado no teste da
verificação se a lista está vazia.
```

```
Por enquanto, não é necessário fazer nenhuma refatoração.

- Criação do teste 2
Para adicionar um filme na lista de filmes, imagina-se que deve existir uma
classe que corresponda ao filme, que exista um método de adição na classe
MovieList e que o valor do tamanho da lista seja 1.\
Dessa forma, o teste fica da seguinte forma:

#code
public void testListaUmItem(){
  Movie starWars = new Movie();
  MovieList listaUmItem = new MovieList();
  listaUmItem.add(starWars);
  assertEquals("O tamanho da lista deve ser 1", 1, listaUmItem.size());
}
#code
Voltamos ao caso da luz amarela, já que a função add não existe da classe
MovieList e a classe Movie também não existe.\
Para resolver isso, adiciona-se o que falta:

#code
public class MovieList {
  public void add(Movie novoFilme){

  }
  public int size(){
    return 0;
  }
}
#code
#code
public class Movie {

}
#code
A luz amarela transformou-se em luz vermelha. Isso ocorreu porque a função
size está retornando o valor constante 0 e nesse teste é esperado que o
valor do retorno do size seja 1.

Para resolver esse problema não é possível trocar o valor do retorno da
função size de 0 para 1 porque o teste da linha vazia começará a falhar.\
Por isso, deve-se capturar o valor do tamanho em uma variável e retorná-la
na função size. Com isso, o código ficará assim:

#code
public class MovieList {
  private int qtdFilmes = 0;
  public void add(Movie novoFilme){
    this.qtdFilmes = 1;
  }
  public int size(){
    return this.qtdFilmes;
  }
}
#code
Por enquanto, não há duplicações, então, não é necessário fazer
refatoração.

- Criação do teste 3
Esse teste é semelhante ao anterior, porém serão adicionados dois filmes:
```

```
#code
public void testListaDoisItens(){
  Movie starWars = new Movie();
  Movie starTrek = new Movie();
  MovieList listaDoisItens = new MovieList();
  listaDoisItens.add(starWars);
  listaDoisItens.add(starTrek);
  assertEquals("O tamanho da lista deve ser 2", 2, listaDoisItens.size());
}
#code
```

Esse teste gerará luz vermelha, porque a função add atribui o valor 1 a variável que armazena a quantidade de filmes. Para isso, deve-se substituir um valor constante por um dinâmico:

```
#code
public class MovieList {
  private int qtdFilmes = 0;
  public void add(Movie novoFilme){
    this.qtdFilmes++;
  }
  public int size(){
    return this.qtdFilmes;
  }
}
#code
```

Chegamos a uma luz verde sem duplicações no código, mas com duplicações nos testes: instâncias repetidas da classe MovieList e dos filmes starTrek e starWars.\
Por isso, serão adicionadas variáveis com esses valores e uma função setup:

```
#code
private MovieList listaFilmes = null;
private Movie starWars = null;
private Movie starTrek = null;

protected void setUp(){
  listaFilmes = new MovieList();
  starWars = new Movie();
  starTrek = new Movie();
}
#code
```

Então, as instâncias locais de cada teste devem ser substituídas:

```
#code
public void testListaVazia(){
  assertEquals("O tamanho da lista deve ser 0", 0, listaFilmes.size());
}

public void testListaUmItem(){
  listaFilmes.add(starWars);
  assertEquals("O tamanho da lista deve ser 1", 1, listaFilmes.size());
}

public void testListaDoisItens(){
  listaFilmes.add(starWars);
  listaFilmes.add(starTrek);
  assertEquals("O tamanho da lista deve ser 2", 2, listaFilmes.size());
}
#code
```

- Criação do teste 4
O teste para verificar o conteúdo da lista fica assim:

```
#code
public void testConteudoLista(){
  listaFilmes.add(starWars);
  listaFilmes.add(starTrek);
  assertTrue("A lista deve conter o filme Star Wars",
listaFilmes.cointains(starWars));
  assertTrue("A lista deve conter o filme Star Trek",
listaFilmes.cointains(starTrek));
  assertFalse("A lista não deve conter o filme Stargate",
listaFilmes.cointains(stargate));
}
#code
```

Nesse momento, ficamos com uma luz amarela, já que a função contains não existe na classe MovieList. Para resolver isso, a classe MovieList fica da seguinte forma:

```
#code
  private int qtdFilmes = 0;
  public void add(Movie novoFilme){
    this.qtdFilmes++;
  }
  public boolean contains(Movie filme){
    return false;
  }
  public int size(){
    return this.qtdFilmes;
  }
}
#code
```

Agora a luz ficou vermelha. O teste falhou porque o retorno da função contains é sempre false e nos dois primeiros testes o retorno precisa ser true.

Note também que até o momento a classe não está mantendo os filmes passados no parâmetro da função add em nenhum lugar. É hora de refatorar a classe MovieList para que armazene esses dados:

```
#code
  private int qtdFilmes = 0;
  private Collection filmes = new ArrayList()
  public void add(Movie novoFilme){
    this.qtdFilmes++;
    filmes.add(novoFilme);
  }
  public boolean contains(Movie filme){
    return false;
  }
  public int size(){
    return this.qtdFilmes;
  }
}
#code
```

Os testes anteriores continuam passando. Antes de prosseguir para a função contains, deve-se remover a variável que armazena a quantidade de filmes pela função size da classe ArrayList do Java:

```
#code
```

```
  private Collection filmes = new ArrayList()
  public void add(Movie novoFilme){
    filmes.add(novoFilme);
  }
  public boolean contains(Movie filme){
    return false;
  }
  public int size(){
    return filmes.size();
  }
}
#code
Agora que os filmes estão sendo armazenados da maneira correta, podemos
voltar a nossa atenção para a função contains, que é o foco do teste
atual.\
Para transformar a luz vermelha em uma luz verde basta utilizar a função
contains da classe ArrayList:

#code
  private Collection filmes = new ArrayList()
  public void add(Movie novoFilme){
    filmes.add(novoFilme);
  }
  public boolean contains(Movie filme){
    return filmes.contains(filme);
  }
  public int size(){
    return filmes.size();
  }
}
#code
Os testes já estão funcionando, mas eles contém duplicação. Como é possível
ver, os testes testConteudoLista e testListaDoisItens começam adicionando
os filmes starWars e starTrek:

#code
private MovieList listaFilmes = null;
private Movie starWars = null;
private Movie starTrek = null;
private Movie stargate = null;

protected void setUp(){
  listaFilmes = new MovieList();
  starWars = new Movie();
  starTrek = new Movie();
  stargate = new Movie();
}

public void testListaVazia(){
  assertEquals("O tamanho da lista deve ser 0", 0, listaFilmes.size());
}

public void testListaUmItem(){
  listaFilmes.add(starWars);
  assertEquals("O tamanho da lista deve ser 1", 1, listaFilmes.size());
}

public void testListaDoisItens(){
  listaFilmes.add(starWars);
  listaFilmes.add(starTrek);
```

```
  assertEquals("O tamanho da lista deve ser 2", 2, listaFilmes.size());
}

public void testConteudoLista(){
  listaFilmes.add(starWars);
  listaFilmes.add(starTrek);
  assertTrue("A lista deve conter o filme Star Wars",
listaFilmes.cointains(starWars));
  assertTrue("A lista deve conter o filme Star Trek",
listaFilmes.cointains(starTrek));
  assertFalse("A lista não deve conter o filme Stargate",
listaFilmes.cointains(stargate));
}
#code
Para resolver essa duplicação, o melhor é criar uma classe separada para
esses dois testes, inserindo então esses dois filmes na função setup:

#code
private MovieList listaFilmes = null;
private Movie starWars = null;
private Movie starTrek = null;
private Movie stargate = null;

protected void setUp(){
  listaFilmes = new MovieList();
  starWars = new Movie();
  starTrek = new Movie();
  stargate = new Movie();
  listaFilmes.add(starWars);
  listaFilmes.add(starTrek);
}

public void testListaDoisItens(){
  assertEquals("O tamanho da lista deve ser 2", 2, listaFilmes.size());
}

public void testConteudoLista(){
  assertTrue("A lista deve conter o filme Star Wars",
listaFilmes.cointains(starWars));
  assertTrue("A lista deve conter o filme Star Trek",
listaFilmes.cointains(starTrek));
  assertFalse("A lista não deve conter o filme Stargate",
listaFilmes.cointains(stargate));
}
#code
Para melhorar ainda mais o nome dos testes anteriores, deve-se criar uma
classe separada para a lista vazia e outra para o teste de adicionar um
filme na lista.\
Elas ficariam, respectivamente, dessa forma:

#code
private MovieList listaFilmes = null;

protected void setUp(){
  listaFilmes = new MovieList();
}

public void testTamanho(){
  assertEquals("O tamanho da lista deve ser 0", 0, listaFilmes.size());
}
```

```
#code
#code
private MovieList listaFilmes = null;
private Movie starWars = null;

protected void setUp(){
  listaFilmes = new MovieList();
  starWars = new Movie();
}
public void testTamanho(){
  listaFilmes.add(starWars);
  assertEquals("O tamanho da lista deve ser 1", 1, listaFilmes.size());
}
#code
#scenario
```

Fonte: elaborada pela autora

Figura 29: Página HTML gerada para a aplicação prática do TDD

3. Aplicação Prática do TDD

Esse exemplo consiste em desenvolver algo que armazene uma lista de filmes e permita que se adicione filmes. Dessa forma, pode-se considerar os seguintes testes iniciais:

- Teste 1: Uma lista vazia deve ter tamanho zero.
- Teste 2: Adicionar um item em uma lista deve tornar o seu tamanho 1.
- Teste 3: Adicionar dois itens em uma lista deve tornar o seu tamanho 2.
- Teste 4: Verificar se um determinado filme está na lista de filmes, dando uma resposta positiva se ele estiver e negativa se não estiver.

1.Criação do teste 1

O teste de verificação de uma lista vazia com tamanho zero fica da seguinte forma:

```
public void testListaVazia(){
  MovieList listaVazia = new MovieList();
  assertEquals("O tamanho da lista deve ser 0", 0, listaVazia.size());
}
```

Esse teste gerará uma luz amarela, já que a classe MovieList. Então, deve-se resolver isso da maneira mais simples possível, criando uma classe e adicionando a função size com retorno zero:

```
public class MovieList {
  public int size(){
    return 0;
  }
}
```

Agora ficamos com um caso de luz verde. Em TDD, normalmente depois da luz amarela, vem uma luz vermelha e depois a verde.
Nesse caso, isso não ocorreu porque java é uma linguagem fortemente tipada, então para resolver os erros de compilação uma função com tipo de retorno int deve retornar algo.
Em linguagens fracamente tipadas, seria possível não colocar retorno, o teste falhar para depois atribuir o retorno correto.
No caso do java, foi colocado o tipo mais simples de retorno no caso de um número inteiro (zero), que acabou sendo o valor desejado no teste da verificação se a lista está vazia.

Por enquanto, não é necessário fazer nenhuma refatoração.

2.Criação do teste 2

Para adicionar um filme na lista de filmes, imagina-se que deve existir uma classe que corresponda ao filme, que exista um método de adição na classe MovieList e que o valor do tamanho da lista seja 1.
Dessa forma, o teste fica da seguinte forma:

```
public void testListaUmItem(){
  Movie starWars = new Movie();
  MovieList listaUmItem = new MovieList();
  listaUmItem.add(starWars);
  assertEquals("O tamanho da lista deve ser 1", 1, listaUmItem.size());
}
```

Voltamos ao caso da luz amarela, já que a função add não existe da classe MovieList e a classe Movie também não existe.
Para resolver isso, adiciona-se o que falta:

Fonte: elaborada pela autora

5.1.4 Avaliação Final do Aprendizado

O último assunto da trilha é formado por um único conteúdo que trata da avaliação final do aprendizado, onde o aluno vai responder questões sobre a criação de um código usando a metodologia do TDD. O principal comando desse conteúdo é o **#question**, já bastante utilizado nos conteúdos anteriores. A Figura 30 apresenta o conteúdo em LCML e a Figura 31 apresenta parcialmente a página HTML gerada para esse conteúdo.

Figura 30: Código LCML da avaliação final do aprendizado

```
#1 4. Avaliação Final sobre o uso de TDD

Deverá ser criada uma função para somar 2 números inteiros usando Java e
JUnit.\
Os testes mais simples a serem feitos nesse caso são:

#list
Teste 1: Soma de 0 + 0 deve ser igual a 0.
Teste 2: Soma de 0 + 1 deve ser igual a 1.
#list

#question
1-Qual deve ser a primeira ação a ser feita?

