Universidade Federal da Paraíba

Departamento de Ciências Exatas

Curso de Licenciatura em Ciência da Computação

Um jogo de tabuleiro para ensino de fundamentos de projeto orientado a objetos

Trabalho de Conclusão de Curso de Graduação

por

Thallyson de Oliveira Freitas

Orientador: Rodrigo Rebouças de Almeida

Rio Tinto, Junho / 2021

Thallyson de Oliveira Freitas
(thallyson.oliveira@dcx.ufpb.br)

Um jogo de tabuleiro para ensino de fundamentos de projeto orientado a objetos

Trabalho apresentado ao Curso de Ciência da Computação, como requisito parcial para a obtenção do Título de Licenciatura em Ciência da Computação, do departamento de Ciências Exatas da Universidade Federal da Paraíba.

Orientador: Rodrigo Rebouças de Almeida

Rio Tinto

2021

Um jogo de tabuleiro para ensino de fundamentos de projeto orientado a objetos

Introdução

Desde sua fundação no século XVIII por Ada Lovelace e, chegando no Brasil, com a primeira turma de graduação em Ciência da Computação, criada pela universidade Unicamp em 1969, a programação tem tomado cada vez mais espaço na sociedade, chegando no ponto em que hoje, diversas pessoas e profissões giram em torno dela, se tornando um pilar fundamental.

Segundo dados da revista Tiinside, em uma previsão feita pelo Ricardo Villate, vice-presidente da IDC para a América Latina:

> Ecossistemas de TIC. Até 2024, 70% das empresas latino-americanas vão rever seu relacionamento com fornecedores e parceiros para melhor executar estratégias digitais e para a implantação generalizada de recursos e operações autônomas de TI. (Villate, 2020).

A demanda por cursos na área de Tecnologia da Informação tem aumentado ao longo dos anos, porém a evasão dos cursos de graduação permanecem alta, com muitos alunos desistindo ao longo do curso. Levantamento realizado pelo Sindicato das Entidades Mantenedoras de Estabelecimentos de Ensino Superior no Estado de São Paulo (Semesp) revelou que os cursos da área de tecnologia da informação são os que têm a maior taxa de desistência de estudantes.

Ao pleitear a entrada no ensino superior boa parte dos estudantes opta por cursos que estejam em sintonia com seus interesses e habilidades, e, também, cuja formação lhe assegure competência profissional para garantir melhor atuação no mercado de trabalho [Soares e Carvalho 2017]. Como no currículo do ensino médio não está presente a computação, ao ingressarem em cursos superiores que trabalham com termos de computação avançados, passarão por dificuldade de assimilação e aprendizagem dos conteúdos que envolvem tais habilidades.

Disciplinas presentes em cursos de tecnologia da informação (TI), como Introdução a programação, Análise e projetos de sistemas ou Banco de dados, exigem do estudante, a habilidade de interpretação e resolver problemas, lógica de programação, adaptar a solução

para a linguagem de programação trabalhada, raciocínio lógico, entre outros conceitos. Diante dessas habilidades, muitos estudantes acabam apresentando um baixo desempenho nessas disciplinas, conduzindo-os à reprovação ou, até mesmo, à desistência do referido curso [Gomes 2008].

Em um um estudo das disciplinas de programação e os altos índices de reprovação, Valaski e Paraiso comentaram.

> Tradicionalmente, as disciplinas ligadas a programação de computadores tem um alto índice de reprovação. Vários alunos ingressantes em cursos de Computação tem seu primeiro contato com programação nesta disciplina. A necessidade de desenvolver um raciocínio lógico, descrevendo este através de uma linguagem com sintaxe bastante específica, é determinante para este alto índice de reprovações. (Valaski e Paraiso, 2012).

Visando atuar nos problemas mencionados, esse trabalho tem como objetivo desenvolver um jogo de tabuleiro com o objetivo de auxiliar o ensino e aprendizagem de conceitos básicos de projeto orientada a objetos (OO). O objetivo do jogo é auxiliar o aprendizado de projeto OO, trabalhando conceitos como herança, interface, classe, polimorfismo, acoplamento e complexidade, além de fortalecer a habilidade de resolver problemas.

Objetos de aprendizagem em forma de jogos de tabuleiro

A ponte estabelecida entre jogos e aprendizado é uma estrutura sólida para ser explorada nas demais áreas de ensino. Segundo Fernandes (1995) os jogos podem ser empregados em uma variedade de propósitos dentro do contexto de aprendizado. Até mesmo o mais simplório dos jogos pode ser empregado para proporcionar informações factuais e praticar habilidades, conferindo destreza e competência. Silva e Kodama (2004) afirmam que a utilização dos jogos na educação propicia ao educando compreender regras a serem utilizadas no processo de aquisição do conhecimento e assimilar conteúdos que até então pareciam totalmente abstratos.

Segundo Diretrizes Curriculares da Educação Básica (DCES, 2008) acerca dos jogos, destaca-se:

> [...] os jogos se integram aos currículos escolares deixando de ser simples passatempo inconsequente, e sim um lugar de destaque. As atividades em forma de jogo são as que mais podem facilitar e contribuir para o desenvolvimento metodológico de ensino-aprendizagem da criança, em virtude da riqueza de oportunidades que o lúdico oferece. Estimula a criatividade, a crítica, e a socialização, sendo assim uma atividade importante e significativa pelo seu conteúdo pedagógico-social. (PARANÁ, 2008, p.33).

Jogos de tabuleiro propriamente ditos, têm a capacidade de explorar em inúmeras mecânicas que trabalham com gerenciamento de recursos, estratégias de combate, planejamento de ações, uso de probabilidade, raciocínio lógico, além de trabalhar a socialização dos indivíduos. Conforme Oliveira (2004), “o jogo é a forma mais simples e natural para o desenvolvimento de um sentimento grupal. É o elemento da cultura que contém maiores possibilidades para socializar.”

O jogo construído neste trabalho envolve conceitos avançados de programação, vistos no estudo de programação orientada a objetos. Apesar de trabalhar conceitos mais complexos, as características de entretenimento e diversão, comumente presentes em jogos de tabuleiros, não foram deixadas de lado. Desde a construção do protótipo, o aspecto constantemente observado foi se o jogo ainda cumpre seu objetivo de entreter juntamente com o de auxiliar no ensino.

A seção a seguir trata dos conceitos de orientação a objetos que foram utilizados na composição do jogo, que são: classe, herança, polimorfismo, Interface, acoplamento e complexidade de domínio.

Conceitos de Orientação a objetos explorados no jogo

Como o objetivo principal do jogo é o auxílio no ensino e aprendizado de programação orientada a objetos, foi preciso escolher os primeiros conjuntos de conceitos que estariam presentes na estrutura do jogo, conceitos esses que seriam de fácil visualização das ligações no designer de orientação a objetos. Dentre os conceitos de OO, foram escolhidos classe, herança, polimorfismo, e outros que compõem a estrutura de programação, que são: interface, acoplamento e complexidade de um código, pois suas estruturas são mais trabalhadas em diagramas *UML* e são mais usados nas aulas de programação orientada a objetos.

Dos conceitos:

- Classe. É a estrutura mais importante do paradigma de orientação a objetos. Nelas são definidos atributos que são comuns em todos os objetos derivados e métodos que são capazes de fazer operações em uma classe (apêndice 11).
  Além da especificação de atributos, a definição de uma classe descreve também qual o comportamento de objetos da classe, ou seja, que funcionalidades podem ser aplicadas a objetos da classe. Essas funcionalidades são descritas através de métodos. Um método nada mais é que o equivalente a um procedimento ou função, com a restrição que ele manipula apenas suas variáveis locais e os atributos que foram definidos para a classe (Ivan Luiz Marques, 2001).
- Herança. Também conhecida como classe filha, que "herda" características da classe mãe, fazendo uso dos métodos e atributos da classe mãe (apêndice 12).
  É um mecanismo que permite que características comuns a diversas classes sejam fatoradas em uma classe base, ou superclasse. A partir de uma classe base, outras classes podem ser especificadas. Cada classe derivada ou subclasse apresenta as características (estrutura e métodos) da classe base e acrescenta a elas o que for definido de particularidade para ela (Ivan Luiz Marques, 2001).
- Polimorfismo. Uma ação que ocorre quando um objeto tem um comportamento diferente para o mesmo método dependendo de como ele é usado (apêndice 13).
  É o princípio pelo qual duas ou mais classes derivadas de uma mesma superclasse

podem invocar métodos que têm a mesma identificação (assinatura) mas comportamentos distintos, especializados para cada classe derivada, usando para tanto uma referência a um objeto do tipo da superclasse. Esse mecanismo é fundamental na programação orientada a objetos, permitindo definir funcionalidades que operem genericamente com objetos, abstraindo-se de seus detalhes particulares quando estes não forem necessários. (Ivan Luiz Marques, 2001).

- Interface. São padrões já pré definidos através de especificação que determina um conjunto de métodos que podem ser implementados pelas classes que a utilizam (apêndice 14).
- Acoplamento. Neste conceito, é trabalhado a relação das classes dentro da aplicação, quanto maior a relação de dependência ou conhecimento entre elas, maior o acoplamento do código. A necessidade de um alto ou baixo nível de acoplamento varia dependendo do objetivo do projeto.
- Complexidade de domínio. A complexidade de um domínio ou projeto OO está ligada à quantidade de entidades envolvidas, suas relações e ao grau de acoplamento entre estas entidades e os conceitos que compõem a solução. (Citar Domain Driven Design).

Processo de construção do jogo

O planejamento para a construção do jogo, foi dividido no estudo de mecânicas de jogos de tabuleiro e cartas, a definição dos conceitos de programação que seriam abordados, a construção das primeiras cartas e rodadas de testes para refinamento do trabalho. A figura 1 mostra o caminho que foi seguido durante a construção desse trabalho.

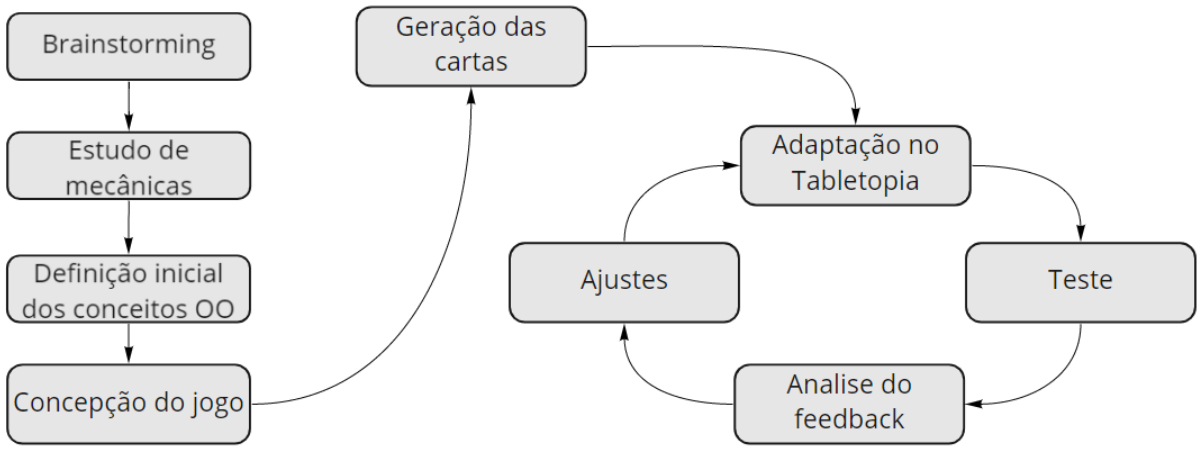


Figura 1 - Estrutura de construção do jogo.

O processo de construção iniciou com diversas rodadas de discussão de ideias entre os autores do jogo: Klayton Souza de Almeida, Rodrigo Rebouças de Almeida e Thallyson de Oliveira Freitas. Neste processo de concepção, foram consideradas inúmeras mecânicas previamente utilizadas em outros de tabuleiro.

Durante as primeiras reuniões de *brainstorming*, decidimos focar no escopo de orientação a objetos, e criar um jogo que pudesse servir de ferramenta para exploração de

conceitos básicos de projeto de sistemas OO.

Com o que foi definido no primeiro ponto, o foco do trabalho foi estudar as mais variáveis mecânicas em jogos de tabuleiro. Nessa parte foi feita mais reuniões de *brainstorming* e utilizando a ferramenta Board Game Arena[1] para usar na prática as mecânicas estudadas em jogos online tanto nos cenários um jogador quanto multijogadores.

O Board Game Arena foi escolhido devido ser uma plataforma especializada em board games, permitindo que pessoas tenham acesso a jogos de tabuleiro em um ambiente virtual. Este ambiente permitiu o estudo de mecânicas, usadas em diferentes jogos, em conjunto a outros jogadores, já que não é possível se reunir presencialmente devido ao contexto da pandemia e isolamento social. Tendo um vasto catálogo de jogos disponíveis para jogar sozinho ou de maneira multijogador, entre amigos ou outros usuários da plataforma. Além de trazer atualizações dos jogos já registrados e tradução sempre que possível.

Para os testes práticos de mecânicas foram escolhidos alguns jogos, como Ticket to Ride[2], Japur e 7 wonders pois eles apresentam estratégias únicas que possam ser trabalhadas na construção do jogo. Entre os jogos escolhidos para testar tivemos Ticket to Ride, onde foi estudado construções de soluções no tabuleiro, forma a qual podíamos elaborar rotas que consiga completar mais de um objetivo simultaneamente e interface dinâmica das peças e dos contratos. Também foi estudado o jogo Jaipur, o qual trabalha com um sistema de *draft* de cartas onde os jogadores têm que disputar pelas cartas na mesa e gerenciar as cartas de camelos presente no jogo, além do jogo trabalhar com o sistema de combos, onde o usuário pode juntar um número de cartas maior e tocar do fixas, quanto maior o número de cartas a serem trocadas, maior é a recompensa e com adicional de pontos extras. Em 7 Wonders, foi trabalhado a interação com outros jogadores, com o foco maior na competitividades entre eles. No jogo também foi visto a gerência de recursos dos jogadores para atrapalhar o desenvolvimento do adversário usando o sistema de draft de cartas, onde a melhoria escolhida pelo o jogador interfere positivamente ou negativamente os demais jogadores.

Diante todas as mecânicas estudadas, foi escolhido trabalhar com o sistema de combos, onde o jogador pode construir uma solução que consiga resolver vários contratos ao mesmo tempo, se encaixando nas estruturas e métricas estabelecidas por cada contrato, e a construção da solução, onde o jogador usará as cartas disponíveis e o próprio tabuleiro para a construção da solução. Os contratos são problemas que representam problemas de design OO, impondo a necessidade de relação entre classes, através de métodos e atributos. De acordo com a forma de aplicação escolhida, a solução vai está sendo montada a cada rodada e visível para todos os jogadores.

Definidas as primeiras mecânicas a serem usadas, foi possível a definição do primeiro conjunto de cartas a serem trabalhadas(apêndice 1) e a primeira demanda de contratos a serem resolvidos(apêndice 2). Foi usada a ferramenta Miro para prototipação, brainstorming e o planejamento de ambas e a construção do design inicial.

Para as cartas referentes a programação, foi escolhido trabalhar com os conceitos de classe concreta, interface, implementar, métodos, atributos, coleções e herança, pois são termos bastante trabalhados em programação orientada a objetos.

Das primeiras cartas de programação definidas temos:

---

[1] http://boardgamearena.com

[2] https://boardgamegeek.com/boardgame/9209/ticket-ride

- Classe Concreta,

Carta criada para representar o conceito de uma classe concreta. Ela trabalha com as métricas de acoplamento e complexidade. (apêndice 3).

- Implementar,

Carta que representa o conceito de implementação de uma interface. A construção de seu design foi baseado na sua representação em em diagramas UML.(apêndice 4)

- Interface,

A carta de interface foi escolhida para representar o conceito de interface em programação. Ela trabalha com as métricas de acoplamento e complexidade. A estrutura do seu nome foi baseada em diagramas UML.(apêndice 5)

- Método,

A carta de Método foi criada para trabalhar o conceito de métodos em programação orientada a objetos. No jogo, é usada para representar a ligação de uma carta do tipo Classe Concreta a outra carta. Sua estrutura teve como inspiração sua representação em diagramas UML.(apêndice 6)

- Atributo,

A carta de Atributo foi criada para trabalhar o conceito de atributo em programação orientada a objetos. No jogo, ela é usada para representar a ligação de uma carta do tipo Classe Concreta a outra carta. Sua estrutura teve como inspiração sua representação em diagramas UML(apêndice 7)

- Coleção,

A carta de Coleção foi criada para trabalhar o conceito de coleção em programação orientada a objetos. No jogo é usada para representar a ligação de uma carta do tipo Classe Concreta a outra carta. Sua estrutura teve como inspiração sua representação em diagramas UML (apêndice 8)

- Extends

A carta de Extends foi criada para a representação do conceito de herança entre as classes. Sua estrutura teve como inspiração sua representação em diagramas UML (apêndice 9).

Os contratos foram construídos tendo como a base a estruturas de diagramas UML, deixando claro os as cartas que serão utilizadas na construção da sua solução e com uma ordem sugerida. Com a mecânica de combos definida, foi criado um total de quinze contratos inicialmente, com estruturas semelhantes para facilitar a criação de soluções que abranjam diversos problemas. Foi usada a ferramenta Miro para a construção inicial do designer das cartas(apêndice 2) e os contratos mais complexos foram baseados nas estruturas de padrões de projeto como o strategy..

Devido a circunstâncias atuais de pandemia e isolamento social, logo no início da sua criação, o jogo precisou ser adaptado para o meio virtual, fazendo uso da ferramenta Tabletopia para tornar essa transformação possível. A plataforma oferece inúmeros jogos em seu catálogo para testes, permite a criação do seu próprio jogo, com diversos recursos que permite a customização de cartas, adicionamento de novos designs, configuração da mesa inicial, sistemas de rodadas, pontuação entre outras possibilidades acessíveis. Tal modificação precisou ser estudada para melhor experiência dos jogadores, tornando necessário a aplicação com estudantes para coleta de dados e melhorias.

Com o estudo da ferramenta, a adaptação do projeto foi feita de maneira simples. Primeiramente foi padronizado e polido todas as cartas e contratos, com ajustes na resolução para que não ficasse distorcidos durante o jogo, depois foi usado as

configurações padrão da plataforma, com uma mesa(apêndice 10), onde o jogo fica, e a possibilidade de 8 jogadores simultaneamente, e então, foi adicionadas as cartas de programação e os contratos, espalhados na mesa em pilhas distintas para melhor organização e acesso durante a aplicação.

Círculo de testes

Após a adaptação do jogo para a plataforma virtual, foi iniciado o círculo de testes, onde o jogo era submetido a um grupo de *testadores* e era coletado *feedbacks* a da estrutura das cartas, se as regras eram claras, o número de cartas é suficiente, verificar se as cartas de Contrato estão demonstrando de forma clara o seu objetivo e as cartas necessárias para a construção da sua solução.

Ao término de cada sessão de testes, que eram feitas pela ferramenta Tabletopia e o meet para comunicação, eram feitas reuniões, entre os criadores do jogo, para analisar os *feedbacks* coletados. O resultado desses encontros, eram transformados em melhorias que eram implementadas ao jogo, sendo testadas com um novo grupo de jogadores.

A primeira aplicação ocorreu em conjunto com o professor Rodrigo Rebouças, com seu grupo do projeto AYTY onde se encontravam alunos de diferentes períodos e estudantes da turma de análise e desenvolvimento de sistemas.

Durante a aplicação foi feita perguntas para confirmar os períodos e a familiaridade deles com jogos de tabuleiro ou cartas. Como resultado, tivemos que 50% dos alunos, estava cursando a disciplina de análise e desenvolvimento e os outros 50% já haviam cursado em outro período, também foi visto que todos do grupo gostam desse estilo de jogo.

Após aplicar o questionário com as perguntas referente se o manual era claro nas regras e perguntas sobre o jogo conseguiu atingir seu objetivo de trabalhar os conceitos de programação e design orientado a objetos, tivemos como resultado que: 75% dos estudantes falaram que o manual foi bastante claro e útil para o entendimento do jogo, os outros 25% disseram que não ficou muito claro mas com a explicação dada antes do início da partida foi o suficiente para poderem compreenderem as regras.

Os alunos ficaram bastante animados com as possibilidades de construção de diversas soluções para um mesmo problema e quanto usavam interface, mesmo não pedida pelo problema, eles conseguiam reduzir a pontuação para um melhor placar. Todos os estudantes confirmaram que o jogo cumpriu seu objetivo de trabalhar os conceitos e de como as classes se interagem entre elas e a aplicação de interfaces e/ou herança para uma solução melhor construída.

Com o resultado da primeira aplicação do jogo, foi visto que era preciso detalhar mais o manual do jogo, retirar as várias representações das cartas de Classe Concreta(as diferentes cores que ela tinha sendo resumidas em uma só cor), colocar uma orientação nas cartas de ligação(cartas de Método, Atributo e Coleção) para ser visível a origem e o destino.

Foram feitas atualizações no design das cartas de Programação apontadas pelos jogadores, a adaptação de todas as cartas de Contrato, uma reconstrução do manual de regras e a definição do objetivo para o fluxo da partida.

A segunda rodada do círculo de testes foi feita em dois momentos com dois jogadores em cada. Os jogadores eram do curso de ciência da Computação do campus IV

da UFPB e todos cursaram as disciplinas de Programação Orientada a Objeto e Análise de Sistemas de Projeto. Foi fornecido o novo manual(apêndice 16) e o site da ferramenta Tabletopia, sem a explicação das regras que era demonstrado antes da partida, a fim de validar a funcionalidade do manual e se o fluxo da partida está aceitável para os jogadores.

Durante a segunda parte de testes foi observado que o manual foi o suficiente para que os jogadores conseguissem conduzir o fluxo do jogo e montarem as soluções para as cartas de Contrato. Ao longo das três rodadas da partida, tivemos soluções válidas e diferentes entre os jogadores. Foi observado também que os jogadores não utilizaram as cartas de Extends e Interface durante as partidas.

Apresentação da solução

Para a verificação se os contratos estão sendo de fácil compreensão para os jogadores, que as cartas de programação são eficientes e se as três rodas tem uma duração aceitável, foi criado uma primeira versão na ferramenta Tabletopia, com as mecânicas básicas e um manual do jogo (apêndice 16), contendo as informações das cartas, as ações durante as rodas e o objetivo da partida.

O jogo construído neste trabalho possui 15 cartas de Contrato(apêndice 15), 8 tipos de cartas de Programação(apêndices 3-9), um manual de regras e é dividido em três rodadas, onde os jogadores irão construir uma solução única para os desafios propostos durante o jogo, refatorando o código(mudando sua estrutura da sua solução) sempre que for preciso para agregar os novos desafios, em forma de cartas de Contrato, os quais serão adicionados a cada rodada, tendo o cuidado para manter a solução compatível com os problemas anteriores.

Para a construção das soluções das cartas de Contrato é disponibilizado, na mesa do jogo(apêndice 17), um conjunto de cartas de Programação, que estão disponíveis para uso durante toda a partida sem limitações além da sua quantidade(15 cartas de cada tipo). O jogador vencedor é o que, na última rodada, tiver uma solução com a menor pontuação em Acoplamento total. Em caso de empate, vence o jogador que tiver a menor pontuação total de complexidade e se o empate persistir, ambos os jogadores são considerados vencedores.

O jogo usa uma estrutura semelhante a de diagramas *UML* nas cartas de Contrato e nas cartas de Programação para estimular o design OO nos jogadores. Também é explorado em suas regras a interações entre as classes concretas por meio de métodos, atributos, o uso de interface e a implementação da mesma por classes concretas.

Durante o desenvolvimento do jogo, foram criados *playtests* para verificação da solução. Com o primeiro tendo o objetivo de validar as cartas de Contrato e Programação, observando os principais pontos de serem de fácil compreensão, se o jogador consegue estabelecer ligação com os conceitos de programação e se comprem seu objetivo de estimular o pensamento de design OO. Já o segundo *playtests* teve com o papel verificar a duração das partidas, se as regras do jogo estavam claras e funcionais e se as cartas disponíveis para os jogadores, são suficientes para a duração das 3 rodadas.

O resultado obtido durante os *playtests* foi satisfatório para a evolução do jogo. Pontos como ajustes nas cartas de Programação para demonstrar o sentido de posse, tamanho das cartas de Contrato e design das cores de certas estruturas, foram rapidamente

modificados para realização de mais testes com grupos diferentes(todos tendo um conhecimento de programação para poder compreender melhor os conceitos).

Foi visto que a construção do designer das cartas se baseando na estrutura de UML foi bem aceito pelos jogadores, principalmente nas cartas de Desafio, pois facilitou o entendimento e a lógica para a construção das soluções, apontadas pelos mesmo que era fácil ver a ligação inicial entre as classes. Um dos participantes declarou: "É como se eu estivesse programando sem usar códigos".

Atualizações Futuras

Apesar do jogo cumprir com seu objetivo de trabalhar conceitos de designer de orientação a objeto para auxílio no aprendizado do mesmo em turmas de programação, ao término das primeiras séries de testes, foi identificado que é necessário a realização de mais rodadas de *playtests* para poder colher mais pontos de melhorias. Além das atualizações realizadas durante esse trabalho foi identificado alguns pontos de melhoria como que seriam implementados em atualizações futuras.

Durante as partidas foi identificado o pouco uso das cartas de Extends e Interface, nesse ponto é preciso melhorar a descrição delas no manual e fazer ajustes nas cartas de Contrato para retirar suas representação de maneira de deixar o seu uso intuitivo para que o jogador a utilize como estratégia de diminuição de Acoplamento da solução.

Também é previsto o aumento do número de mecânicas usadas no jogo, retirando o acesso ilimitado às cartas de Programação pelos jogadores, dessa maneira deixando o jogo mais competitivo e atraente para um público que já domina os conceitos de programação do jogo, querendo fazer seu uso como entretenimento.

As pontuações de Acoplamento e Complexidade estão bem definidas nas cartas que a usam, sendo observado nas partidas que a pontuação final era bem próxima entre os jogadores. Porém é previsto seu uso em todas as outras cartas de Programação para que o uso da carta de Interface, e outras que diminuam a pontuação, sejam mais necessárias na construção das soluções pelos jogadores.

Em forma de complemento, sequência ou inseridas diretamente no jogo, é possível expandir os conceitos de programação do jogo. Como a aplicação direta de padrões de projeto, banco de dados entre outros que são vistos em sala de aula de cursos de programação.

Referências

Gomes, A. J. (2010) “Dificuldades de aprendizagem de programação de computadores: contributos para a sua compreensão e resolução”. Tese (Doutorado em Engenharia Informática) – Faculdade de Ciências e Tecnologia, Universidade de Coimbra. Coimbra, p. 492. Acesso em: 6 maio 2021.

Soares, F. A. L. Carvalho, R. B. (2017) “Proposta de um Portal Educacional para estudantes de programação de computadores”. In: Revista Abakós. v. 5, n. 2, p. 36- 58, maio. 2017 - ISSN: 2316-9451. Acesso em: 3 maio 2021.

Joselaine Valaski e Emerson Cabrera Paraiso. Limitações da utilização do Alice no ensino de programação para alunos de graduação. In Anais do Simpósio Brasileiro de Informática na Educação, Vol. 23, No. 1, 2012. 21 .Acesso em: 5 maio 2021.

BOARD Game Arena. 210506-1332-VO. [*S. l.*]: © Copyright Tabletopia, 2010. Disponível em: https://boardgamearena.com. Acesso em: 6 maio 2021.

LUDOPEDIA. [*S. l.*]: LUDOPEDIA COMERCIO LTDA - ME. Disponível em: https://www.ludopedia.com.br. Acesso em: 19 mar. 2021.

MIRO. [*S. l.*], 2010. Disponível em: https://miro.com. Acesso em: 6 maio 2021.

TABLETOPIA. [*S. l.*]: © Copyright Tabletopia, 2014. Disponível em: https://tabletopia.com/. Acesso em: 6 maio 2021.

Ticket to Ride. Alan R. Moon. ed. Galápagos Jogos, Days of Wonder, 2004.

Jaipur. Sébastien Pauchon. ed. Galápagos Jogos, Asmodee, 2009.

7 Wonders : Second Edition. Antoine Bauza. ed. Asmodee, Rebel Sp. z o.o., Asmodee Italia, ADC Blackfire Entertainment, Repos Production, 2020.

GEHLEN, Salete; LIMA, Christine. JOGOS DE TABULEIRO: UMA FORMA LÚDICA DE ENSINAR E APRENDER. Governo do estado do Paraná, [*s. l.*], 2013. Disponível em: http://www.diaadiaeducacao.pr.gov.br/portals/cadernospde/pdebusca/producoes_pde/2013/2013_unicentro_edfis_artigo_salete_marcolina_gehlen.pdf. Acesso em: 6 maio 2021.

OLIVEIRA, E. E. Relação com jogo infantil e sua aplicação na área da psicopedagogia. Disponível em: . Acesso em: 06 mai. 2021.

PARANÁ. Diretrizes Curriculares da Educação Básica – DCES - Educação Física. Curitiba, PR: SEED, 2008.