() Desenvolver o código da função de soma.
(*) Fazer o Teste 1 utilizando o JUnit.
#question

#question
2-Marque a opção que melhor descreve o Teste 1:

() Código 1:
#code
public void testSomaZeroMaisZero(){
  assertEquals(0+0, 0);
}
#code
() Código 2:
#code
public void testSomaZeroMaisZero(){
  Matematica m = new Matematica();
  int soma = m.somar("0","0");
  assertEquals(soma, 0);
}
#code
() Código 3:
#code
public void testSomaZeroMaisZero(){
  int soma = somar(0,0);
  assertEquals(soma, 0, "Soma com valor diferente do esperado.");
}
#code
(*) Código 4:
#code
public void testSomaZeroMaisZero(){
  int soma = somar(0,0);
```

```
  assertEquals(soma, 0);
}
#code
#question

#question
3-Qual é a luz do TDD Traffic Light atual e qual o motivo?

() Luz vermelha. A classe Matematica e a função somar ainda não existem.
() Luz vermelha. O resultado foi diferente de 0.
(*) Luz amarela. A classe Matematica e a função somar ainda não existem.
() Luz verde. O resultado foi igual a 0.
#question

#question
5-Assinale a alternativa que resolveria o erro mencionado na questão
anterior.

(*) Código 1:
#code
public class Matematica {
  public int somar (int num1, int num2){
    return 0;
  }
}
#code
() Código 2:
#code
public void testSomaZeroMaisZero(){
  Matematica m = new Matematica();
  int soma = m.somar(0,0);
  assertEquals(soma, 2);
}
#code
() Código 3:
#code
public class Matematica {

}
#code
() Código 4:
#code
public void testSomaZeroMaisZero(){
  int soma = somar(0,0);
  assertEquals(soma, 1);
}
#code
#question

#question
6-Qual a luz do TDD Traffic Light atual e qual o motivo, depois de corrigir
o erro anterior?

() Luz vermelha. Em todos os casos a luz vermelha sucede a luz amarela.
() Luz verde. Em todos os casos a luz vermelha sucede a luz amarela.
() Luz vermelha. O código não está dando o resultado esperado pelo teste.
(*) Luz verde. Apesar do fluxo comum ser luz vermelha após uma luz amarela,
nesse primeiro teste acabou sendo verde por causa da tipagem do Java.
#question
```

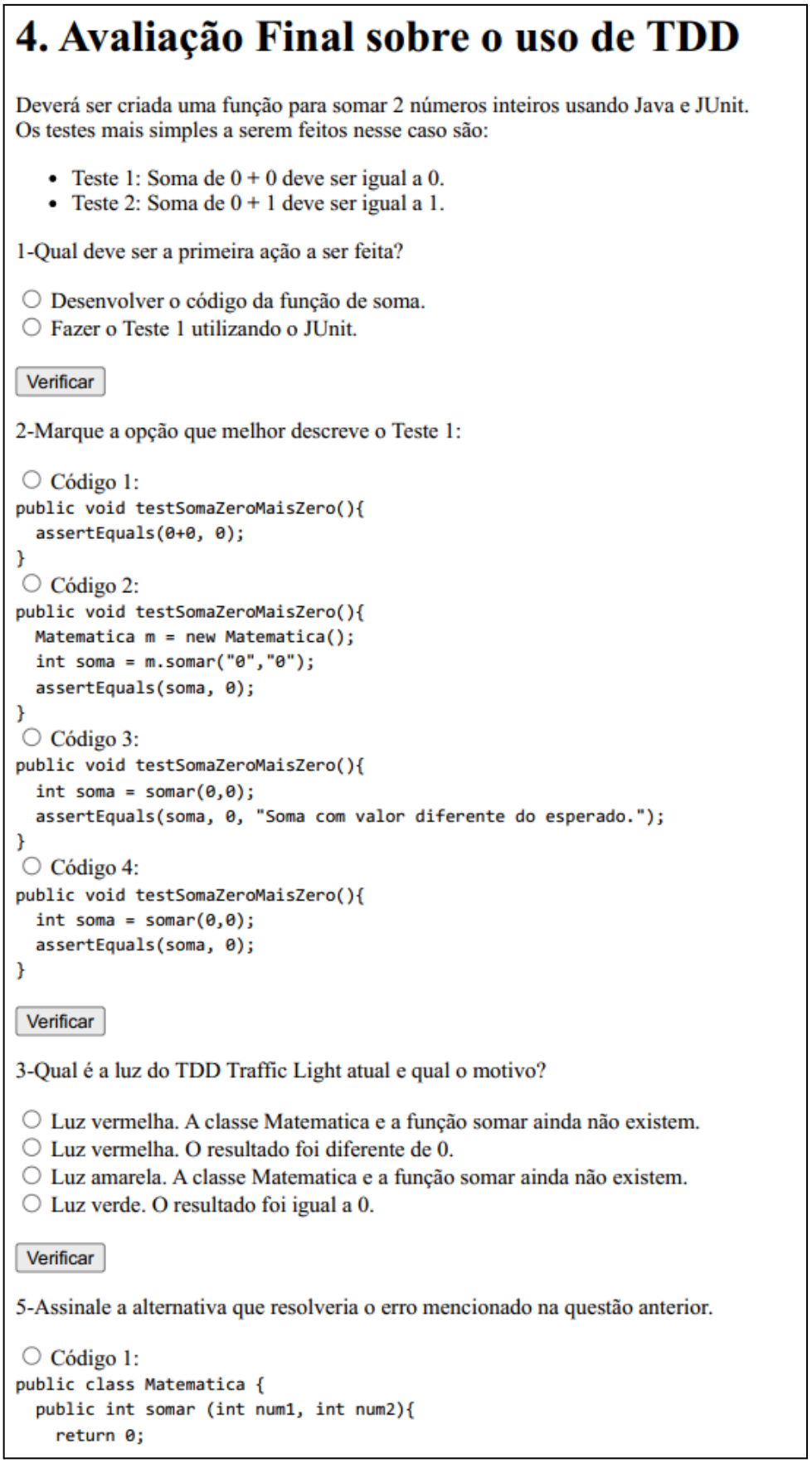
```
#question
7-Qual a próxima ação que deve ser feita?

() Refatorar o código.
(*) Ir para o próximo teste.
#question

#question
8-Marque a alternativa que corresponde a próxima etapa do processo de
desenvolvimento:

() Código 1:
#code
public void testSomaZeroMaisUm(){
  Matematica m = new Matematica();
  int soma = m.somar(2,1);
  assertEquals(soma, 3);
}
#code
(*) Código 2:
#code
public void testSomaZeroMaisUm(){
  Matematica m = new Matematica();
  int soma = m.somar(0,1);
  assertEquals(soma, 1);
}
#code
() Código 3:
#code
public class Matematica {
  public int somar (int num1, int num2){
    return num1+num2;
  }
}
#code
() Código 4:
#code
public void testSomaZeroMaisUm(){
  Matematica m = new Matematica();
  int soma = m.somar(0,1);
  assertEquals(soma, 2);
}
#code
#question

#question
9-Qual é a luz atual do TDD Traffic Light e qual o motivo?

() Luz amarela. Existe alguma função sendo chamada que ainda não existe.
(*) Luz vermelha. O resultado esperado é 1, mas a função somar está
retornando 0.
() Luz verde. O resultado do teste é o mesmo valor do resultado da função
somar.
() Luz amarela. Existe alguma classe sendo instanciada que ainda não
existe.
#question

#question
10-Marque a alternativa que corresponde com a próxima ação a ser feita:
```

```
(*) Código 1:
#code
public class Matematica {
  public int somar (int num1, int num2){
    return num1+num2;
  }
}
#code
() Código 2:
#code
public class Matematica {
  public int somar (int num1, int num2){
    return 1;
  }
}
#code
() Código 3:
#code
public void testSomaZeroMaisUm(){
  Matematica m = new Matematica();
  int soma = m.somar(0,1);
  assertEquals(soma, 0);
}
#code
() Código 4:
#code
public void testSomaZeroMaisUm(){
  Matematica m = new Matematica();
  int soma = m.somar(0,0);
  assertEquals(soma, 1);
}
#code
#question

#question
11-Qual é a luz do TDD Traffic Light atual e qual o motivo?

() Luz vermelha. O resultado da função ainda não o que se espera nos
testes.
() Luz amarela. Foi utilizada uma instânciação de classe que não existe.
(*) Luz verde. O resultado da função somar corresponde ao resultado
esperado dos testes feitos anteriormente.
()Luz amarela. Foi utilizada uma função que não existe.
#question

#question
12-Qual a próxima ação a ser feita?

() Nada. A função somar já funciona perfeitamente.
() Refatorar a classe Matematica.
() Implementar o código a seguir:
#code
private Matematica m;
public void testSomaZeroMaisZero(){
  Matematica m1 = new Matematica();
  int soma = m1.somar(0,0);
  assertEquals(soma, 0);
}
public void testSomaZeroMaisUm(){
  Matematica m2 = new Matematica();
```

```
  int soma = m2.somar(0,1);
  assertEquals(soma, 1);
}
#code
(*) Implementar o código a seguir:
#code
private Matematica m;
protected void setUp(){
  m = new Matematica();
}

public void testSomaZeroMaisZero(){
  int soma = this.m.somar(0,0);
  assertEquals(soma, 0);
}
public void testSomaZeroMaisUm(){
  int soma = this.m.somar(0,1);
  assertEquals(soma, 1);
}
#code
#question
```

Fonte: elaborada pela autora

Figura 31: Página HTML gerada para a avaliação final do aprendizado



4. Avaliação Final sobre o uso de TDD

Deverá ser criada uma função para somar 2 números inteiros usando Java e JUnit.
Os testes mais simples a serem feitos nesse caso são:

- Teste 1: Soma de 0 + 0 deve ser igual a 0.
- Teste 2: Soma de 0 + 1 deve ser igual a 1.

1-Qual deve ser a primeira ação a ser feita?

- ○ Desenvolver o código da função de soma.
- ○ Fazer o Teste 1 utilizando o JUnit.

Verificar

2-Marque a opção que melhor descreve o Teste 1:

○ Código 1:
```
public void testSomaZeroMaisZero(){
  assertEquals(0+0, 0);
}
```
○ Código 2:
```
public void testSomaZeroMaisZero(){
  Matematica m = new Matematica();
  int soma = m.somar("0","0");
  assertEquals(soma, 0);
}
```
○ Código 3:
```
public void testSomaZeroMaisZero(){
  int soma = somar(0,0);
  assertEquals(soma, 0, "Soma com valor diferente do esperado.");
}
```
○ Código 4:
```
public void testSomaZeroMaisZero(){
  int soma = somar(0,0);
  assertEquals(soma, 0);
}
```

Verificar

3-Qual é a luz do TDD Traffic Light atual e qual o motivo?

- ○ Luz vermelha. A classe Matematica e a função somar ainda não existem.
- ○ Luz vermelha. O resultado foi diferente de 0.
- ○ Luz amarela. A classe Matematica e a função somar ainda não existem.
- ○ Luz verde. O resultado foi igual a 0.

Verificar

5-Assinale a alternativa que resolveria o erro mencionado na questão anterior.

○ Código 1:
```
public class Matematica {
  public int somar (int num1, int num2){
    return 0;
```

Fonte: elaborada pela autora

5.2 CONCLUSÃO DA PROVA DE CONCEITO

Como demonstrado nas seções anteriores, foi possível desenvolver os conteúdos de uma trilha de aprendizagem usando a LCML. Essa prova de conceito apresenta indícios de que a LCML não só cumpre seu objetivo como uma linguagem para definição de conteúdos de aprendizagem, mas também pode ajudar a minimizar o esforço de desenvolvimento de conteúdos de uma trilha de aprendizagem. A partir dos dados da Tabela 16, que compara a quantidade de caracteres da LCML e do conteúdo convertido para HTML, é possível perceber que a quantidade é muito maior no formato HTML do que na LCML, o que significa que o esforço para desenvolver trilhas de aprendizagem utilizando LCML pode ser menor do que se o mesmo conteúdo fosse feito diretamente com HTML. Sem contar com o fato de que quem que vai desenvolver o conteúdo não precisará de conhecimentos em HTML para desenvolvê-la. Com isso, é possível inferir que a linguagem desenvolvida cumpre os requisitos de uma DSL, já que esta deve ser uma linguagem que resolva um problema específico e que seja simples tanto para pessoas que tenham conhecimento em desenvolvimento, nesse caso em HTML, quanto para pessoas leigas tecnicamente, mas que possuam conhecimento no desenvolvimento de conteúdos de aprendizagem. Por fim, essa prova de conceito revelou algumas melhorias que podem ser implementadas na LCML, mas elas serão discutidas no capítulo 6.

Tabela 16: Quantidade de caracteres do conteúdo em LCML x conteúdo em HTML

Conteúdo	Linguagem	Qtd. de caracteres	Variação em caracteres
1.1	LCML	1919	+136%
	HTML	2623	
1.2	LCML	1976	+260%
	HTML	5150	
2.1	LCML	2233	+170%
	HTML	3796	
2.2	LCML	6727	+263%
	HTML	17230	
3	LCML	10565	+160%
	HTML	16893	
4	LCML	5552	+277%
	HTML	15410	

Fonte: elaborada pela autora

6. CONCLUSÃO

Neste trabalho foi apresentada uma DSL, que recebeu o nome de LCML (*Learning Content Markup Language*), com comandos específicos voltados para facilitar a criação de conteúdos de aprendizagem. Também foi desenvolvido, em Java, um compilador que traduz códigos LCML em páginas HTML. Para avaliar a utilidade e a viabilidade da LCML e seu compilador foi elaborada uma trilha de aprendizagem sobre TDD chamada "Trilha de Introdução ao TDD" que foi dividida em quatro assuntos e seis conteúdos: Conceitos Básicos do TDD, Funções Básicas do JUnit, Aplicação Prática do TDD, Avaliação Final sobre o uso de TDD, além de dois conteúdos de Exercícios de Fixação, que são referentes aos dois primeiros mencionados. Todos esses conteúdos foram criados com a linguagem LCML e compilados para HTML, mostrando que a linguagem atende ao propósito para o qual ela foi projetada. Além disso, foi possível obter indícios de que a utilização da LCML no desenvolvimento de trilhas de aprendizagem pode reduzir o esforço de criação desses conteúdos, já que a quantidade de caracteres dos conteúdos em HTML são consistentemente maiores do que a quantidade dos mesmos conteúdos descritos em LCML.

Apesar do objetivo ter sido alcançado, existem melhorias que podem ser implementadas, tanto na linguagem LCML quanto no seu compilador. O compilador da LCML é um compilador de linha de comando, ou seja, não é possível visualizar os erros de compilação no momento da criação do conteúdo. Portanto, um trabalho futuro seria criar um editor que incorpore o compilador desenvolvido neste trabalho e que permita ao conteudista visualizar a apresentação final do conteúdo em tempo de edição. Outro trabalho futuro seria criar geradores para que os conteúdos em LCML fossem traduzidos para outras formas de apresentação como PDF ou LaTeX.

Do ponto de vista da LCML, algumas melhorias podem ser implementadas para que a linguagem aumente sua capacidade de definição de conteúdos:

- Permitir a inclusão de códigos-fonte, listas e imagens dentro de tabelas, pois atualmente as tabelas só aceitam textos.

- Permitir que os itens das listas sejam ordenados com números ou letras, pois atualmente os itens das listas não são ordenados, ou seja, são apresentados com *bullets*.
- Permitir que os conceitos possam ser ordenados com números ou letras. Atualmente, os conceitos são apresentados somente como textos
- Acrescentar novos comandos para estilização de textos, como a definição de cores e fontes.
- Definir um mecanismo que permita que um conteúdo referencie outro conteúdo, permitindo a criação de um hipertexto de conteúdos.
- Criar um comando específico para fatos de execução.

Por fim, para que esse projeto tenha um aproveitamento mais holístico, poderia ser desenvolvida uma infraestrutura que permitisse o desenvolvimento conjunto de trilhas de aprendizagem e de seus conteúdos. Seria um ambiente integrado que permitisse a criação dos conteúdos em LCML e de toda a estrutura da trilha de aprendizagem, permitindo referências entre esses conteúdos e trilhas, e que oferecesse a visualização e a navegação por esses elementos em tempo de edição, tornando mais fácil para o conteudista avaliar as mudanças durante o processo de desenvolvimento.

REFERÊNCIAS BIBLIOGRÁFICAS

ASTELS, D. **Test-Driven Development - A Practical Guide**, 1ª ed., Prentice Hall, 2003. <https://archive.org/details/testdrivendevelo00aste_0/page/n593/mode/1up?view=theater>. Acesso em: 21 dez 2023.

CAMPOS, J. P. P. **Um modelo para a definição de trilhas de aprendizagem aplicado ao ensino de computação**. Trabalho de conclusão de curso - Bacharelado em Sistemas de Informação, Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro, 2023.

FOWLER, M. **Domain Specific Languages**, 1ª ed., Addison-Wesley Professional, 2010.

GAMMA, E. et al. **Design Patterns: Elements of Reusable Object-Oriented Software**. [s.l.] Pearson Deutschland GmbH, 1995.

SILVA, J. G. T. **Plataforma Learning Curve: uma versão preliminar do suporte computacional para elaboração colaborativa da documentação do desenvolvedor**. Trabalho de conclusão de curso - Bacharelado em Sistemas de Informação, Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro, 2022.

STACK OVERFLOW. **Developer Survey Results 2023**. Disponível em: <https://survey.stackoverflow.co/2023/#section-learning-to-code-learning-how-to-code>. Acesso em: 19 dez 2023.

THAYER, K.; CHASINS, S. E.; KO, A. J. **A Theory of Robust API Knowledge**. ACM Trans. Comput. Educ. 21, 1, 32 p., 2021. https://doi.org/10.1145/3444945

W3C. **Extensible Markup Language (XML) 1.0 (Fifth Edition)**. Disponível em: <https://www.w3.org/TR/2008/REC-xml-20081126/#sec-notation>. Acesso em: 18 jan 2024.

YANG, F. **Learning Path Construction in e-Learning – What to Learn and How to Learn?** Doctoral—[s.l.] Durham University, 2013.

APÊNDICE A - DEFINIÇÃO FORMAL DA LCML

A BNF estendida adotada neste trabalho utiliza a sintaxe proposta pelo W3C (World Wide Web Consortium) (W3C, 2024). A proposta do W3C utiliza vários recursos existentes nas expressões regulares, o que facilita tanto a definição quanto a leitura das regras de produção pelos desenvolvedores atuais.

```
CONTENT ::=          LINE+ EOF ;
LINE ::=               HEADER1
                     | HEADER2
                     | HEADER3
                     | IMAGE
                     | VIDEO
                     | PARAGRAPH
                     | CODE
                     | LIST
                     | TABLE
                     | CONCEPTS
                     | SCENARIO
                     | QUESTION
                     | EMPTY_LINE;
HEADER1 ::=          '#1' TITLE NEWLINE ;
HEADER2 ::=          '#2' TITLE NEWLINE ;
HEADER3 ::=          '#3' TITLE NEWLINE ;
TITLE ::=            NON_EMPTY_STRING ;
IMAGE ::=            '#img' URL ALT_TEXT? (WIDTH HEIGHT)?
URL ::=              NON_EMPTY_STRING ;
ALT_TEXT ::=         QUOTE NON_EMPTY_STRING QUOTE ;
WIDTH ::=            NUMBER
HEIGHT ::=           NUMBER
VIDEO ::=            '#video' URL (WIDTH HEIGHT)?
PARAGRAPH ::=        FIRST_LINE NEWLINE (OTHER_LINE NEWLINE)* EMPTY_LINE ;
FIRST_LINE ::=       [^#-(\] .* ;
OTHER_LINE ::=       [^#-(] .* ;
CODE ::=             '#code' NEWLINE
                     SOURCE_CODE*
                     '#code' NEWLINE ;
SOURCE_CODE ::=      NON_EMPTY_STRING NEWLINE ;
LIST ::=             '#list' NEWLINE
                     LIST_ITEM*
                     '#list' NEWLINE ;
LIST_ITEM ::=        NON_EMPTY_STRING NEWLINE ;
CONCEPTS ::=         '#concepts' NEWLINE
                     CONCEPT*
                     '#concepts' NEWLINE ;
```

```
CONCEPT ::=           CONCEPT_ITEM NEWLINE
                      DESCRIPTION* ;
CONCEPT_ITEM ::=      ITEM_MARK CONCEPT_NAME ;
CONCEPT_NAME ::=      NON_EMPTY_STRING ;
SCENARIO ::=          '#scenario' NEWLINE
                      DESCRIPTION*
                      SCENARIO_ITEM*
                      '#scenario' NEWLINE ;
SCENARIO_ITEM ::=     ITEM_MARK SCENARIO_STEP NEWLINE
                      DESCRIPTION* ;
SCENARIO_STEP ::=     NON_EMPTY_STRING ;
TABLE ::=             '#table' 'border'? NEWLINE
                      TABLE_ROW+
                      '#table' NEWLINE ;
TABLE_ROW ::=         TABLE_CELL ( CELL_SEP TABLE_CELL )* NEWLINE ;
TABLE_CELL ::=        NON_EMPTY_STRING ;
QUESTION ::=          '#question' FNAME? NEWLINE
                      DESCRIPTION*
                      ANSWER+
                      '#question' NEWLINE ;
STATEMENT ::=         DESCRIPTION* ;
ANSWER ::=            ANSWER_MARK ANSWER_TEXT NEWLINE
                      DESCRIPTION* ;
ASNWER_MARK ::=       '(' '*'? ')' ;
ANSWER_TEXT ::=       NON_EMPTY_STRING ;
DESCRITPION ::=         IMAGE
                      | VIDEO
                      | PARAGRAPH
                      | CODE
                      | LIST
                      | TABLE
                      | EMPTY_LINE ;
CELL_SEP ::=          '|' ;
ITEM_MARK ::=         '-' ;
QUOTE ::=             '"' ;
NON_EMPTY_STRING ::=  .+ ;
NUMBER ::=            [0-9]+ ;
```

APÊNDICE B - PÁGINAS HTML DOS CONTEÚDOS DE TDD

Página HTML do conteúdo 1.1