RICARTE, Ivan. Programação Orientada a Objetos: Uma Abordagem com Java. Universidade Estadual de Campinas, 2001. Disponível em: https://www.dca.fee.unicamp.br/cursos/PooJava/Aulas/poojava.pdf. Acesso em: 6 maio 2021.

LUQUE, Leandro. Coesão e Acoplamento em Sistemas Orientados a Objetos: Melhorando a qualidade de seus projetos. Centro Paula Souza, 2010. Disponível em: https://www.researchgate.net/publication/261026207_ Coesao_e_Acoplamento_em_Sistemas_OO. Acesso em: 6 maio 2021.

BOARD Game Design: Your Guide to Better Playtests. *In*: HUMPHREY, Amber. [*S. l.*], 2021. Disponível em: https://www.backerkit.com/blog /board-game-design/. Acesso em: 6 maio 2021.

SILVA, Rodrigo; MIRANDA, Elisangela; PEREIRA, Thiago. Uma Experiência no Ensino de Padrões de Projeto Utilizando Jogo no Processo de Ensino-Aprendizagem. V Congresso Brasileiro de Informática na Educação, 2016. Disponível em: https://br-ie.org/pub/index.php/wcbie/article/ view/7047. Acesso em: 6 maio 2021.

ISAAC, Shalev; ENGELSTEIN, Geoffrey. Buildins Blocks of Tabletop Game Design: An Encyclopedia of Mechanisms. [*S. l.*]: Taylor & Francis Group, 2020.

PEREIRA, Wilder. Introdução à Complexidade de Algoritmos. Medium.com, 2019. Disponível em: https://medium.com/nagoya-foundation/introdução-à-complexidade-de-algoritmos-4a9c237e4ecc#:~:text=Um%20algoritmo%20pode%20ser%20melhor,um%20programa%20executa%20suas%20computações. Acesso em: 8 maio 2021.

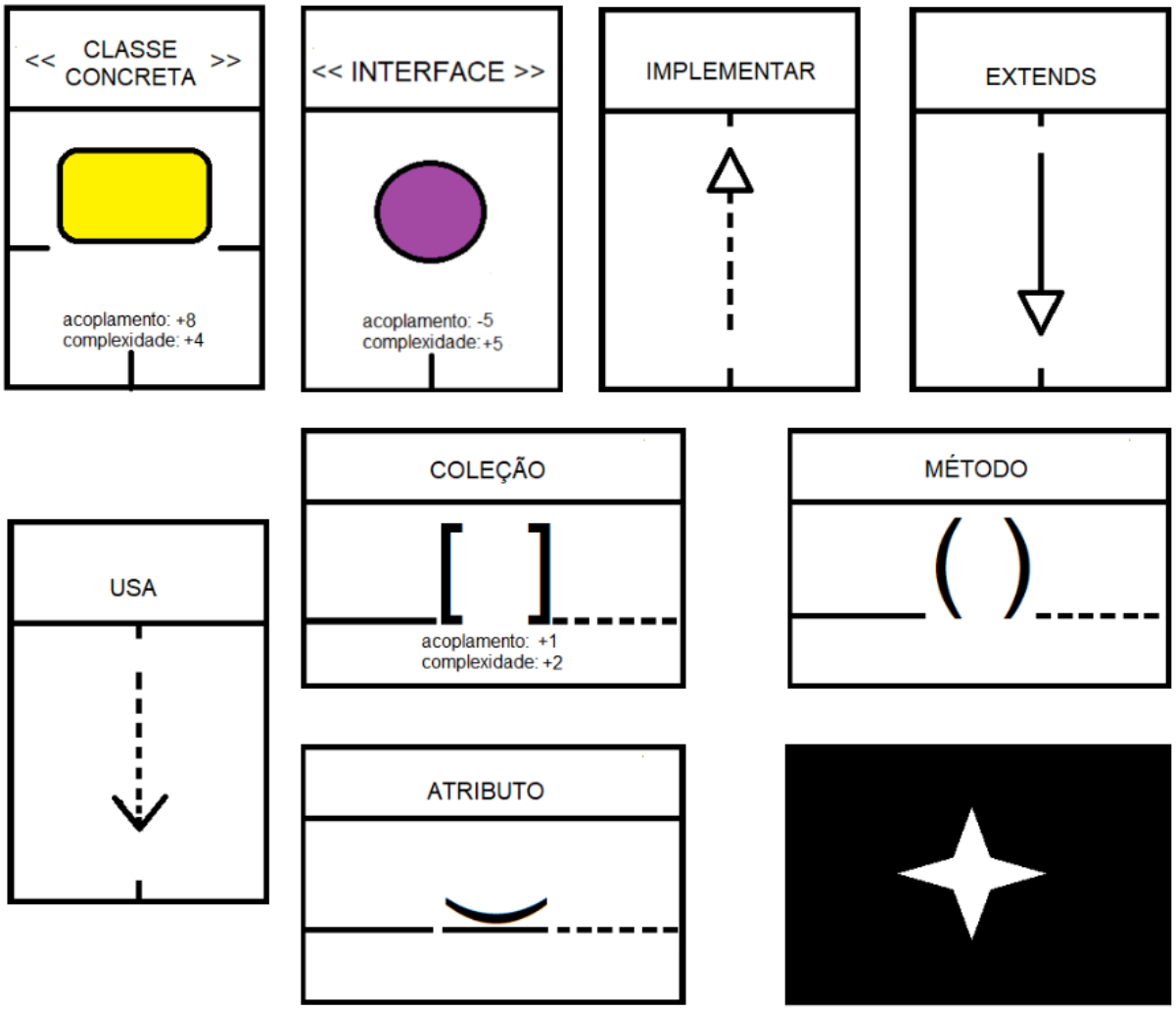
TEIXEIRA, Susane Fernandes de Abreu. Uma reflexão sobre a ambiguidade do conceito de jogo na educação matemática**.** 2008. Dissertação. Mestrado, programa de pós graduação em educação. Universidade de São Paulo. Faculdade de Educação. São Paulo.

DA SILVA, Aparecida Francisco; KODAMA, Helia Matiko Yano. Jogos no ensino da matemática. II Bienal da Sociedade Brasileira de Matemática, 2004.

SILVA, Maria José de Castro. O jogo como estratégia para a resolução de problemas de conteúdo matemático. Psicologia Escolar e Educacional, 2008.

Apêndice

Apêndice 1

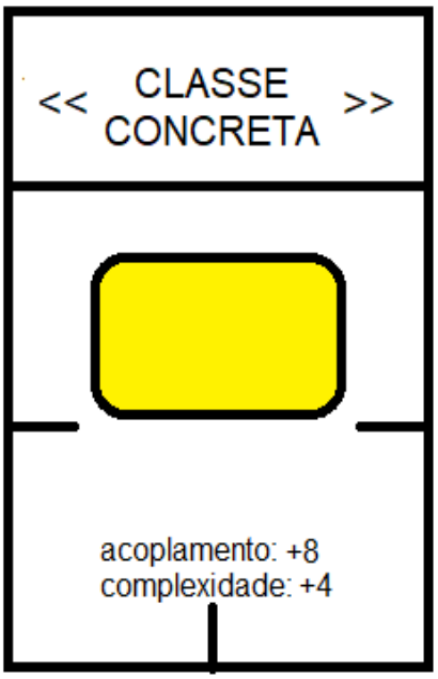


Conjunto de cartas de programação.

Apêndice 2 - Conjunto de contratos

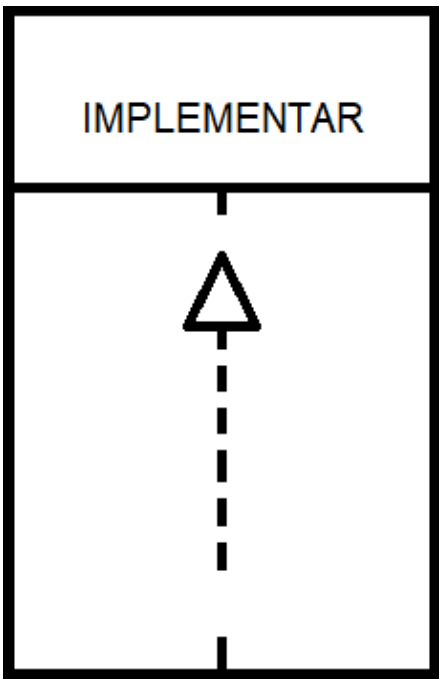
Primeiro conjunto de contratos estabelecidos

Apêndice 3



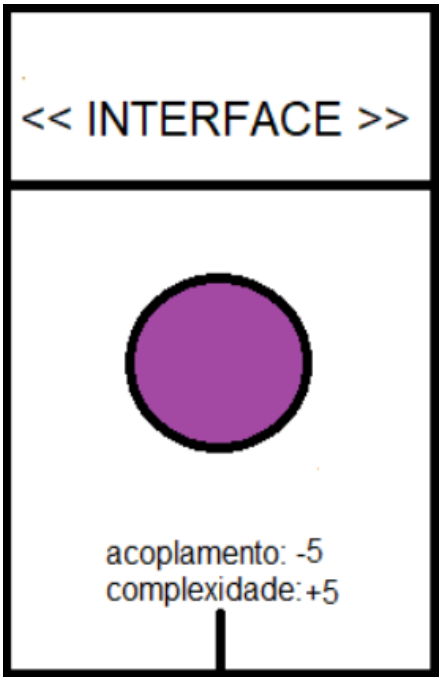
Carta de programação - Classe Concreta

Apêndice 4



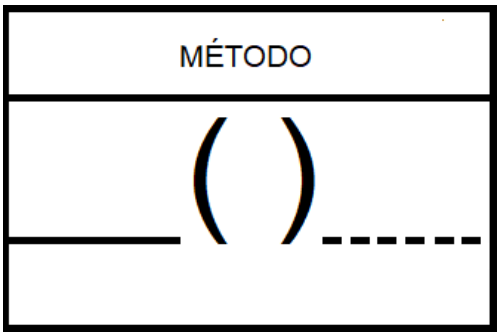
Carta de Implementa.

Apêndice 5



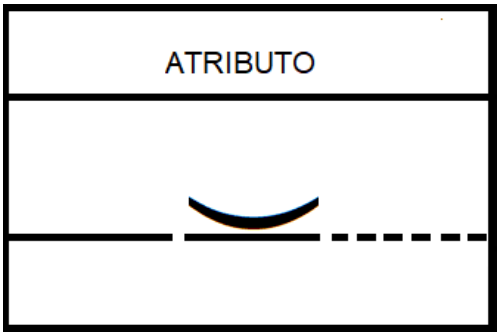
Carta Interface.

Apêndice 6



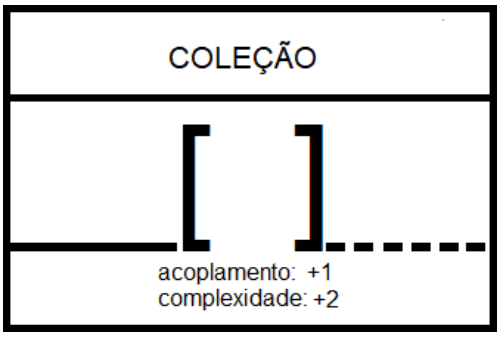
Carta Método.

Apêndice 7



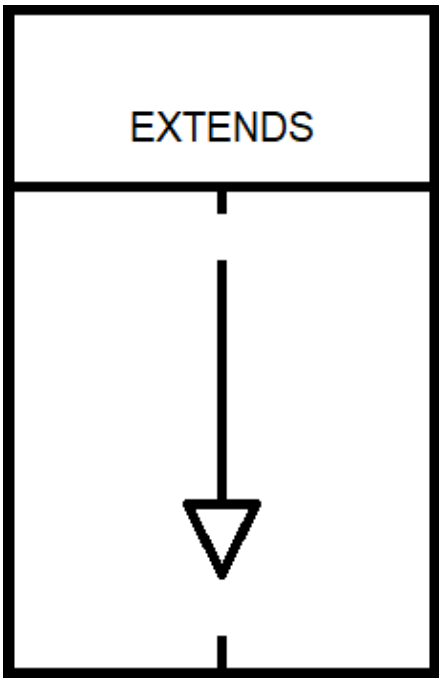
Carta Atributo.

Apêndice 8



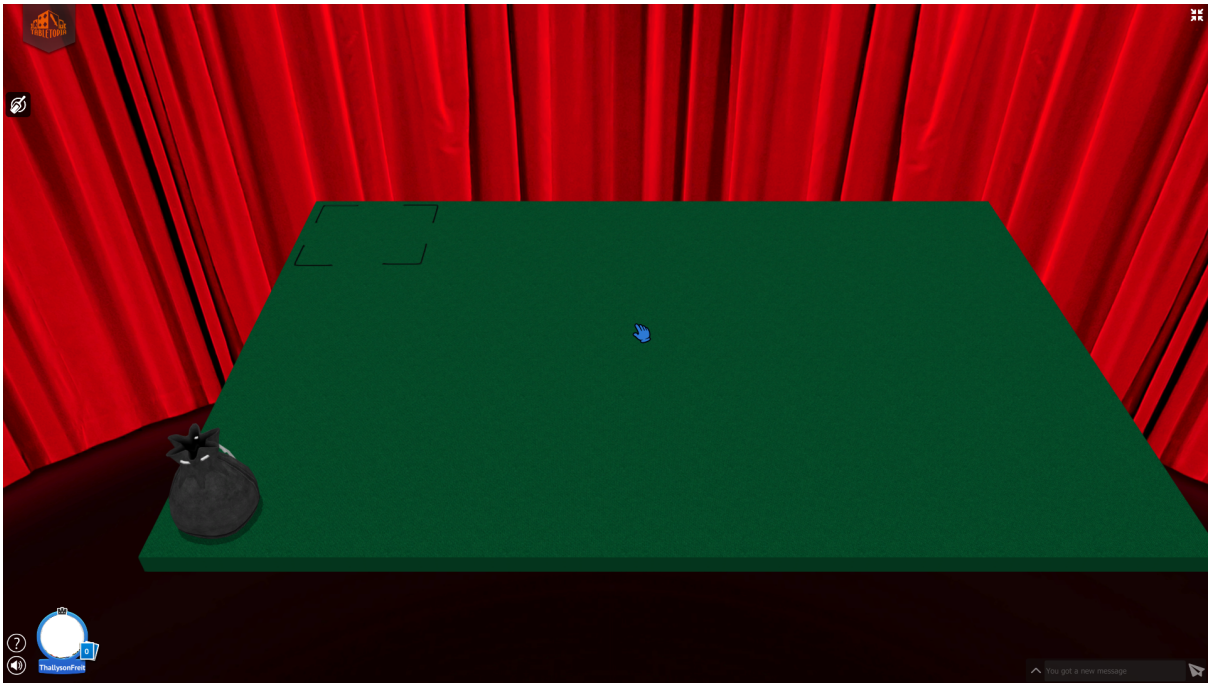
Carta Coleção.