```
<!DOCTYPE html>
<html>
<body>
<h1 class="content_h1" id="header1">1. Principais Conceitos do TDD</h1>
<h2 class="content_h2" id="header2">1.1 Conceitos Básicos do TDD</h2>
<div class="content_concepts" id="concepts1">
<p><span style='font-size:larger;'>TDD</span></p>
<p class="content_p" id="p1"><i>Test Driven Development (Desenvolvimento
Dirigido por Teste)</i> é um modo de desenvolvimento em que os testes são
desenvolvidos antes do código e são feitas constantes refatorações até que a
lógica seja totalmente implementada.<br>Dessa forma, há uma suíte de testes
exaustiva e ela determina como o código deve ser escrito. Geralmente
praticado no XP (Extreme Programming).</p>
<p><span style='font-size:larger;'>Testes de programador</span></p>
<p class="content_p" id="p2">São os tipos de testes realizados em TDD. Eles
são testes que definem como o código deve funcionar. Diferente de testes de
unidade, que verificam se o código está funcionando como deveria.</p>
<p><span style='font-size:larger;'>TDD Traffic Light</span></p>
<p class="content_p" id="p3">Metáfora criada por William Wake. Ela define 3
situações durante a utilização de TDD:</p>
<ul  class="content_list" id="list1">
<li><b>Luz amarela</b>: erros de compilação</li>
<li><b>Luz vermelha</b>: falha nos testes</li>
<li><b>Luz verde</b>: sucesso nos testes</li>
</ul>
<p><span style='font-size:larger;'>Duplicação</span></p>
<p class="content_p" id="p4">Um dos principais problemas do código. Pode ser
explícita ou sutil. Alguns exemplos são:</p>
<ul  class="content_list" id="list2">
<li>Conjunto de linhas repetidas na mesma ordem em diferentes partes do
código. Solução: colocar as linhas que se repetem em uma função e substituir
as demais ocorrências pela chamada da função.</li>
```

```
<li>Retorno constante em uma função. Forma sutil de duplicação muito comum em TDD, já que ao implementar o mínimo possível, muitas vezes será atribuído um retorno constante para que o teste passe. Solução: implementar lógica de programação.</li>
<li>Mesma informação mantida em locais diferentes. Solução: selecionar a melhor implementação e remover a outra ocorrência.</li>
</ul>
<p><span style='font-size:larger;'>Refatoração</span></p>
<p class="content_p" id="p5">É uma forma de alterar a lógica do código sem alterar a sua saída com o objetivo de melhorar a sua estrutura interna. Deve ser feita nas seguintes circunstâncias:</p>
<ul  class="content_list" id="list3">
<li>Duplicação</li>
<li>Código e/ou seu objetivo não está claro</li>
<li>Code Smells (indícios de que há algo errado no código)</li>
</ul>
</div>
</body>
</html>
```

Página HTML do conteúdo 1.2

```
<!DOCTYPE html>
<html>
<body>
<h1 class="content_h1" id="header1">1. Principais Conceitos do TDD</h1>
<h2 class="content_h2" id="header2">1.2 Exercícios de Fixação</h2>
<div class="content_question" id="question1">
<p class="content_p" id="p1">1-TDD é uma forma de desenvolvimento em que se deve desenvolver código depois que cada funcionalidade estiver concluída.</p>
<input type='radio' id='q1_a1' name='q1' value=' '><label for='q1_a1'>Verdadeiro</label><br>
<input type='radio' id='q1_a2' name='q1' value='*'><label for='q1_a2'>Falso</label><br>
<br><button onclick='(function (rg) {
let answers= document.getElementsByName(rg);
for (let i=0; i<answers.length; i++) if (answers[i].checked && answers[i].value==="*") { alert("Resposta correta!"); return; }
alert("Resposta incorreta :-(");})("q1");'
```

```
>Verificar
</button>
</div>
<div class="content_question" id="question2">
<p class="content_p" id="p2">2-Selecione a opção correta relacionada a
TDD.</p>
<input type='radio' id='q2_a1' name='q2' value='*'><label for='q2_a1'>Test
Driven Development (TDD) é uma forma de desenvolvimento que os testes
orientam como o código deve ser desenvolvido. O programador deve fazer os
testes de programador utilizando a metáfora do semáforo (TDD Traffic Light)
até que o teste passe e depois faça uma refatoração caso haja duplicação ou
seja encontrado algum outro problema, como code smells.</label><br>
<input type='radio' id='q2_a2' name='q2' value=' '><label for='q2_a2'>Test
Driven Development (TDD) é uma forma de desenvolvimento que os testes
orientam como o código deve ser desenvolvido. Por isso, deve-se fazer testes
de unidade logo depois que um trecho do código for concluído.</label><br>
<input type='radio' id='q2_a3' name='q2' value=' '><label for='q2_a3'>O TDD
normalmente é utilizado no SCRUM.</label><br>
<br><button onclick='(function (rg) {
let answers= document.getElementsByName(rg);
for (let i=0; i<answers.length; i++) if (answers[i].checked &&
answers[i].value==="*") { alert("Resposta correta!"); return; }
alert("Resposta incorreta :-(");})("q2");'
>Verificar
</button>
</div>
<div class="content_question" id="question3">
<p class="content_p" id="p3">3-O que significam as luzes amarela, vermelha e
verde no TDD Traffic Light, respectivamente?</p>
<input type='radio' id='q3_a1' name='q3' value=' '><label for='q3_a1'>Erros
de compilação, sucesso no teste e falha no teste.</label><br>
<input type='radio' id='q3_a2' name='q3' value=' '><label for='q3_a2'>Sucesso
no teste, falha no teste e erros de compilação.</label><br>
<input type='radio' id='q3_a3' name='q3' value=' '><label for='q3_a3'>Falha
no teste, sucesso no teste e erros de compilação.</label><br>
<input type='radio' id='q3_a4' name='q3' value='*'><label for='q3_a4'>Erros
de compilação, falha no teste e sucesso no teste.</label><br>
<br><button onclick='(function (rg) {
let answers= document.getElementsByName(rg);
for (let i=0; i<answers.length; i++) if (answers[i].checked &&
answers[i].value==="*") { alert("Resposta correta!"); return; }
```

```
alert("Resposta incorreta :-(");})("q3");'
>Verificar
</button>
</div>
<div class="content_question" id="question4">
<p class="content_p" id="p4">4-Selecione a opção que melhor define o que é
refatoração:</p>
<input type='radio' id='q4_a1' name='q4' value=' '><label for='q4_a1'>Alterar
o resultado do código, porque ele é indesejado.</label><br>
<input type='radio' id='q4_a2' name='q4' value=' '><label
for='q4_a2'>Modificar o código para que fique mais organizado, mesmo que isso
não melhore a performance.</label><br>
<input type='radio' id='q4_a3' name='q4' value='*'><label for='q4_a3'>Alterar
a lógica do código sem alterar o seu resultado, com o objetivo de
melhorá-lo.</label><br>
<input type='radio' id='q4_a4' name='q4' value=' '><label for='q4_a4'>Dividir
um pedaço de código em partes menores.</label><br>
<br><button onclick='(function (rg) {
let answers= document.getElementsByName(rg);
for (let i=0; i<answers.length; i++) if (answers[i].checked &&
answers[i].value==="*") { alert("Resposta correta!"); return; }
alert("Resposta incorreta :-(");})("q4");'
>Verificar
</button>
</div>
<div class="content_question" id="question5">
<p class="content_p" id="p5">5-Marque o único caso que <b>NÃO</b> corresponde
aos principais casos em que é necessário fazer refatoração:</p>
<input type='radio' id='q5_a1' name='q5' value=' '><label
for='q5_a1'>Duplicação.</label><br>
<input type='radio' id='q5_a2' name='q5' value='*'><label for='q5_a2'>Código
que segue fielmente as regras do SOLID.</label><br>
<input type='radio' id='q5_a3' name='q5' value=' '><label for='q5_a3'>Código
e/ou seu objetivo não está claro.</label><br>
<input type='radio' id='q5_a4' name='q5' value=' '><label for='q5_a4'>Code
Smells (indícios de que há algo errado no código).</label><br>
<br><button onclick='(function (rg) {
let answers= document.getElementsByName(rg);
for (let i=0; i<answers.length; i++) if (answers[i].checked &&
answers[i].value==="*") { alert("Resposta correta!"); return; }
alert("Resposta incorreta :-(");})("q5");'
```

```
>Verificar
</button>
</div>
</body>
</html>
```

Página HTML do conteúdo 2.1

```
<!DOCTYPE html>
<html>
<body>
<h1 class="content_h1" id="header1">2. JUnit</h1>
<h2 class="content_h2" id="header2">2.1 Funções Básicas do JUnit</h2>
<p class="content_p" id="p1">Em todas as funções a seguir é possível colocar
uma mensagem de erro no primeiro parâmetro além dos demais parâmetros da
função. Contudo, esse parâmetro é opcional.</p>
<div class="content_concepts" id="concepts1">
<p><span style='font-size:larger;'>1) fail</span></p>
<p class="content_p" id="p2">Gera um erro no teste imediatamente. Útil para
demonstrar que algo não ocorreu como o esperado.<br>Como exemplo, considere
um teste que deveria gerar uma exceção se houvesse uma divisão por zero. Se
ele não entrar no bloco catch, isso significa que uma exceção não foi
gerada.</p>
<div class="content_code" id="code1">
<code>
public void testDivisaoPorZero(){<br>
  try{<br>
    int x = 3/0;<br>
    fail("A divisão por zero não 
;gerou exceção.");<br>
  } catch(ArithmeticException ex){<br>
<br>
  }<br>
}<br>
</code>
</div>
<p><span style='font-size:larger;'>2) assertTrue e assertFalse</span></p>
```

Se o método assertTrue for utilizado, o teste falhará se o resultado for false, já o assertFalse falhará se o resultado for true.

<code>
public void testListaVazia(){
  ArrayList<Integer> numeros = new ArrayList<Integer>();
  assertTrue(numeros.isEmpty());
  numeros.add(1);
  numeros.add(2);
  numeros.add(3);
  assertFalse("A lista está vazia", numeros.isEmpty());
}
</code>

3) assertNull e assertNotNull
Verificam se um valor é nulo ou não nulo.

<code>
public void testValorNull(){
  ArrayList<Integer> numeros = new ArrayList<Integer>();
  String valorNulo = null;
  assertNull(valorNulo);
  assertNotNull("Uma lista vazia deve ter valor não nulo", numeros.isEmpty());
}
</code>

4) assertSame e assertNotSame
Verifica se os dois objetos colocados nos parâmetros se referem ao mesmo objeto. Também pode receber uma mensagem de erro no primeiro parâmetro que corresponde a mensagem de erro que será exibida se o teste falhar.

<code>
public void testObjetosIguais(){
  Movie m1 = new Movie("Star Wars");
  Movie m2 = new Movie("Star Trek");

```
  ArrayList<Movie> movies = new ArrayList<Movie
>();<br>
  movies.add(m1);<br>
  assertNotSame("Esses filmes são iguais.", m2,
 movies.get(0));<br>
  assertSame(m1, movies.get(0));<br>
}<br>
</code>
</div>
<p><span style='font-size:larger;'>5) assertEquals</span></p>
<p class="content_p" id="p6">Verifica se os valores passados pelos parâmetros
são iguais.</p>
<div class="content_code" id="code5">
<code>
public void testValoresIguais(){<br>
  int x = 2+4;<br>
  int y = 3+3;<br>
  assertEquals("Os valores não são iguais",&nbs
p;x, y);<br>
}<br>
</code>
</div>
</div>
</body>
</html>
```

Página HTML do conteúdo 2.2

```
<!DOCTYPE html>
<html>
<body>
<h1 class="content_h1" id="header1">2. JUnit</h1>
<h2 class="content_h2" id="header2">2.2 Exercícios de Fixação</h2>
<div class="content_question" id="question1">
<p class="content_p" id="p1">1-Em todas as funções vistas de JUnit, é
possível escrever opcionalmente a mensagem de erro que aparecerá se o teste
falhar adicionando no primeiro parâmetro, além dos parâmetros já existentes
na própria função.</p>
```

```
<input type='radio' id='q1_a1' name='q1' value='*'><label for='q1_a1'>
Verdadeiro</label><br>
<input type='radio' id='q1_a2' name='q1' value=' '><label for='q1_a2'>
Falso</label><br>
<br><button onclick='(function (rg) {
let answers= document.getElementsByName(rg);
for (let i=0; i<answers.length; i++) if (answers[i].checked &&
answers[i].value==="*") { alert("Resposta correta!"); return; }
alert("Resposta incorreta :-(");})("q1");'
>Verificar
</button>
</div>
<p class="content_p" id="p2">2-Sobre a função fail responda:</p>
<div class="content_question" id="question2">
<p class="content_p" id="p3">a) Marque a opção que melhor define o que essa
função faz:</p>
<input type='radio' id='q2_a1' name='q2' value=' '><label for='q2_a1'>
Verifica se uma exceção foi lançada.</label><br>
<input type='radio' id='q2_a2' name='q2' value=' '><label for='q2_a2'>
Verifica se um resultado é diferente do esperado.</label><br>
<input type='radio' id='q2_a3' name='q2' value='*'><label for='q2_a3'> Gera
uma falha no teste imediatamente. Útil para indicar que a função não deu erro
onde deveria dar.</label><br>
<input type='radio' id='q2_a4' name='q2' value=' '><label for='q2_a4'>
Verifica se o retorno foi undefined.</label><br>
<br><button onclick='(function (rg) {
let answers= document.getElementsByName(rg);
for (let i=0; i<answers.length; i++) if (answers[i].checked &&
answers[i].value==="*") { alert("Resposta correta!"); return; }
alert("Resposta incorreta :-(");})("q2");'
>Verificar
</button>
</div>
<div class="content_question" id="question3">
<p class="content_p" id="p4">b) Marque a opção que demonstra a função sendo
utilizada da maneira correta:</p>
<input type='radio' id='q3_a1' name='q3' value=' '><label for='q3_a1'> Código
1:</label><br>
<div class="content_code" id="code1">
<code>
  public void testDivisao(){<br>
```

```
    try{<br>
      int x = dividirNumeros(5,0
);<br>
      fail("A divisão por zero&n
bsp;deveria gerar exceção, mas isso não ocorreu
.", x);<br>
    }<br>
   catch(ArithmeticException ex){<br>
    }<br>
  }<br>
</code>
</div>
<input type='radio' id='q3_a2' name='q3' value='*'><label for='q3_a2'> Código
2:</label><br>
<div class="content_code" id="code2">
<code>
  public void testDivisao(){<br>
    try{<br>
      int x = dividirNumeros(5,0
);<br>
      fail("A divisão por zero&n
bsp;deveria gerar exceção, mas isso não ocorreu
.");<br>
    }<br>
   catch(ArithmeticException ex){<br>
    }<br>
  }<br>
</code>
</div>
<input type='radio' id='q3_a3' name='q3' value=' '><label for='q3_a3'> Código
3:</label><br>
<div class="content_code" id="code3">
<code>
  public void testDivisao(){<br>
    try{<br>
      fail("A divisão por zero&n
bsp;deveria gerar exceção, mas isso não ocorreu
.", dividirNumeros(5,0));<br>
    }<br>
   catch(ArithmeticException ex){<br>
    }<br>
```

```
  }<br>
</code>
</div>
<input type='radio' id='q3_a4' name='q3' value=' '><label for='q3_a4'> Código
4:</label><br>
<div class="content_code" id="code4">
<code>
  public void testDivisao(){<br>
    try{<br>
      int x = dividirNumeros(5,0
);<br>
      System.out.println(x);<br>
    } catch(ArithmeticException ex){<br>
      fail("A divisão por zero&n
bsp;deveria gerar exceção, mas isso não ocorreu
.");<br>
    }<br>
  }<br>
</code>
</div>
<br><button onclick='(function (rg) {
let answers= document.getElementsByName(rg);
for (let i=0; i<answers.length; i++) if (answers[i].checked &&
answers[i].value==="*") { alert("Resposta correta!"); return; }
alert("Resposta incorreta :-(");})("q3");'
>Verificar
</button>
</div>
<p class="content_p" id="p5">3-Sobre as funções assertTrue e assertFalse
responda:</p>
<div class="content_question" id="question4">
<p class="content_p" id="p6">a) Marque a opção que melhor define o que essas
funções fazem:</p>
<input type='radio' id='q4_a1' name='q4' value='*'><label for='q4_a1'> A
função assertTrue verifica se um valor é true e a assertFalse verifica se é
false.</label><br>
<input type='radio' id='q4_a2' name='q4' value=' '><label for='q4_a2'> A
função assertTrue verifica se um valor é false e a assertFalse verifica se é
true.</label><br>
```

```
<input type='radio' id='q4_a3' name='q4' value=' '><label for='q4_a3'> A
função assertTrue verifica se um valor é diferente de undefined e a
assertFalse verifica se é undefined.</label><br>
<input type='radio' id='q4_a4' name='q4' value=' '><label for='q4_a4'> A
função assertTrue verifica se um valor é undefined e a assertFalse verifica
se é diferente de undefined.</label><br>
<br><button onclick='(function (rg) {
let answers= document.getElementsByName(rg);
for (let i=0; i<answers.length; i++) if (answers[i].checked &&
answers[i].value==="*") { alert("Resposta correta!"); return; }
alert("Resposta incorreta :-(");})("q4");'
>Verificar
</button>
</div>
<div class="content_question" id="question5">
<p class="content_p" id="p7">b) Marque a opção que demonstra alguma das
funções sendo utilizada da maneira correta:</p>
<input type='radio' id='q5_a1' name='q5' value=' '><label for='q5_a1'> Código
1:</label><br>
<div class="content_code" id="code5">
<code>
public void testListaVazia(){<br>
  ArrayList<String> nomes = new ArrayList<Strin
g>();<br>
  nomes.add("Ana");<br>
  nomes.add("Beatriz");<br>
  nomes.add("Carlos");<br>
  assertTrue("A lista está vazia", nomes.isEmpt
y());<br>
}<br>
</code>
</div>
<input type='radio' id='q5_a2' name='q5' value=' '><label for='q5_a2'> Código
2:</label><br>
<div class="content_code" id="code6">
<code>
public void testListaVazia(){<br>
  ArrayList<String> nomes = new ArrayList<Strin
g>();<br>
  assertFalse(nomes.isEmpty());<br>
}<br>
```

```
</code>
</div>
<input type='radio' id='q5_a3' name='q5' value=' '><label for='q5_a3'> Código
3:</label><br>
<div class="content_code" id="code7">
<code>
public void testListaVazia(){<br>
  ArrayList<String> nomes = new ArrayList<Strin
g>();<br>
  assertFalse(nomes.isEmpty());<br>
  nomes.add("Ana");<br>
  nomes.add("Beatriz");<br>
  nomes.add("Carlos");<br>
  assertTrue("A lista está vazia", nomes.isEmpt
y());<br>
}<br>
</code>
</div>
<input type='radio' id='q5_a4' name='q5' value='*'><label for='q5_a4'> Código
4:</label><br>
<div class="content_code" id="code8">
<code>
public void testListaVazia(){<br>
  ArrayList<String> nomes = new ArrayList<Strin
g>();<br>
  assertTrue(nomes.isEmpty());<br>
}<br>
</code>
</div>
<br><button onclick='(function (rg) {
let answers= document.getElementsByName(rg);
for (let i=0; i<answers.length; i++) if (answers[i].checked &&
answers[i].value==="*") { alert("Resposta correta!"); return; }
alert("Resposta incorreta :-(");})("q5");'
>Verificar
</button>
</div>
<p class="content_p" id="p8">3- Sobre as funções assertNull e assertNotNull
responda:</p>
<div class="content_question" id="question6">
```

```
<p class="content_p" id="p9">a) Marque a opção que melhor define o que essas funções fazem:</p>
<input type='radio' id='q6_a1' name='q6' value=' '><label for='q6_a1'> AssertNull verifica se o valor é diferente de null e assertNotNull verifica se é null.</label><br>
<input type='radio' id='q6_a2' name='q6' value='*'><label for='q6_a2'> AssertNull verifica se o valor é null e assertNotNull verifica se é diferente de null.</label><br>
<input type='radio' id='q6_a3' name='q6' value=' '><label for='q6_a3'> AssertNull verifica se o valor é um número e assertNotNull verifica se é uma string.</label><br>
<input type='radio' id='q6_a4' name='q6' value=' '><label for='q6_a4'> AssertNull verifica se o valor é uma string e assertNotNull verifica se é um número.</label><br>
<br><button onclick='(function (rg) {
let answers= document.getElementsByName(rg);
for (let i=0; i<answers.length; i++) if (answers[i].checked &&
answers[i].value==="*") { alert("Resposta correta!"); return; }
alert("Resposta incorreta :-(");})("q6");'
>Verificar
</button>
</div>
<div class="content_question" id="question7">
<p class="content_p" id="p10">b) Marque a opção que demonstra alguma das funções sendo utilizadas da maneira correta:</p>
<input type='radio' id='q7_a1' name='q7' value=' '><label for='q7_a1'> Código 1:</label><br>
<div class="content_code" id="code9">
<code>
public void testValorNull(){<br>
  String s = null;<br>
  assertNotNull(s);<br>
}<br>
</code>
</div>
<input type='radio' id='q7_a2' name='q7' value=' '><label for='q7_a2'> Código 2:</label><br>
<div class="content_code" id="code10">
<code>
public void testValorNotNull(){<br>
  int num = 5;<br>
```