Apêndice 9



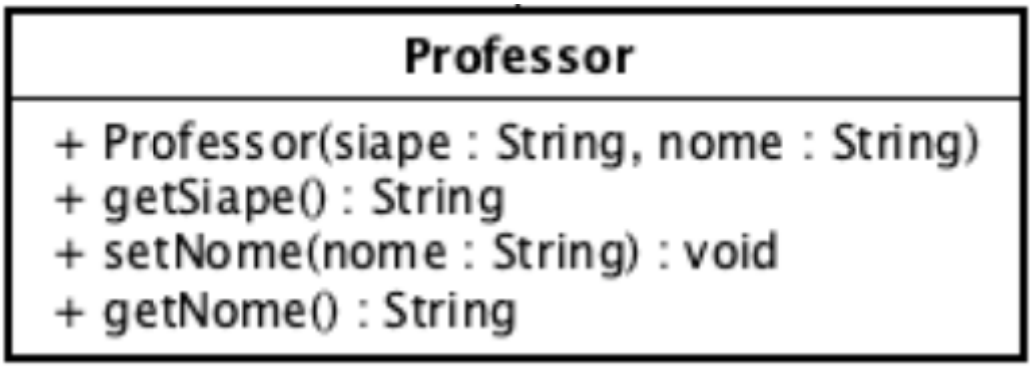
Carta Extends.

Apêndice 10



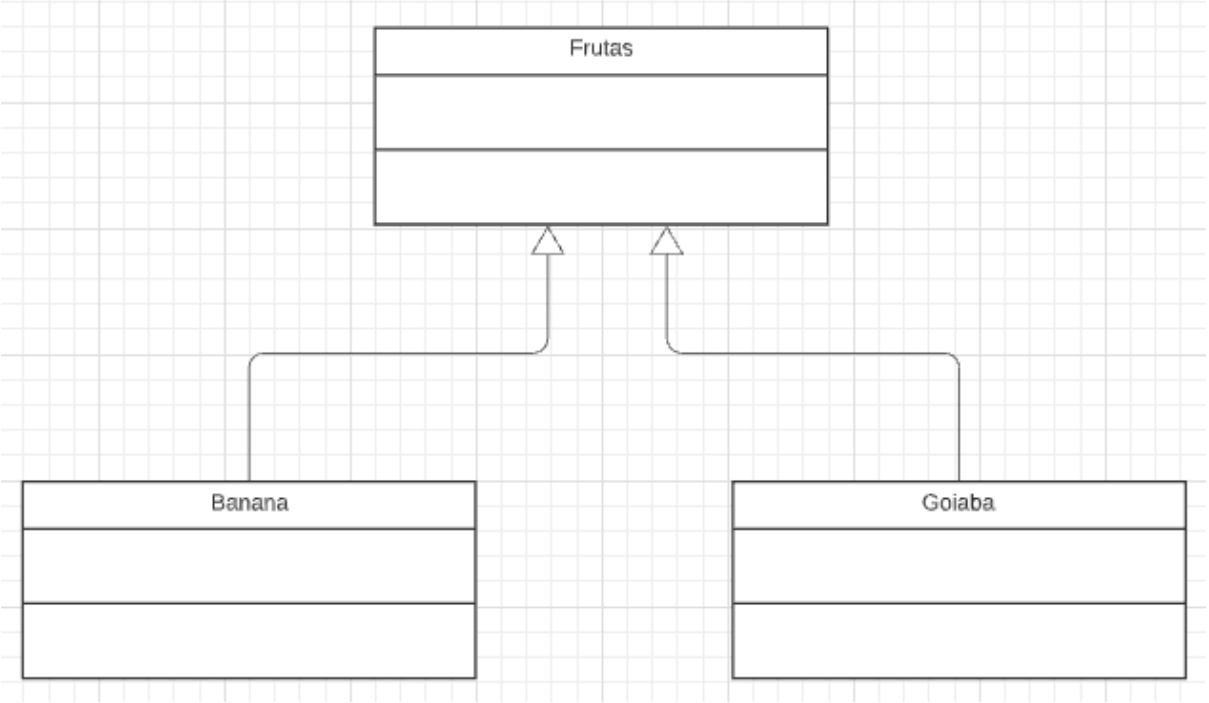
Mesa do Tabletopia

Apêndice 11



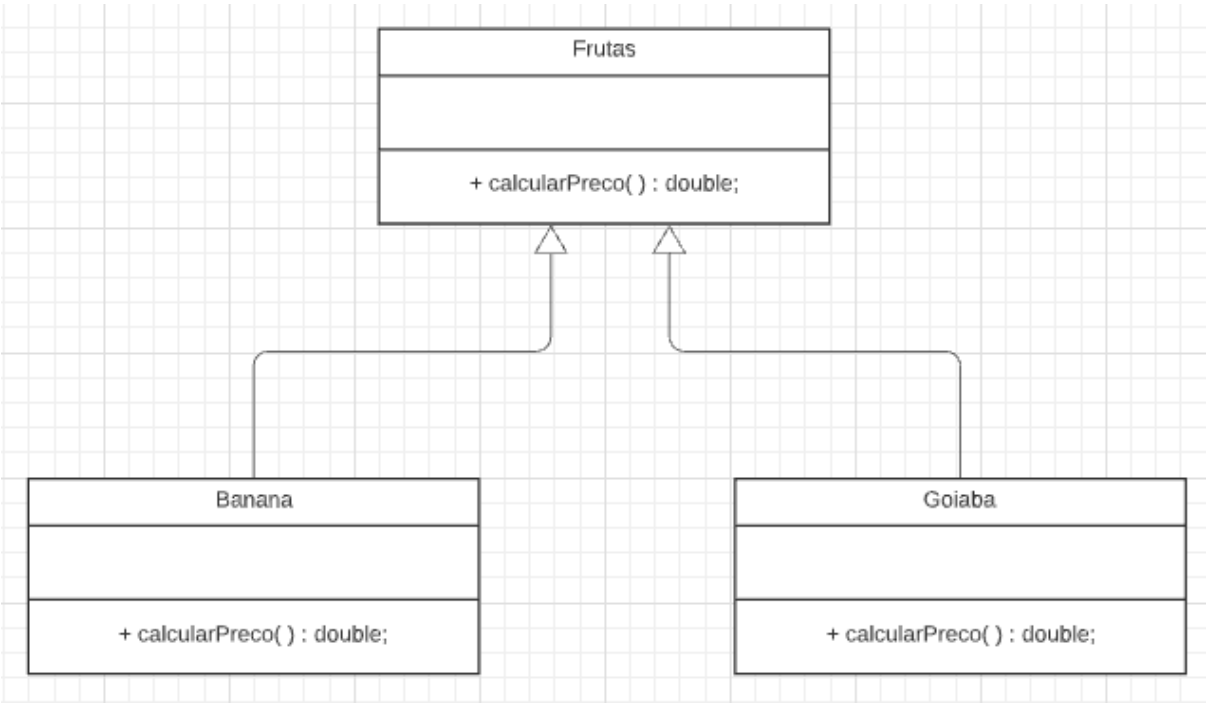
Representação de uma classe em UML

Apêndice 12



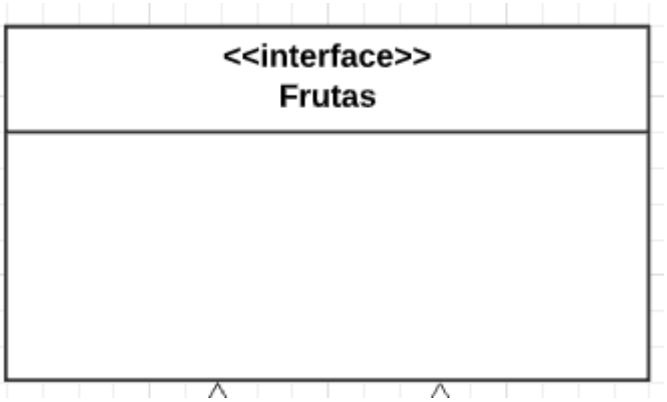
Representação de herança em UML

Apêndice 13



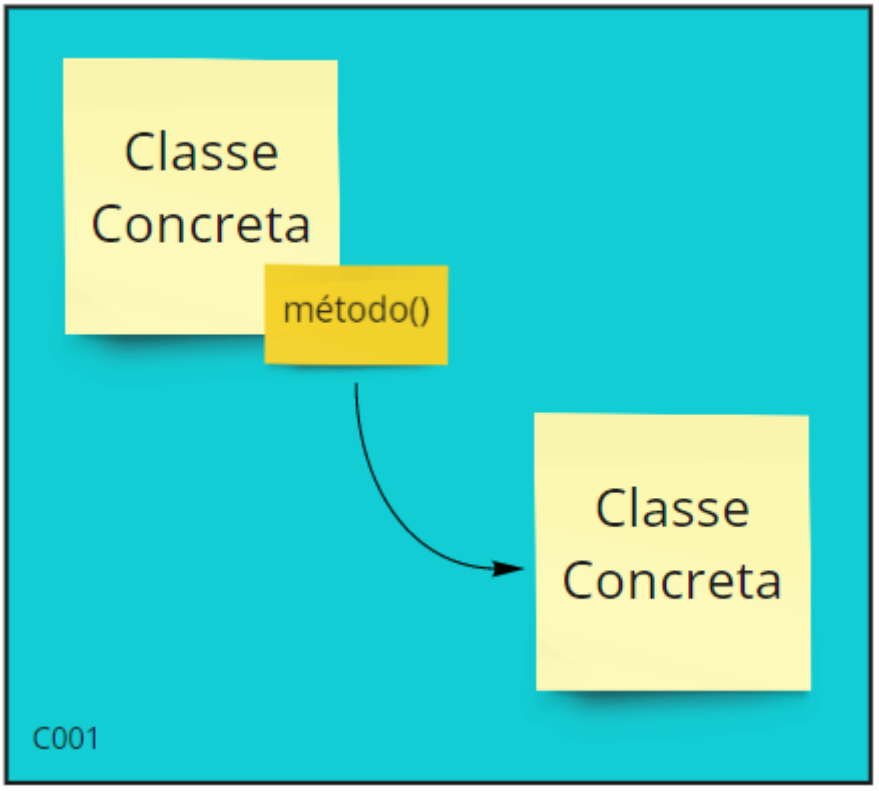
Representação de polimorfismo em UML

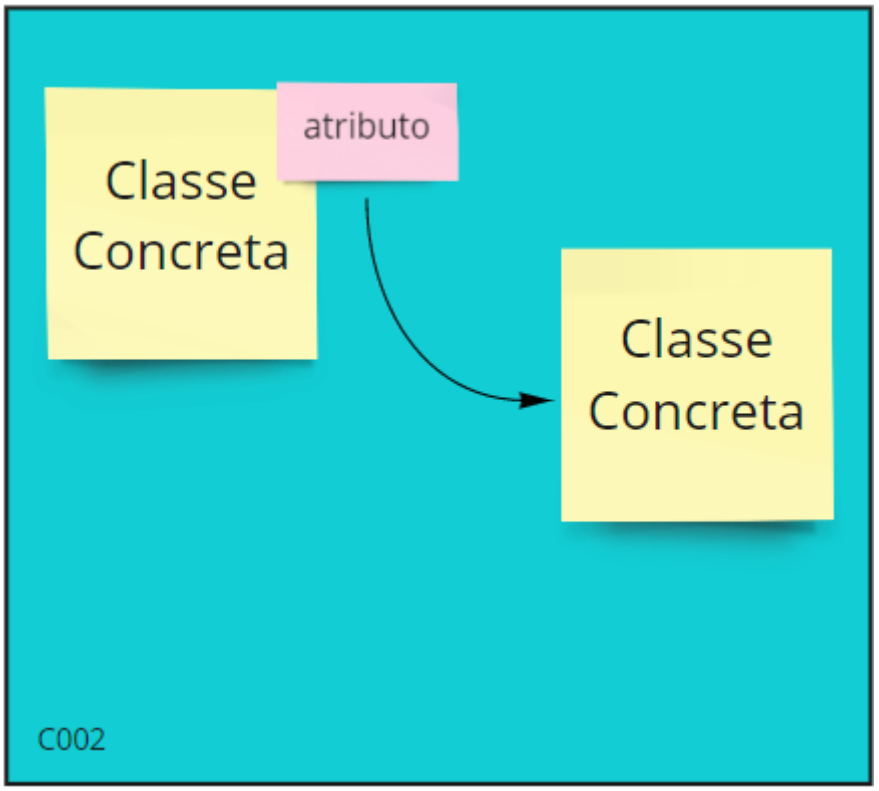
Apêndice 14

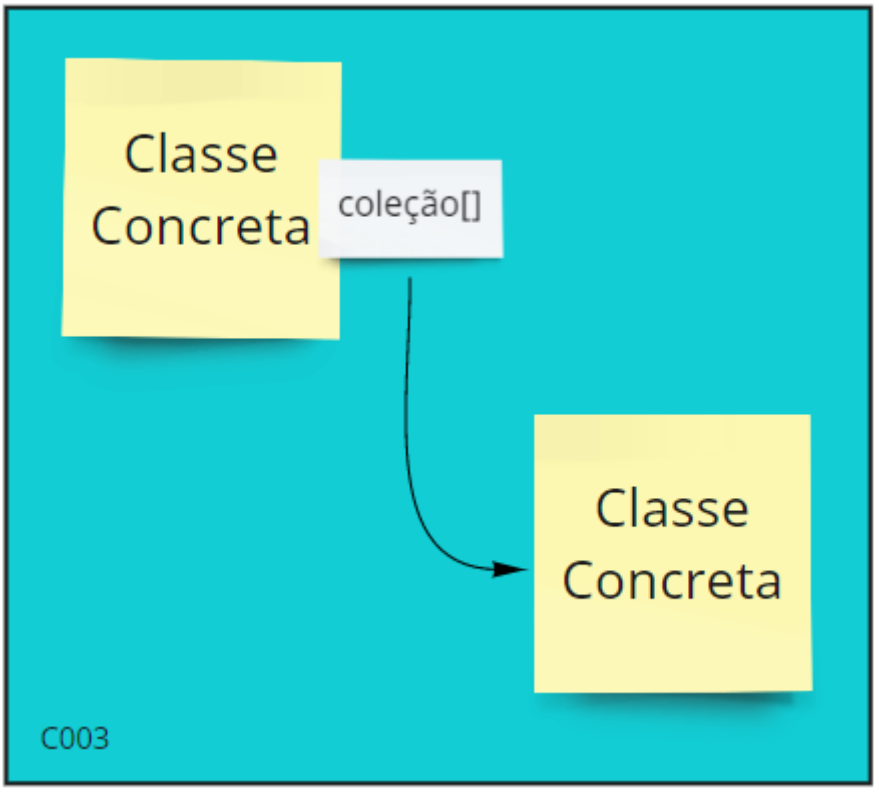


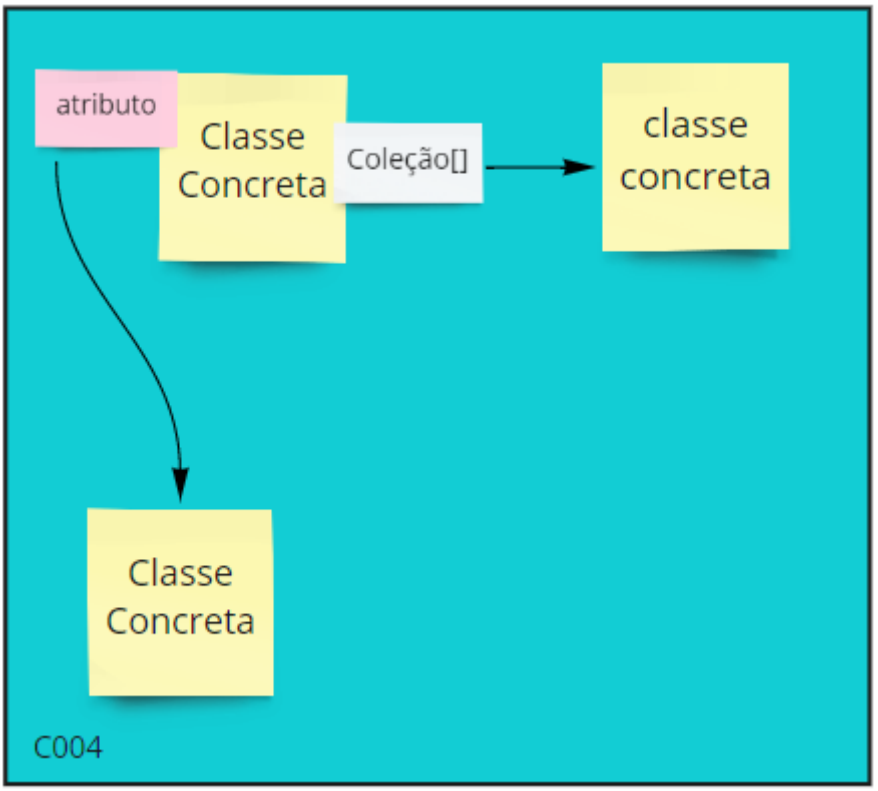
Representação de interface em UML

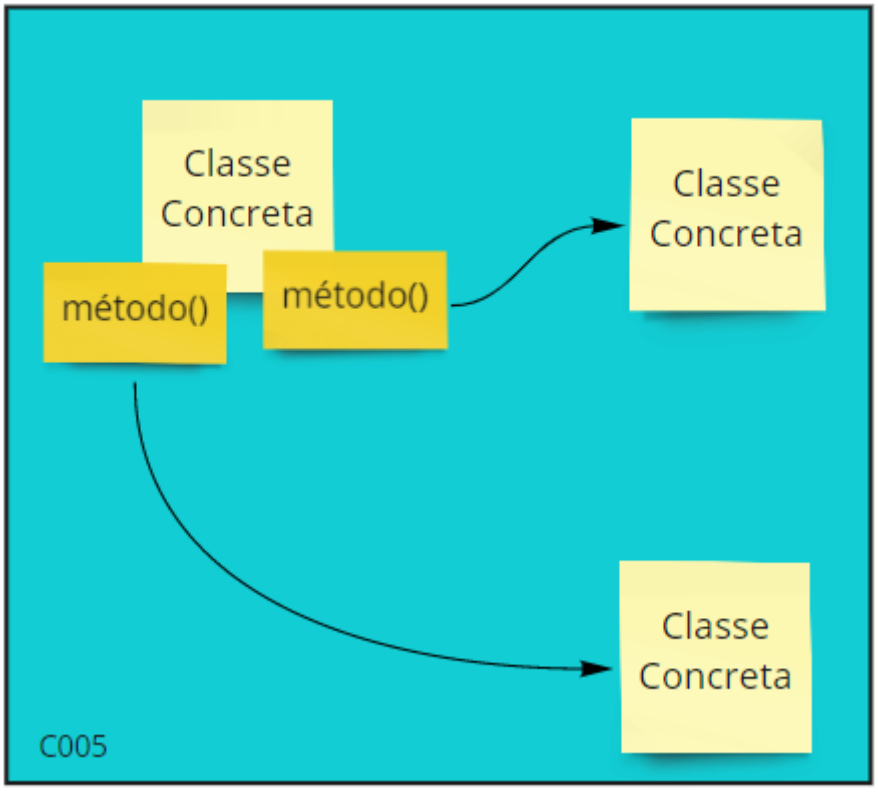
Apêndice 15 - Cartas de Contrato

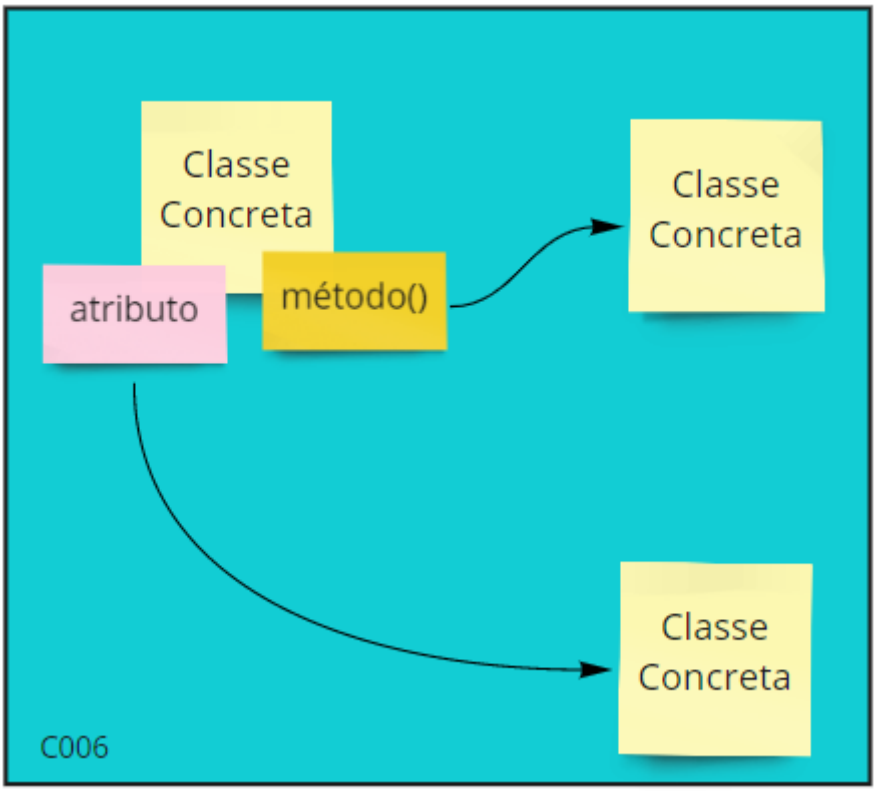


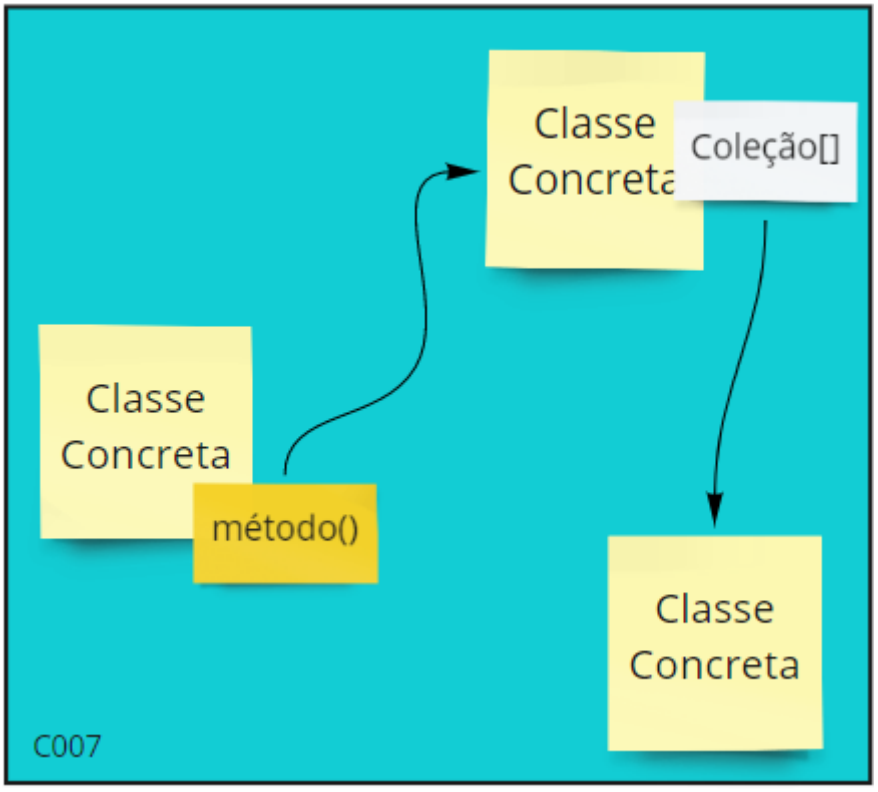

Classe
Concreta
atributo
Classe
Concreta
C002

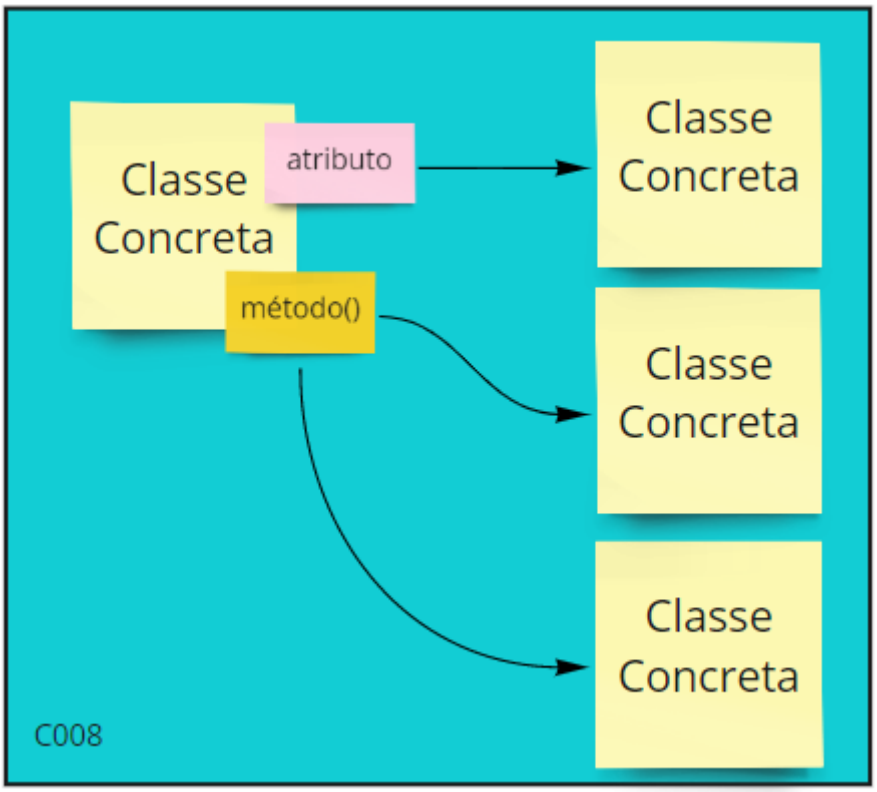

Classe
Concreta
coleção[]
Classe
Concreta
C003

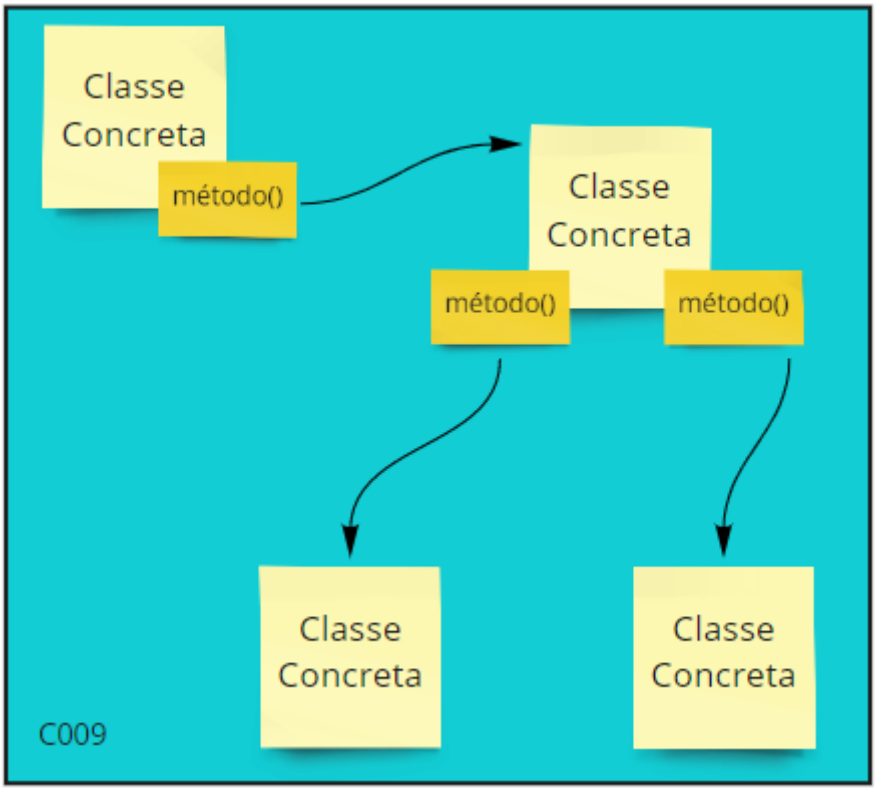

atributo
Classe Concreta
Coleção[]
classe concreta
Classe Concreta
C004

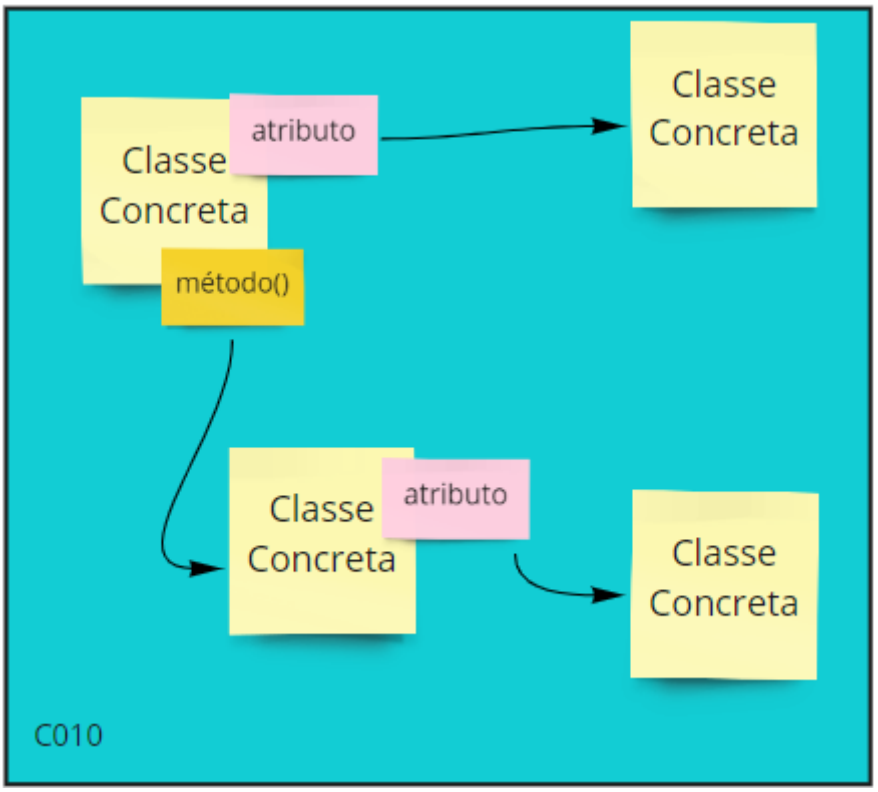

Classe Concreta
método()
método()
Classe Concreta
Classe Concreta
C005

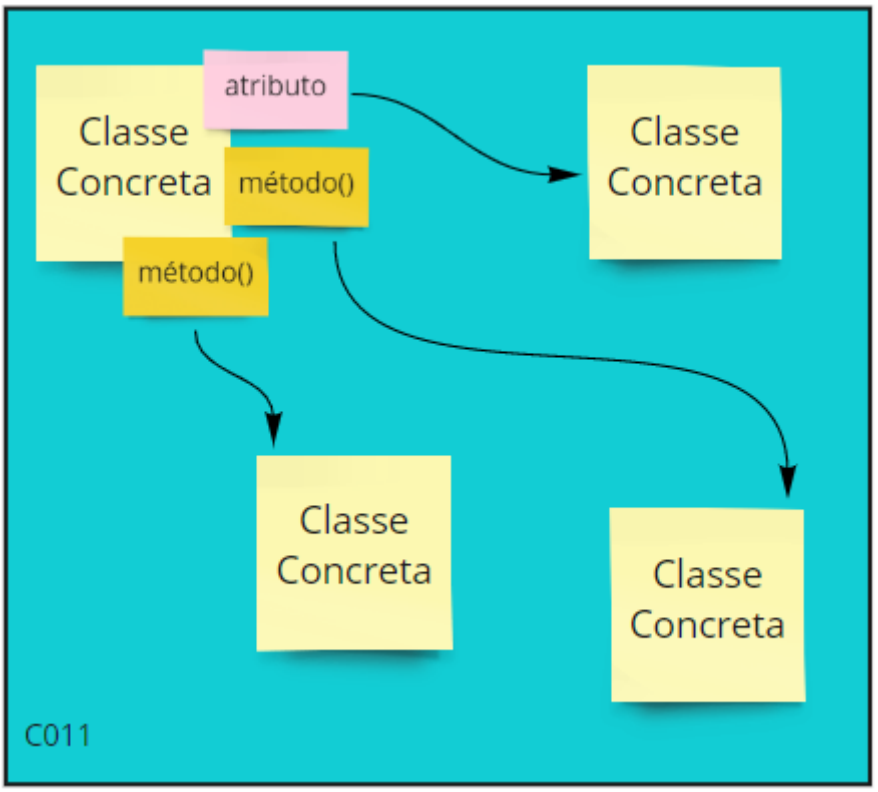

Classe
Concreta
atributo
método()
Classe
Concreta
Classe
Concreta
C006

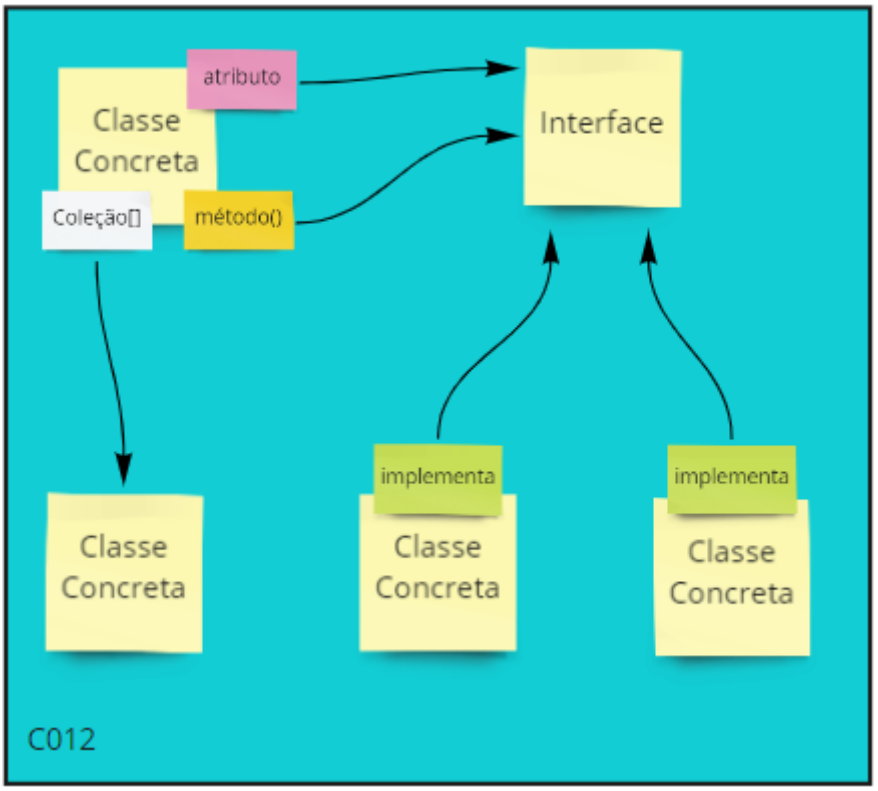

Classe
Concreta
Coleção[]
Classe
Concreta