```
  assertNull(num);<br>
}<br>
</code>
</div>
<input type='radio' id='q7_a3' name='q7' value='*'><label for='q7_a3'> Código
3:</label><br>
<div class="content_code" id="code11">
<code>
public void testValorNotNull(){<br>
  ArrayList<Integer> numeros = new ArrayList<In
teger>();<br>
  assertNotNull(numeros);<br>
}<br>
</code>
</div>
<input type='radio' id='q7_a4' name='q7' value=' '><label for='q7_a4'> Código
4:</label><br>
<div class="content_code" id="code12">
<code>
public void testValorNotNull(){<br>
  ArrayList<Integer> numeros = new ArrayList<In
teger>();<br>
  assertNotNull(numeros, "Uma lista vazia não&n
bsp;pode ter valor null");<br>
}<br>
</code>
</div>
<br><button onclick='(function (rg) {
let answers= document.getElementsByName(rg);
for (let i=0; i<answers.length; i++) if (answers[i].checked &&
answers[i].value==="*") { alert("Resposta correta!"); return; }
alert("Resposta incorreta :-(");})("q7");'
>Verificar
</button>
</div>
<p class="content_p" id="p11">4-Sobre as funções assertSame e assertNotSame
responda:</p>
<div class="content_question" id="question8">
<p class="content_p" id="p12">a) Marque a opção que melhor define o que essa
função faz:</p>
```

```
<input type='radio' id='q8_a1' name='q8' value=' '><label for='q8_a1'>
Verifica se os valores colocados nos parâmetros são iguais.</label><br>
<input type='radio' id='q8_a2' name='q8' value=' '><label for='q8_a2'>
Verifica se os dois arrays colocados nos parâmetros são iguais.</label><br>
<input type='radio' id='q8_a3' name='q8' value=' '><label for='q8_a3'>
Verifica se os dois números colocados nos parâmetros são iguais.</label><br>
<input type='radio' id='q8_a4' name='q8' value='*'><label for='q8_a4'>
Verifica se os dois objetos colocados nos parâmetros são o mesmo
objeto.</label><br>
<br><button onclick='(function (rg) {
let answers= document.getElementsByName(rg);
for (let i=0; i<answers.length; i++) if (answers[i].checked &&
answers[i].value==="*") { alert("Resposta correta!"); return; }
alert("Resposta incorreta :-(");})("q8");'
>Verificar
</button>
</div>
<div class="content_question" id="question9">
<p class="content_p" id="p13">b) Marque a opção que demonstra a função sendo
utilizada da maneira correta:</p>
<input type='radio' id='q9_a1' name='q9' value='*'><label for='q9_a1'> Código
1:</label><br>
<div class="content_code" id="code13">
<code>
public void testObjetosIguais(){<br>
  Book b1 = new Book();<br>
  Book b2 = new Book();<br>
  ArrayList<Book> books = new ArrayList<Book>()
;<br>
  books.add(b1);<br>
  assertNotSame("Esses livros são iguais.", b2,
 books.get(0));<br>
  assertSame(b1, books.get(0));<br>
}<br>
</code>
</div>
<input type='radio' id='q9_a2' name='q9' value=' '><label for='q9_a2'> Código
2:</label><br>
<div class="content_code" id="code14">
<code>
public void testValoresIguais(){<br>
```

```
  Pessoa p1 = new Pessoa();<br>
  Pessoa p2 = new Pessoa();<br>
  p1.setName("Ana");<br>
  p1.setLastName("Smith");<br>
  p2.setName("Ana");<br>
  p2.setLastName("Torres");<br>
<br>
  assertNotSame(p1.getLastName(), p2.getLastName());<br>
  assertSame(p1.getName(), p2.getName());<br>
}<br>
</code>
</div>
<input type='radio' id='q9_a3' name='q9' value=' '><label for='q9_a3'> Código
3:</label><br>
<div class="content_code" id="code15">
<code>
public void testValoresIguais(){<br>
  int soma = 3+3;<br>
<br>
  assertSame(soma, 6);<br>
}<br>
</code>
</div>
<input type='radio' id='q9_a4' name='q9' value=' '><label for='q9_a4'> Código
4:</label><br>
<div class="content_code" id="code16">
<code>
public void testObjetosIguais(){<br>
  ArrayList<Integer> numeros = new ArrayList<In
teger>();<br>
  ArrayList<String> nomes = new ArrayList<Strin
g>();<br>
  numeros.add(1);<br>
  numeros.add(2);<br>
<br>
  nomes.add("Ana");<br>
  nomes.add("Alexandre");<br>
<br>
  assertNotSame(numeros, nomes);<br>
}<br>
</code>
```

```
</div>
<br><button onclick='(function (rg) {
let answers= document.getElementsByName(rg);
for (let i=0; i<answers.length; i++) if (answers[i].checked &&
answers[i].value==="*") { alert("Resposta correta!"); return; }
alert("Resposta incorreta :-(");})("q9");'
>Verificar
</button>
</div>
<p class="content_p" id="p14">5- Sobre a função assertEquals responda:</p>
<div class="content_question" id="question10">
<p class="content_p" id="p15">a) Marque a opção que melhor define o que essa
função faz:</p>
<input type='radio' id='q10_a1' name='q10' value=' '><label for='q10_a1'>
Verifica se o valor é ímpar.</label><br>
<input type='radio' id='q10_a2' name='q10' value='*'><label for='q10_a2'>
Verifica se dois valores são iguais.</label><br>
<input type='radio' id='q10_a3' name='q10' value=' '><label for='q10_a3'>
Verifica se o valor é par.</label><br>
<input type='radio' id='q10_a4' name='q10' value=' '><label for='q10_a4'>
Verifica se o valor é null.</label><br>
<br><button onclick='(function (rg) {
let answers= document.getElementsByName(rg);
for (let i=0; i<answers.length; i++) if (answers[i].checked &&
answers[i].value==="*") { alert("Resposta correta!"); return; }
alert("Resposta incorreta :-(");})("q10");'
>Verificar
</button>
</div>
<div class="content_question" id="question11">
<p class="content_p" id="p16">b) Marque a opção que demonstra a função sendo
utilizada da maneira correta:</p>
<input type='radio' id='q11_a1' name='q11' value=' '><label for='q11_a1'>
Código 1:</label><br>
<div class="content_code" id="code17">
<code>
public void testValoresIguais(){<br>
  assertEquals(4);<br>
}<br>
</code>
</div>
```

```
<input type='radio' id='q11_a2' name='q11' value=' '><label for='q11_a2'>
Código 2:</label><br>
<div class="content_code" id="code18">
<code>
public void testValoresIguais(){<br>
  assertEquals(3);<br>
}<br>
</code>
</div>
<input type='radio' id='q11_a3' name='q11' value=' '><label for='q11_a3'>
Código 3:</label><br>
<div class="content_code" id="code19">
<code>
public void testValoresIguais(){<br>
  int soma = 1+3;<br>
<br>
  assertEquals(soma, 4, "Os valores são di
ferentes.");<br>
}<br>
</code>
</div>
<input type='radio' id='q11_a4' name='q11' value='*'><label for='q11_a4'>
Código 4:</label><br>
<div class="content_code" id="code20">
<code>
public void testValoresIguais(){<br>
  int soma = 1+3;<br>
<br>
  assertEquals(soma, 4);<br>
}<br>
</code>
</div>
<br><button onclick='(function (rg) {
let answers= document.getElementsByName(rg);
for (let i=0; i<answers.length; i++) if (answers[i].checked &&
answers[i].value==="*") { alert("Resposta correta!"); return; }
alert("Resposta incorreta :-(");})("q11");'
>Verificar
</button>
</div>
</body>
```

```
</html>
```

Página HTML do conteúdo 3

```
<!DOCTYPE html>
<html>
<body>
<h1 class="content_h1" id="header1">3. Aplicação Prática do TDD</h1>
<div class="content_scenario" id="scenario1">
<p class="content_p" id="p1">Esse exemplo consiste em desenvolver algo que
armazene uma lista de filmes e permita que se adicione filmes. Dessa forma,
pode-se considerar os seguintes testes iniciais:</p>
<ul  class="content_list" id="list1">
<li>Teste 1: Uma lista vazia deve ter tamanho zero.</li>
<li>Teste 2: Adicionar um item em uma lista deve tornar o seu tamanho 1.</li>
<li>Teste 3: Adicionar dois itens em uma lista deve tornar o seu tamanho
2.</li>
<li>Teste 4: Verificar se um determinado filme está na lista de filmes, dando
uma resposta positiva se ele estiver e negativa se não estiver.</li>
</ul>
<h3>1.Criação do teste 1</h3>
<p class="content_p" id="p2">O teste de verificação de uma lista vazia com
tamanho zero fica da seguinte forma:</p>
<div class="content_code" id="code1">
<code>
public void testListaVazia(){<br>
  MovieList listaVazia = new MovieList();<br>
  assertEquals("O tamanho da lista deve se
r 0", 0, listaVazia.size());<br>
}<br>
</code>
</div>
<p class="content_p" id="p3">Esse teste gerará uma luz amarela, já que a
classe MovieList. Então, deve-se resolver isso da maneira mais simples
possível, criando uma classe e adicionando a função size com retorno
zero:</p>
<div class="content_code" id="code2">
<code>
public class MovieList {<br>
  public int size(){<br>
```

    return 0;
  }
}
</code>

Agora ficamos com um caso de luz verde. Em TDD, normalmente depois da luz amarela, vem uma luz vermelha e depois a verde.
Nesse caso, isso não ocorreu porque java é uma linguagem fortemente tipada, então para resolver os erros de compilação uma função com tipo de retorno int deve retornar algo.
Em linguagens fracamente tipadas, seria possível não colocar retorno, o teste falhar para depois atribuir o retorno correto.
No caso do java, foi colocado o tipo mais simples de retorno no caso de um número inteiro (zero), que acabou sendo o valor desejado no teste da verificação se a lista está vazia.
Por enquanto, não é necessário fazer nenhuma refatoração.
<h3>2.Criação do teste 2</h3>
Para adicionar um filme na lista de filmes, imagina-se que deve existir uma classe que corresponda ao filme, que exista um método de adição na classe MovieList e que o valor do tamanho da lista seja 1.
Dessa forma, o teste fica da seguinte forma:

<code>
public void testListaUmItem(){
  Movie starWars = new Movie();
  MovieList listaUmItem = new MovieList();
  listaUmItem.add(starWars);
  assertEquals("O tamanho da lista deve ser 1", 1, listaUmItem.size());
}
</code>

Voltamos ao caso da luz amarela, já que a função add não existe da classe MovieList e a classe Movie também não existe.
Para resolver isso, adiciona-se o que falta:

<code>
public class MovieList {
  public void add(Movie novoFilme){

  }

  public int size(){
    return 0;
  }
}
</code>

<code>
public class Movie {

}
</code>

A luz amarela transformou-se em luz vermelha. Isso ocorreu porque a função size está retornando o valor constante 0 e nesse teste é esperado que o valor do retorno do size seja 1.
Para resolver esse problema não é possível trocar o valor do retorno da função size de 0 para 1 porque o teste da linha vazia começará a falhar.
Por isso, deve-se capturar o valor do tamanho em uma variável e retorná-la na função size. Com isso, o código ficará assim:

<code>
public class MovieList {
  private int qtdFilmes = 0;
  public void add(Movie novoFilme){
    this.qtdFilmes = 1;
  }
  public int size(){
    return this.qtdFilmes;
  }
}
</code>

Por enquanto, não há duplicações, então, não é necessário fazer refatoração.
<h3>3.Criação do teste 3</h3>
Esse teste é semelhante ao anterior, porém serão adicionados dois filmes:

<code>

public void testListaDoisItens(){
  Movie starWars = new Movie();
  Movie starTrek = new Movie();
  MovieList listaDoisItens = new MovieList();
  listaDoisItens.add(starWars);
  listaDoisItens.add(starTrek);
  assertEquals("O tamanho da lista deve ser 2", 2, listaDoisItens.size());
}
</code>

Esse teste gerará luz vermelha, porque a função add atribui o valor 1 a variável que armazena a quantidade de filmes. Para isso, deve-se substituir um valor constante por um dinâmico:

<code>
public class MovieList {
  private int qtdFilmes = 0;
  public void add(Movie novoFilme){
    this.qtdFilmes++;
  }
  public int size(){
    return this.qtdFilmes;
  }
}
</code>

Chegamos a uma luz verde sem duplicações no código, mas com duplicações nos testes: instâncias repetidas da classe MovieList e dos filmes starTrek e starWars.
Por isso, serão adicionadas variáveis com esses valores e uma função setup:

<code>
private MovieList listaFilmes = null;
private Movie starWars = null;
private Movie starTrek = null;

protected void setUp(){
  listaFilmes = new MovieList();
  starWars = new Movie();

  starTrek = new Movie();
}
</code>

Então, as instâncias locais de cada teste devem ser substituídas:

<code>
public void testListaVazia(){
  assertEquals("O tamanho da lista deve ser 0", 0, listaFilmes.size());
}

public void testListaUmItem(){
  listaFilmes.add(starWars);
  assertEquals("O tamanho da lista deve ser 1", 1, listaFilmes.size());
}

public void testListaDoisItens(){
  listaFilmes.add(starWars);
  listaFilmes.add(starTrek);
  assertEquals("O tamanho da lista deve ser 2", 2, listaFilmes.size());
}
</code>

<h3>4.Criação do teste 4</h3>
O teste para verificar o conteúdo da lista fica assim:

<code>
public void testConteudoLista(){
  listaFilmes.add(starWars);
  listaFilmes.add(starTrek);
  assertTrue("A lista deve conter o filme Star Wars", listaFilmes.cointains(starWars));
  assertTrue("A lista deve conter o filme Star Trek", listaFilmes.cointains(starTrek));
  assertFalse("A lista não deve conter o filme Stargate", listaFilmes.cointains(stargate));

}
</code>

Nesse momento, ficamos com uma luz amarela, já que a função contains não existe na classe MovieList. Para resolver isso, a classe MovieList fica da seguinte forma:

<code>
  private int qtdFilmes = 0;
  public void add(Movie novoFilme){
    this.qtdFilmes++;
  }
  public boolean contains(Movie filme){
    return false;
  }
  public int size(){
    return this.qtdFilmes;
  }
}
</code>

Agora a luz ficou vermelha. O teste falhou porque o retorno da função contains é sempre false e nos dois primeiros testes o retorno precisa ser true.
Note também que até o momento a classe não está mantendo os filmes passados no parâmetro da função add em nenhum lugar. É hora de refatorar a classe MovieList para que armazene esses dados:

<code>
  private int qtdFilmes = 0;
  private Collection filmes = new ArrayLis
t()
  public void add(Movie novoFilme){
    this.qtdFilmes++;
    filmes.add(novoFilme);
  }
  public boolean contains(Movie filme){
    return false;
  }
  public int size(){
    return this.qtdFilmes;

  }
}
</code>

Os testes anteriores continuam passando. Antes de prosseguir para a função contains, deve-se remover a variável que armazena a quantidade de filmes pela função size da classe ArrayList do Java:

<code>
  private Collection filmes = new ArrayLis
t()
  public void add(Movie novoFilme){
    filmes.add(novoFilme);
  }
  public boolean contains(Movie filme){
    return false;
  }
  public int size(){
    return filmes.size();
  }
}
</code>

Agora que os filmes estão sendo armazenados da maneira correta, podemos voltar a nossa atenção para a função contains, que é o foco do teste atual.
Para transformar a luz vermelha em uma luz verde basta utilizar a função contains da classe ArrayList:

<code>
  private Collection filmes = new ArrayLis
t()
  public void add(Movie novoFilme){
    filmes.add(novoFilme);
  }
  public boolean contains(Movie filme){
    return filmes.contains(filme);
  }
  public int size(){
    return filmes.size();
  }
}

</code>

Os testes já estão funcionando, mas eles contém duplicação. Como é possível ver, os testes testConteudoLista e testListaDoisItens começam adicionando os filmes starWars e starTrek:

<code>
private MovieList listaFilmes = null;
private Movie starWars = null;
private Movie starTrek = null;
private Movie stargate = null;

protected void setUp(){
  listaFilmes = new MovieList();
  starWars = new Movie();
  starTrek = new Movie();
  stargate = new Movie();
}

public void testListaVazia(){
  assertEquals("O tamanho da lista deve ser 0", 0, listaFilmes.size());
}

public void testListaUmItem(){
  listaFilmes.add(starWars);
  assertEquals("O tamanho da lista deve ser 1", 1, listaFilmes.size());
}

public void testListaDoisItens(){
  listaFilmes.add(starWars);
  listaFilmes.add(starTrek);
  assertEquals("O tamanho da lista deve ser 2", 2, listaFilmes.size());
}

public void testConteudoLista(){
  listaFilmes.add(starWars);
  listaFilmes.add(starTrek);

  assertTrue("A lista deve conter o filme Star Wars", listaFilmes.cointains(starWars));
  assertTrue("A lista deve conter o filme Star Trek", listaFilmes.cointains(starTrek));
  assertFalse("A lista não deve conter o filme Stargate", listaFilmes.cointains(stargate));
}
</code>

Para resolver essa duplicação, o melhor é criar uma classe separada para esses dois testes, inserindo então esses dois filmes na função setup:

<code>
private MovieList listaFilmes = null;
private Movie starWars = null;
private Movie starTrek = null;
private Movie stargate = null;

protected void setUp(){
  listaFilmes = new MovieList();
  starWars = new Movie();
  starTrek = new Movie();
  stargate = new Movie();
  listaFilmes.add(starWars);
  listaFilmes.add(starTrek);
}

public void testListaDoisItens(){
  assertEquals("O tamanho da lista deve ser 2", 2, listaFilmes.size());
}

public void testConteudoLista(){
  assertTrue("A lista deve conter o filme Star Wars", listaFilmes.cointains(starWars));
  assertTrue("A lista deve conter o filme Star Trek", listaFilmes.cointains(starTrek));
  assertFalse("A lista não deve conter o filme Stargate", listaFilmes.cointains(stargate));
}

```
</code>
</div>
<p class="content_p" id="p23">Para melhorar ainda mais o nome dos testes anteriores, deve-se criar uma classe separada para a lista vazia e outra para o teste de adicionar um filme na lista.<br>Elas ficariam, respectivamente, dessa forma:</p>
<div class="content_code" id="code18">
<code>
private MovieList listaFilmes = null;<br>
<br>
protected void setUp(){<br>
  listaFilmes = new MovieList();<br>
}<br>
<br>
public void testTamanho(){<br>
  assertEquals("O tamanho da lista deve ser 0", 0, listaFilmes.size());<br>
}<br>
</code>
</div>
<div class="content_code" id="code19">
<code>
private MovieList listaFilmes = null;<br>
private Movie starWars = null;<br>
<br>
protected void setUp(){<br>
  listaFilmes = new MovieList();<br>
  starWars = new Movie();<br>
}<br>
public void testTamanho(){<br>
  listaFilmes.add(starWars);<br>
  assertEquals("O tamanho da lista deve ser 1", 1, listaFilmes.size());<br>
}<br>
</code>
</div>
</div>
</body>
</html>
```

Página HTML do conteúdo 4

```
<!DOCTYPE html>
<html>
<body>
<h1 class="content_h1" id="header1">4. Avaliação Final sobre o uso de
TDD</h1>
<p class="content_p" id="p1">Deverá ser criada uma função para somar 2
números inteiros usando Java e JUnit.<br>Os testes mais simples a serem
feitos nesse caso são:</p>
<ul  class="content_list" id="list1">
<li>Teste 1: Soma de 0 + 0 deve ser igual a 0.</li>
<li>Teste 2: Soma de 0 + 1 deve ser igual a 1.</li>
</ul>
<div class="content_question" id="question1">
<p class="content_p" id="p2">1-Qual deve ser a primeira ação a ser feita?</p>
<input type='radio' id='q1_a1' name='q1' value=' '><label for='q1_a1'>
Desenvolver o código da função de soma.</label><br>
<input type='radio' id='q1_a2' name='q1' value='*'><label for='q1_a2'> Fazer
o Teste 1 utilizando o JUnit.</label><br>
<br><button onclick='(function (rg) {
let answers= document.getElementsByName(rg);
for (let i=0; i<answers.length; i++) if (answers[i].checked &&
answers[i].value==="*") { alert("Resposta correta!"); return; }
alert("Resposta incorreta :-(");})("q1");'
>Verificar</button>
</div>
<div class="content_question" id="question2">
<p class="content_p" id="p3">2-Marque a opção que melhor descreve o Teste
1:</p>
<input type='radio' id='q2_a1' name='q2' value=' '><label for='q2_a1'> Código
1:</label><br>
<div class="content_code" id="code1">
<code>
public void testSomaZeroMaisZero(){<br>
  assertEquals(0+0, 0);<br>
}<br>
</code>
</div>
<input type='radio' id='q2_a2' name='q2' value=' '><label for='q2_a2'> Código
2:</label><br>
```

```
<div class="content_code" id="code2">
<code>
public void testSomaZeroMaisZero(){<br>
  Matematica m = new Matematica();<br>
  int soma = m.somar("0","0");<br>
  assertEquals(soma, 0);<br>
}<br>
</code>
</div>
<input type='radio' id='q2_a3' name='q2' value=' '><label for='q2_a3'> Código
3:</label><br>
<div class="content_code" id="code3">
<code>
public void testSomaZeroMaisZero(){<br>
  int soma = somar(0,0);<br>
  assertEquals(soma, 0, "Soma com valor di
ferente do esperado.");<br>
}<br>
</code>
</div>
<input type='radio' id='q2_a4' name='q2' value='*'><label for='q2_a4'> Código
4:</label><br>
<div class="content_code" id="code4">
<code>
public void testSomaZeroMaisZero(){<br>
  int soma = somar(0,0);<br>
  assertEquals(soma, 0);<br>
}<br>
</code>
</div>
<br><button onclick='(function (rg) {
let answers= document.getElementsByName(rg);
for (let i=0; i<answers.length; i++) if (answers[i].checked &&
answers[i].value==="*") { alert("Resposta correta!"); return; }
alert("Resposta incorreta :-(");})("q2");'
>Verificar</button>
</div>
<div class="content_question" id="question3">
<p class="content_p" id="p4">3-Qual é a luz do TDD Traffic Light atual e qual
o motivo?</p>
```

```
<input type='radio' id='q3_a1' name='q3' value=' '><label for='q3_a1'> Luz
vermelha. A classe Matematica e a função somar ainda não existem.</label><br>
<input type='radio' id='q3_a2' name='q3' value=' '><label for='q3_a2'> Luz
vermelha. O resultado foi diferente de 0.</label><br>
<input type='radio' id='q3_a3' name='q3' value='*'><label for='q3_a3'> Luz
amarela. A classe Matematica e a função somar ainda não existem.</label><br>
<input type='radio' id='q3_a4' name='q3' value=' '><label for='q3_a4'> Luz
verde. O resultado foi igual a 0.</label><br>
<br><button onclick='(function (rg) {
let answers= document.getElementsByName(rg);
for (let i=0; i<answers.length; i++) if (answers[i].checked &&
answers[i].value==="*") { alert("Resposta correta!"); return; }
alert("Resposta incorreta :-(");})("q3");'
>Verificar</button>
</div>
<div class="content_question" id="question4">
<p class="content_p" id="p5">5-Assinale a alternativa que resolveria o erro
mencionado na questão anterior.</p>
<input type='radio' id='q4_a1' name='q4' value='*'><label for='q4_a1'> Código
1:</label><br>
<div class="content_code" id="code5">
<code>
public class Matematica {<br>
  public int somar (int num1, int num
2){<br>
    return 0;<br>
  }<br>
}<br>
</code>
</div>
<input type='radio' id='q4_a2' name='q4' value=' '><label for='q4_a2'> Código
2:</label><br>
<div class="content_code" id="code6">
<code>
public void testSomaZeroMaisZero(){<br>
  Matematica m = new Matematica();<br>
  int soma = m.somar(0,0);<br>
  assertEquals(soma, 2);<br>
}<br>
</code>
</div>
```