método()
Classe
Concreta
C007

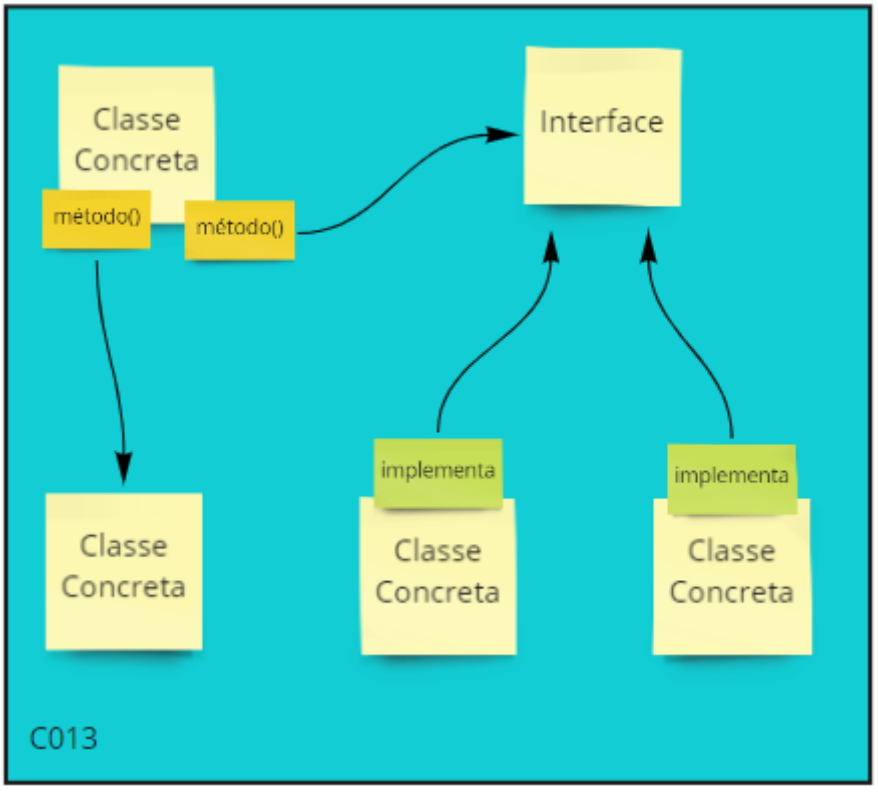

Classe
Concreta
atributo
método()
Classe
Concreta
Classe
Concreta
Classe
Concreta
C008

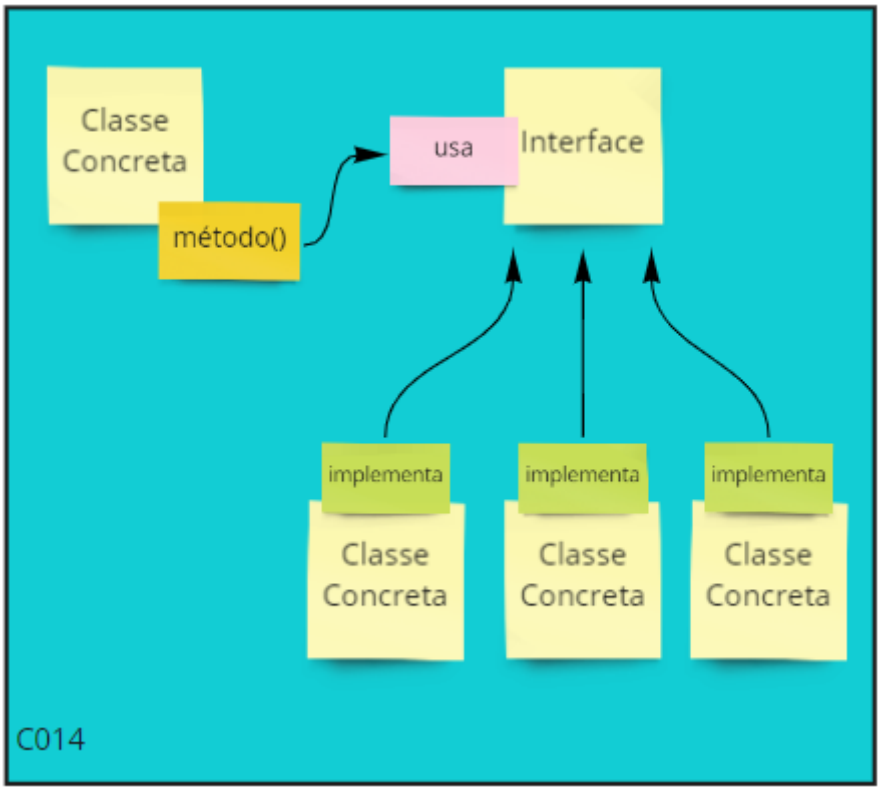

Classe
Concreta
método()
Classe
Concreta
método()
método()
Classe
Concreta
Classe
Concreta
C009

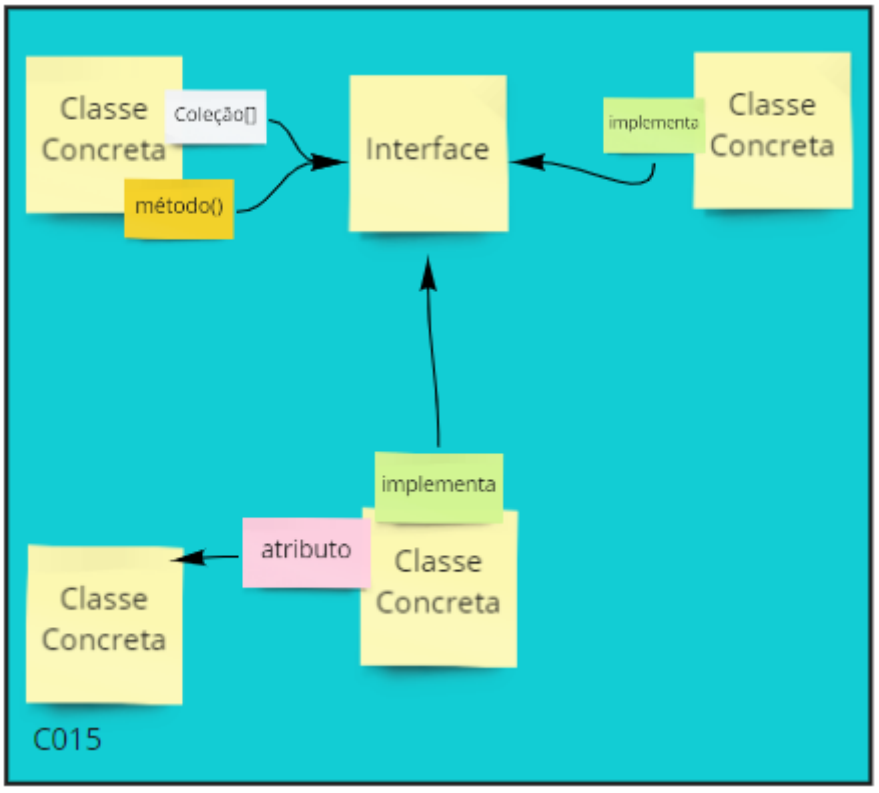

Classe Concreta
atributo
Classe Concreta
método()
Classe Concreta
atributo
Classe Concreta
C010


atributo
Classe Concreta
método()
Classe Concreta
método()
Classe Concreta
Classe Concreta
C011


atributo
Classe
Concreta
Coleção[]
método()
Interface
implementa
implementa
Classe
Concreta
Classe
Concreta
Classe
Concreta
C012


Classe
Concreta
método()
método()
Interface
implementa
implementa
Classe
Concreta
Classe
Concreta
Classe
Concreta
C013





Conjunto de cartas de Contrato construídas nessa versão do jogo.

Apêndice 16 - Manual

Manual do Jogo OO

OBJETIVO DO JOGO:

Os jogadores irão construir uma solução única para os desafios propostos durante as três rodadas do jogo, refatorando o código(mudando sua estrutura) sempre que for preciso para agregar os novos problemas, os quais serão adicionados a cada rodada, tendo o cuidado para manter a solução compatível com os problemas anteriores.

FLUXO DO JOGO:

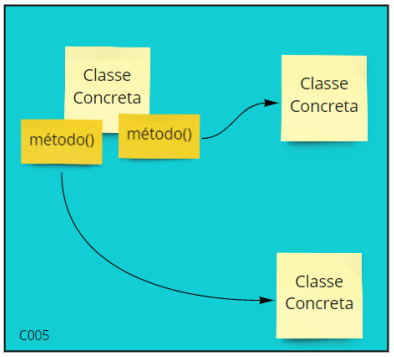
Uma partida possui 3 rodadas, onde o jogador pegará uma carta do baralho de contratos e construirá a solução dela com as cartas de programação que estarão livres na mesa do jogo.
No início de cada rodada, um jogador deve ir no baralho de contratos pegar uma carta e revelar o contrato que irá implementar na solução atual para ambos os jogadores.
No final das 3 rodadas, vence o jogador com a menor pontuação em Acoplamento total da solução. Em caso de empate, vence o jogador que tiver a menor pontuação total de complexidade e se o empate persistir, ambos os jogadores são considerados vencedores.

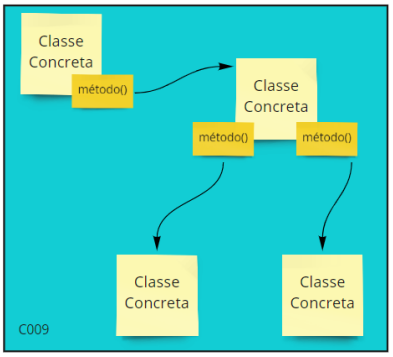
Em cada rodada cada jogador pode fazer as seguintes ações:

1. Revelar a carta de contrato.
2. Pegar cartas de programação.
3. Refatorar o código.
4. Fim da rodada.
5. Fase de pontuação (apenas no final da rodada 3).

1. Revelar carta Contrato.

Essa é uma ação obrigatória a qual o jogador escolhido(sugestão é tirar pedra papel tesoura para decidir) deve embaralhar e pegar um contrato do baralho de contratos e revelar a carta obtida. O jogador deve deixar visível para todos a carta de Contrato pega no início da partida, justo com as das outras cartas Contratos das rodadas anteriores.

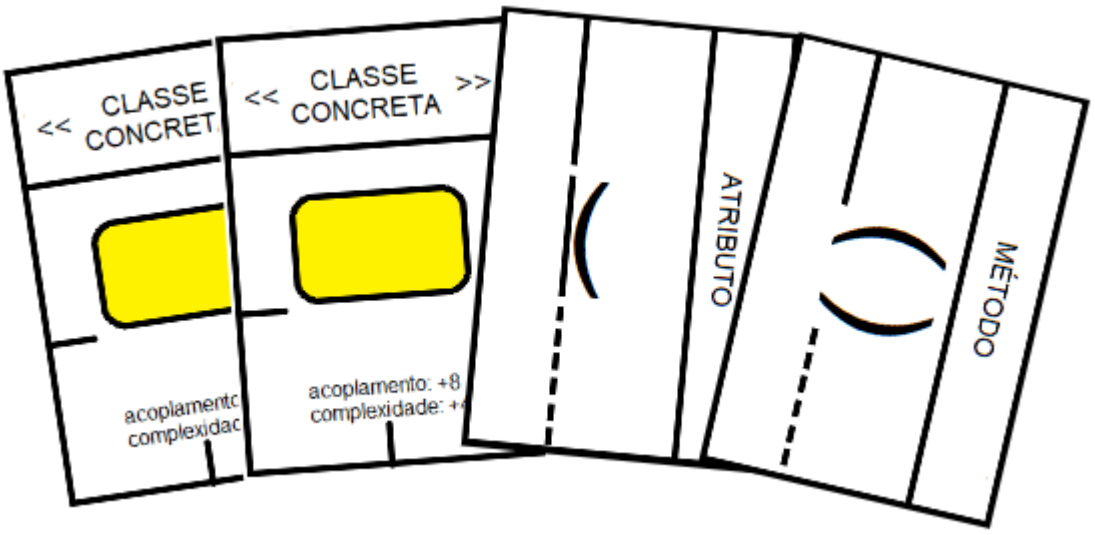




Exemplo da segunda rodada, com as duas cartas de contrato visíveis para ambos os jogadores na mesa do jogo.

2. Pegar cartas de programação.

   Após a revelação da carta de Contrato, cada jogador pode pegar livremente as cartas de programação que serão usadas para construção da sua solução. Os jogadores podem pegar quantas cartas quiserem, mas não podem encerrar a rodada com cartas na mão ou fora da solução.



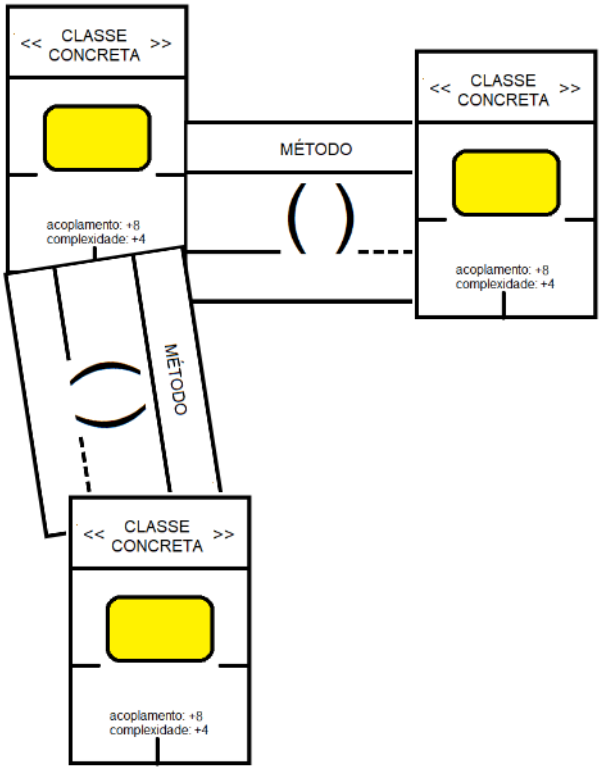
Exemplo: Para a primeira rodada, um dos jogadores pegou duas cartas de Classe Concreta, uma carta de Atributo e uma carta de Método para montar sua solução.

Obs.: Apesar do exemplo acima mostrar o jogador pegando um conjunto de cartas de uma vez, os jogadores estão livres para pegar quantas cartas necessárias até o fim da rodada, desde que quando a rodada terminar, todas as cartas estejam sendo usadas na solução.

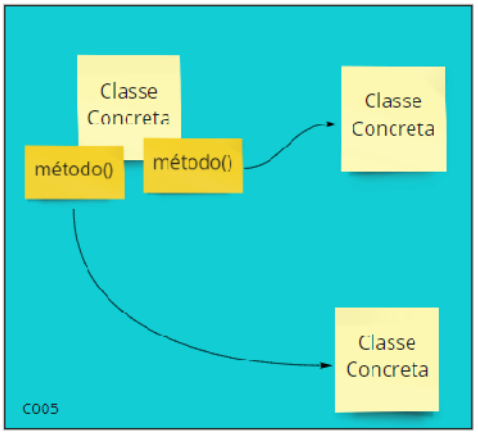
3. Refatorar o código.

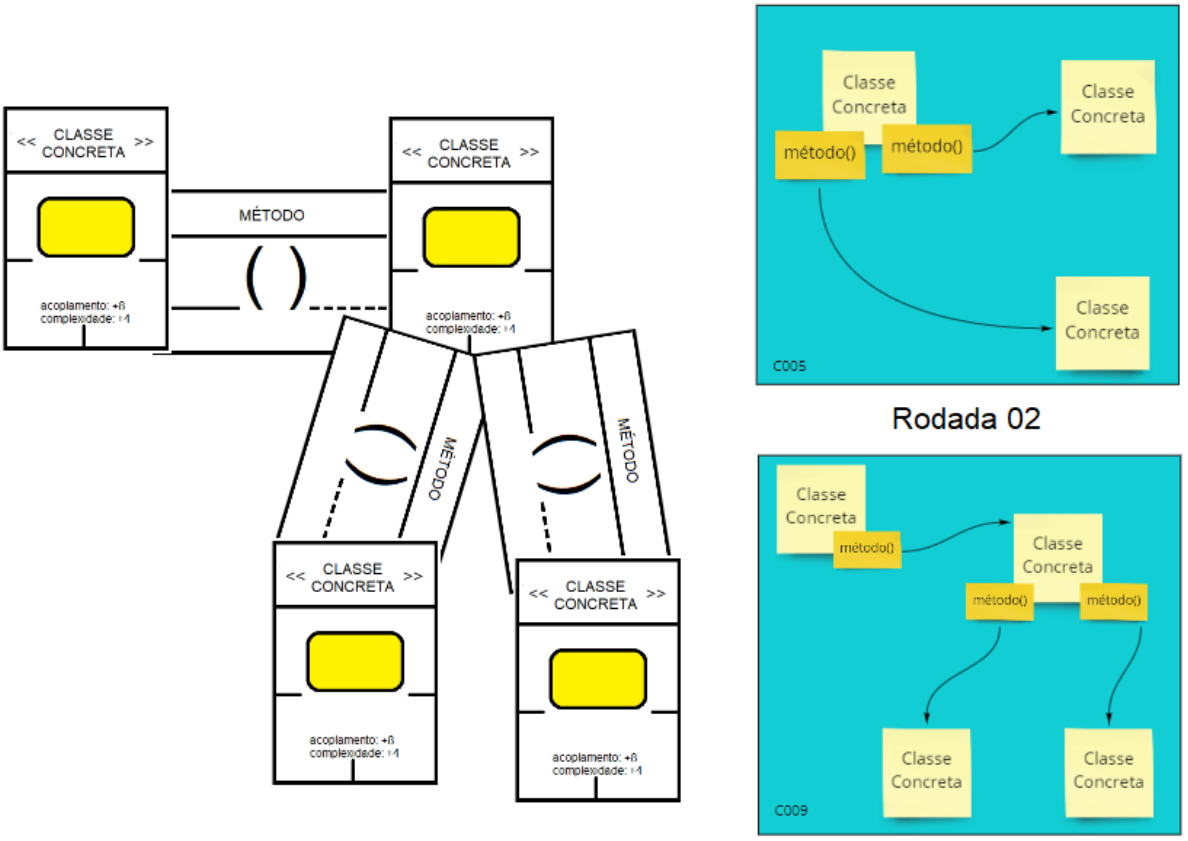
   Essa ação só é possível ser realizada após a primeira rodada, pois os jogadores já terão uma solução pronta vinda da primeira rodada e podem refatorar a sua solução com

remoção de cartas, adicionando novas cartas ou reorganização da estrutura da solução como um todo.
Ao realizar a ação de Refatorar o Código, o jogador deve tomar cuidado para não alterar a solução dos contratos anteriores de forma que se torne uma solução invalida para algum. Se isso acontecer, a solução é considerada inválida, mesmo que ela resolva o contrato atual.
A solução deve ser única e ser considerada válida(seção 6) para resolver o contrato atual e todos os outros contratos das rodadas anteriores.



Rodada 01

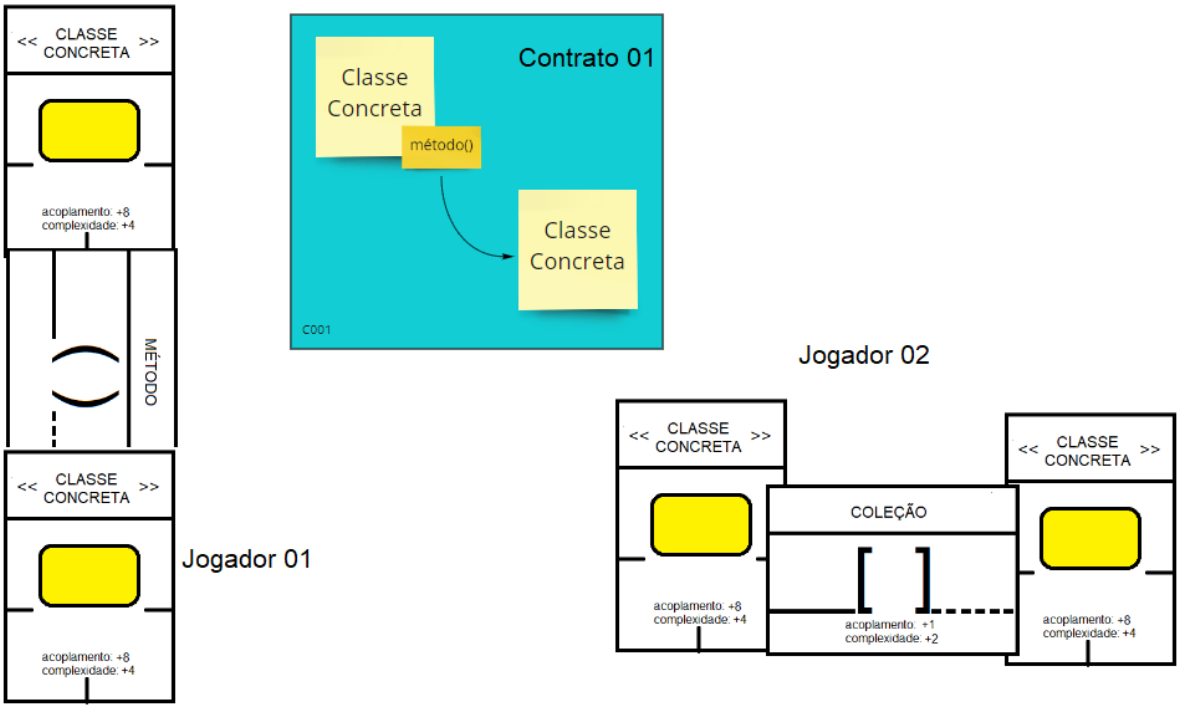




Exemplo: Na primeira rodada o jogador construiu sua solução com três cartas de Classe Concreta e duas cartas de Método. Na segunda rodada ele precisou refatorar sua solução adicionando mais uma carta de Classe Concreta e uma carta de Método, o jogador também mudou a estrutura do seu código para melhor visualização da solução, fazendo as mudanças necessárias para solucionar o novo contrato mantendo a solução para o antigo.