```
<input type='radio' id='q4_a3' name='q4' value=' '><label for='q4_a3'> Código
3:</label><br>
<div class="content_code" id="code7">
<code>
public class Matematica {<br>
<br>
}<br>
</code>
</div>
<input type='radio' id='q4_a4' name='q4' value=' '><label for='q4_a4'> Código
4:</label><br>
<div class="content_code" id="code8">
<code>
public void testSomaZeroMaisZero(){<br>
  int soma = somar(0,0);<br>
  assertEquals(soma, 1);<br>
}<br>
</code>
</div>
<br><button onclick='(function (rg) {
let answers= document.getElementsByName(rg);
for (let i=0; i<answers.length; i++) if (answers[i].checked &&
answers[i].value==="*") { alert("Resposta correta!"); return; }
alert("Resposta incorreta :-(");})("q4");'
>Verificar</button>
</div>
<div class="content_question" id="question5">
<p class="content_p" id="p6">6-Qual a luz do TDD Traffic Light atual e qual o
motivo, depois de corrigir o erro anterior?</p>
<input type='radio' id='q5_a1' name='q5' value=' '><label for='q5_a1'> Luz
vermelha. Em todos os casos a luz vermelha sucede a luz amarela.</label><br>
<input type='radio' id='q5_a2' name='q5' value=' '><label for='q5_a2'> Luz
verde. Em todos os casos a luz vermelha sucede a luz amarela.</label><br>
<input type='radio' id='q5_a3' name='q5' value=' '><label for='q5_a3'> Luz
vermelha. O código não está dando o resultado esperado pelo
teste.</label><br>
<input type='radio' id='q5_a4' name='q5' value='*'><label for='q5_a4'> Luz
verde. Apesar do fluxo comum ser luz vermelha após uma luz amarela, nesse
primeiro teste acabou sendo verde por causa da tipagem do Java.</label><br>
<br><button onclick='(function (rg) {
let answers= document.getElementsByName(rg);
```

```
for (let i=0; i<answers.length; i++) if (answers[i].checked &&
answers[i].value==="*") { alert("Resposta correta!"); return; }
alert("Resposta incorreta :-(");})("q5");'
>Verificar</button>
</div>
<div class="content_question" id="question6">
<p class="content_p" id="p7">7-Qual a próxima ação que deve ser feita?</p>
<input type='radio' id='q6_a1' name='q6' value=' '><label for='q6_a1'>
Refatorar o código.</label><br>
<input type='radio' id='q6_a2' name='q6' value='*'><label for='q6_a2'> Ir
para o próximo teste.</label><br>
<br><button onclick='(function (rg) {
let answers= document.getElementsByName(rg);
for (let i=0; i<answers.length; i++) if (answers[i].checked &&
answers[i].value==="*") { alert("Resposta correta!"); return; }
alert("Resposta incorreta :-(");})("q6");'
>Verificar</button>
</div>
<div class="content_question" id="question7">
<p class="content_p" id="p8">8-Marque a alternativa que corresponde a próxima
etapa do processo de desenvolvimento:</p>
<input type='radio' id='q7_a1' name='q7' value=' '><label for='q7_a1'> Código
1:</label><br>
<div class="content_code" id="code9">
<code>
public void testSomaZeroMaisUm(){<br>
  Matematica m = new Matematica();<br>
  int soma = m.somar(2,1);<br>
  assertEquals(soma, 3);<br>
}<br>
</code>
</div>
<input type='radio' id='q7_a2' name='q7' value='*'><label for='q7_a2'> Código
2:</label><br>
<div class="content_code" id="code10">
<code>
public void testSomaZeroMaisUm(){<br>
  Matematica m = new Matematica();<br>
  int soma = m.somar(0,1);<br>
  assertEquals(soma, 1);<br>
}<br>
```

```
</code>
</div>
<input type='radio' id='q7_a3' name='q7' value=' '><label for='q7_a3'> Código
3:</label><br>
<div class="content_code" id="code11">
<code>
public class Matematica {<br>
  public int somar (int num1, int num
2){<br>
    return num1+num2;<br>
  }<br>
}<br>
</code>
</div>
<input type='radio' id='q7_a4' name='q7' value=' '><label for='q7_a4'> Código
4:</label><br>
<div class="content_code" id="code12">
<code>
public void testSomaZeroMaisUm(){<br>
  Matematica m = new Matematica();<br>
  int soma = m.somar(0,1);<br>
  assertEquals(soma, 2);<br>
}<br>
</code>
</div>
<br><button onclick='(function (rg) {
let answers= document.getElementsByName(rg);
for (let i=0; i<answers.length; i++) if (answers[i].checked &&
answers[i].value==="*") { alert("Resposta correta!"); return; }
alert("Resposta incorreta :-(");})("q7");'
>Verificar
</button>
</div>
<div class="content_question" id="question8">
<p class="content_p" id="p9">9-Qual é a luz atual do TDD Traffic Light e qual
o motivo?</p>
<input type='radio' id='q8_a1' name='q8' value=' '><label for='q8_a1'> Luz
amarela. Existe alguma função sendo chamada que ainda não existe.</label><br>
<input type='radio' id='q8_a2' name='q8' value='*'><label for='q8_a2'> Luz
vermelha. O resultado esperado é 1, mas a função somar está retornando
0.</label><br>
```

```
<input type='radio' id='q8_a3' name='q8' value=' '><label for='q8_a3'> Luz
verde. O resultado do teste é o mesmo valor do resultado da função
somar.</label><br>
<input type='radio' id='q8_a4' name='q8' value=' '><label for='q8_a4'> Luz
amarela. Existe alguma classe sendo instanciada que ainda não
existe.</label><br>
<br><button onclick='(function (rg) {
let answers= document.getElementsByName(rg);
for (let i=0; i<answers.length; i++) if (answers[i].checked &&
answers[i].value==="*") { alert("Resposta correta!"); return; }
alert("Resposta incorreta :-(");})("q8");'
>Verificar</button>
</div>
<div class="content_question" id="question9">
<p class="content_p" id="p10">10-Marque a alternativa que corresponde com a
próxima ação a ser feita:</p>
<input type='radio' id='q9_a1' name='q9' value='*'><label for='q9_a1'> Código
1:</label><br>
<div class="content_code" id="code13">
<code>
public class Matematica {<br>
  public int somar (int num1, int num
2){<br>
    return num1+num2;<br>
  }<br>
}<br>
</code>
</div>
<input type='radio' id='q9_a2' name='q9' value=' '><label for='q9_a2'> Código
2:</label><br>
<div class="content_code" id="code14">
<code>
public class Matematica {<br>
  public int somar (int num1, int num
2){<br>
    return 1;<br>
  }<br>
}<br>
</code>
</div>
```

```
<input type='radio' id='q9_a3' name='q9' value=' '><label for='q9_a3'> Código
3:</label><br>
<div class="content_code" id="code15">
<code>
public void testSomaZeroMaisUm(){<br>
  Matematica m = new Matematica();<br>
  int soma = m.somar(0,1);<br>
  assertEquals(soma, 0);<br>
}<br>
</code>
</div>
<input type='radio' id='q9_a4' name='q9' value=' '><label for='q9_a4'> Código
4:</label><br>
<div class="content_code" id="code16">
<code>
public void testSomaZeroMaisUm(){<br>
  Matematica m = new Matematica();<br>
  int soma = m.somar(0,0);<br>
  assertEquals(soma, 1);<br>
}<br>
</code>
</div>
<br><button onclick='(function (rg) {
let answers= document.getElementsByName(rg);
for (let i=0; i<answers.length; i++) if (answers[i].checked &&
answers[i].value==="*") { alert("Resposta correta!"); return; }
alert("Resposta incorreta :-(");})("q9");'
>Verificar</button>
</div>
<div class="content_question" id="question10">
<p class="content_p" id="p11">11-Qual é a luz do TDD Traffic Light atual e
qual o motivo?</p>
<input type='radio' id='q10_a1' name='q10' value=' '><label for='q10_a1'> Luz
vermelha. O resultado da função ainda não o que se espera nos
testes.</label><br>
<input type='radio' id='q10_a2' name='q10' value=' '><label for='q10_a2'> Luz
amarela. Foi utilizada uma instânciação de classe que não existe.</label><br>
<input type='radio' id='q10_a3' name='q10' value='*'><label for='q10_a3'> Luz
verde. O resultado da função somar corresponde ao resultado esperado dos
testes feitos anteriormente.</label><br>
```

```
<input type='radio' id='q10_a4' name='q10' value=' '><label for='q10_a4'>Luz
amarela. Foi utilizada uma função que não existe.</label><br>
<br><button onclick='(function (rg) {
let answers= document.getElementsByName(rg);
for (let i=0; i<answers.length; i++) if (answers[i].checked &&
answers[i].value==="*") { alert("Resposta correta!"); return; }
alert("Resposta incorreta :-(");})("q10");'
>Verificar</button>
</div>
<div class="content_question" id="question11">
<p class="content_p" id="p12">12-Qual a próxima ação a ser feita?</p>
<input type='radio' id='q11_a1' name='q11' value=' '><label for='q11_a1'>
Nada. A função somar já funciona perfeitamente.</label><br>
<input type='radio' id='q11_a2' name='q11' value=' '><label for='q11_a2'>
Refatorar a classe Matematica.</label><br>
<input type='radio' id='q11_a3' name='q11' value=' '><label for='q11_a3'>
Implementar o código a seguir:</label><br>
<div class="content_code" id="code17">
<code>
private Matematica m;<br>
public void testSomaZeroMaisZero(){<br>
  Matematica m1 = new Matematica();<br>
  int soma = m1.somar(0,0);<br>
  assertEquals(soma, 0);<br>
}<br>
public void testSomaZeroMaisUm(){<br>
  Matematica m2 = new Matematica();<br>
  int soma = m2.somar(0,1);<br>
  assertEquals(soma, 1);<br>
}<br>
</code>
</div>
<input type='radio' id='q11_a4' name='q11' value='*'><label for='q11_a4'>
Implementar o código a seguir:</label><br>
<div class="content_code" id="code18">
<code>
private Matematica m;<br>
protected void setUp(){<br>
  m = new Matematica();<br>
}<br>
<br>
```

```
public void testSomaZeroMaisZero(){<br>
  int soma = this.m.somar(0,0);<br>
  assertEquals(soma, 0);<br>
}<br>
public void testSomaZeroMaisUm(){<br>
  int soma = this.m.somar(0,1);<br>
  assertEquals(soma, 1);<br>
}<br>
</code>
</div>
<br><button onclick='(function (rg) {
let answers= document.getElementsByName(rg);
for (let i=0; i<answers.length; i++) if (answers[i].checked &&
answers[i].value==="*") { alert("Resposta correta!"); return; }
alert("Resposta incorreta :-(");})("q11");'
>Verificar</button>
</div>
</body>
</html>
```