4. Fim da rodada.

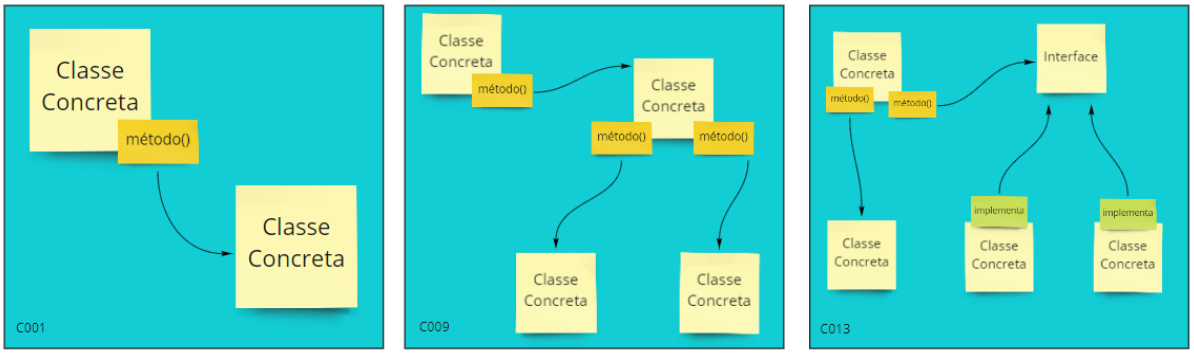
No fim de cada rodada, cada jogador deve conferir sua solução e a do adversário para verificar se é uma solução válida para o contrato atual e para as demais (caso esteja na rodada 2 ou 3). Se for identificado que uma solução não é válida, o jogador deverá refazer a solução para poder prosseguir para a próxima rodada. Caso o jogador não consiga construir uma solução válida para seu contrato, o adversário é considerado vencedor da partida.

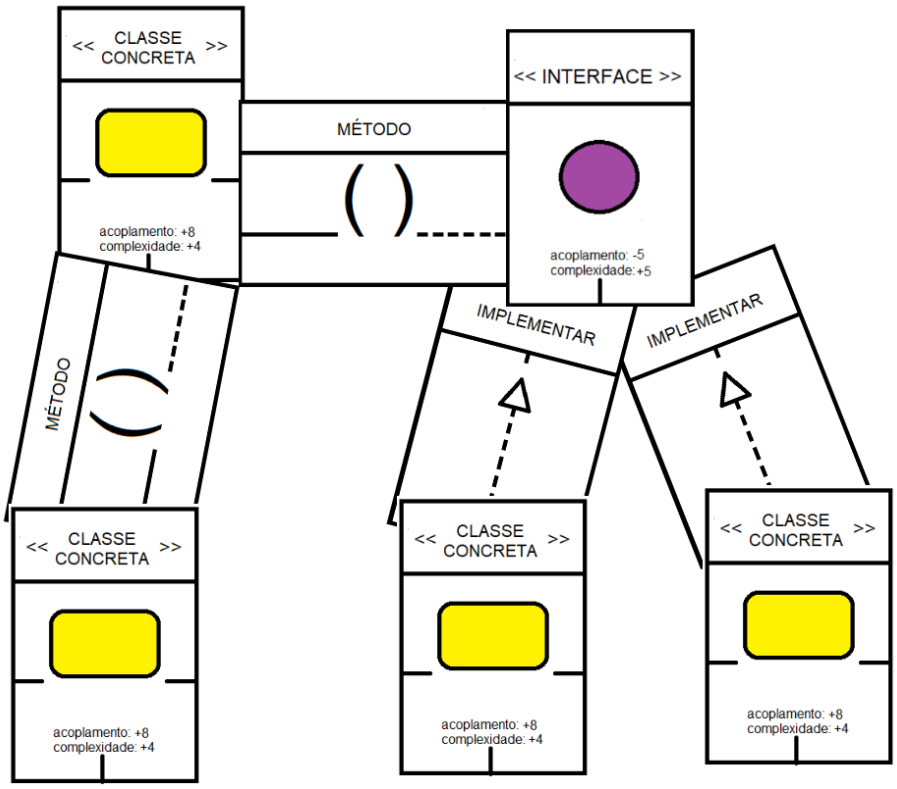


Exemplo: No final da primeira rodada, o jogador 01 construiu uma solução válida para a carta de Contrato, porém o jogador 02 não conseguiu, ele acabou não colocando a carta Método, pedida no contrato, em vez disso ele colocou uma carta Coleção, tornando a solução inválida.

5. Fase de pontuação.

Após o encerramento da terceira rodada, é iniciada a fase de pontuação para decidir o jogador vencedor da partida.
Para determinar o vencedor, primeiro deve verificar se a solução de cada jogador é uma solução válida para todas as cartas Contrato que ele possui. Após a verificação das soluções, é calculado a pontuação total de Acoplamento (a pontuação é calculada somando todos os valores de Acoplamento presente nas cartas usadas para construção da solução) e o vencedor é o jogador que tiver o menor valor de Acoplamento. Em caso de empate é usada a métrica de Complexidade, seguindo o mesmo cálculo da de Acoplamento, somando todos os valores de Complexidade usada na solução. Se o empate persistir, ambos os jogadores desenvolveram a melhor solução possível para os seus contratos e todos os jogadores são considerados vencedores.





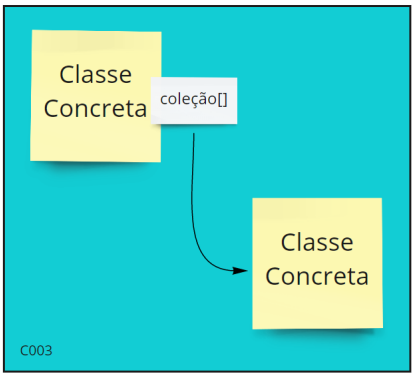
Exemplo: Na solução final de cima, o jogador usou um total de 4 cartas Classe Concreta (total de Acoplamento: +32 e Complexidade: +16), 1 carta de Interface (total de Acoplamento: -5 e Complexidade: +5), 2 cartas de Método (total de Acoplamento: 0 e Complexidade: 0) e duas cartas de Implementar (total de Acoplamento: 0 e Complexidade: 0). Tendo um total de Acoplamento igual a 27 e Complexidade igual a 21.
Para o jogador adversário conseguir vencer, ele precisa ter construído uma solução que resolva suas cartas Contrato com um acoplamento menos que 27 ou um acoplamento igual a 27 e a complexidade menor que 21.
Obs.: Também é possível conseguir a vitória se o jogador adversário ter construído sua solução com 27 de Acoplamento e 21 de Complexidade, nesse exemplo de partida. Nesse caso, ambos os jogadores são considerados vencedores.

6. CONSTRUÇÃO DE SOLUÇÕES VÁLIDAS

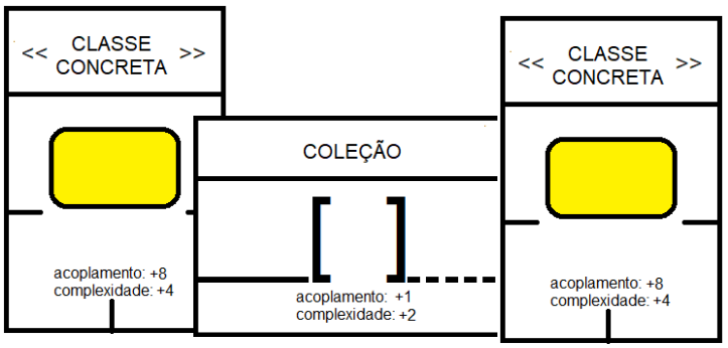
Para a resolução das cartas de Contrato que os jogadores receberão durante cada rodada, é preciso construir uma solução válida, usando as cartas de Programação, seguindo as

regras individuais de cada carta, e respeitando o diagrama presente na carta de Contrato recebida.

Exemplo 1:

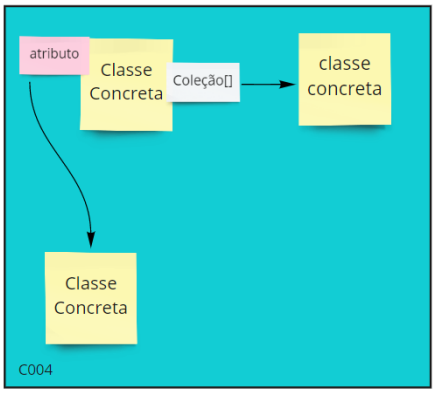


No exemplo acima temos uma classe concreta ligada a outra classe concreta através de uma coleção.
Uma solução válida para esse contrato é:

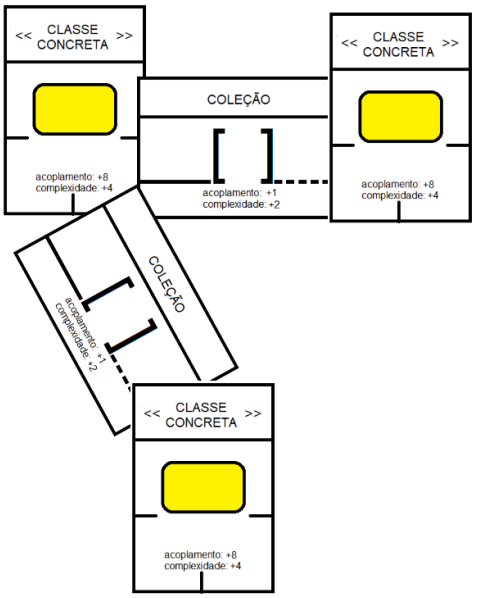


Na solução do exemplo 1 temos duas cartas de Classe Concreta ligadas por uma carta de Coleção.

Exemplo 2:

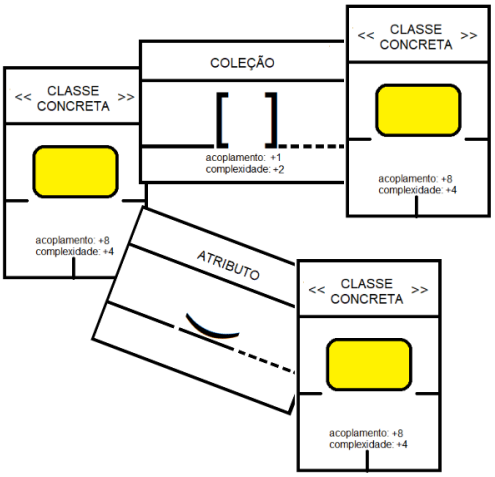


No contrato acima, temos um diagrama mostrando uma classe concreta amarela ligada a classe concreta azul através de uma coleção e a uma classe concreta laranja por meio de um atributo.

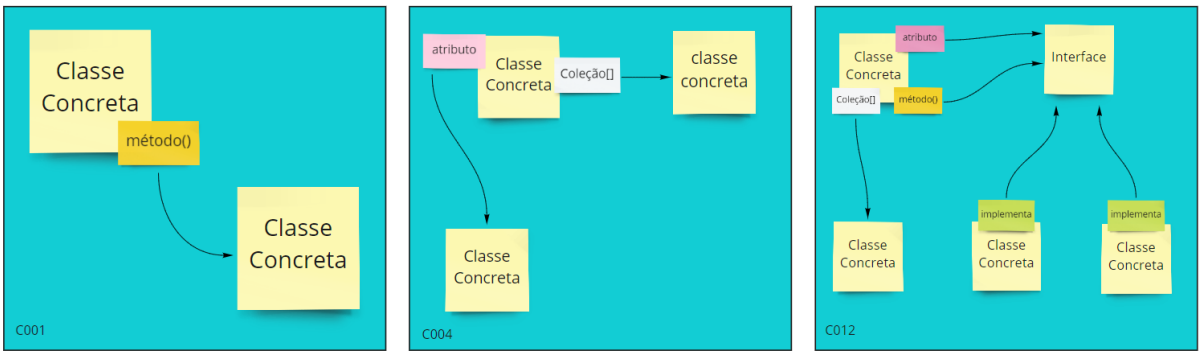


A Solução do lado não pode ser considerada uma solução válida para o contrato, pois apesar de apresentar 3 classes concretas como no diagrama do contrato, a ligação de uma classe concreta a outra classe concreta por meio de um atributo simples, não existe.

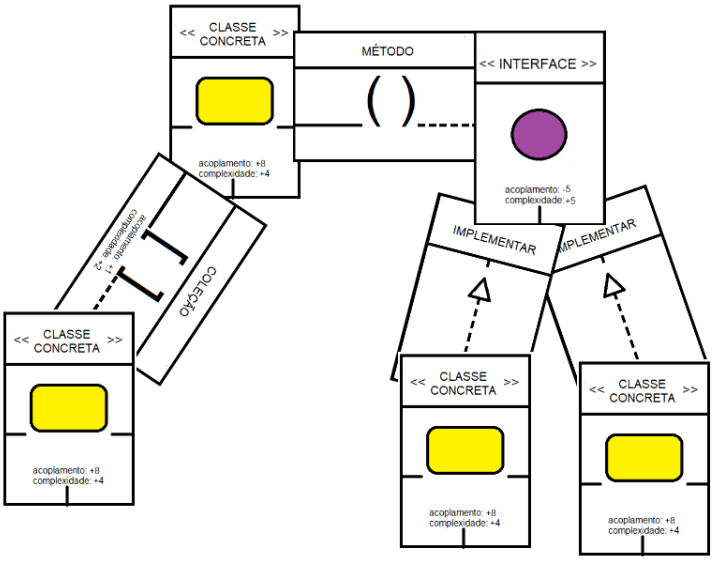
Essa solução é válida para a carta de Contrato do exemplo 2, pois apresenta todas as classes concretas pedida no diagrama e respeitando as ligações por meio de coleção e de atributo.



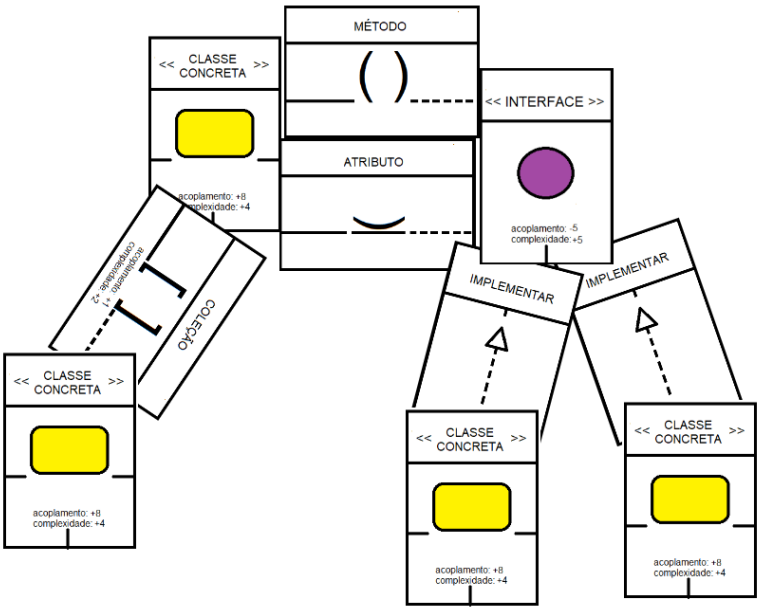
Exemplo 3:



No exemplo acima, temos um conjunto de 3 contratos e uma solução válida deve servir para todos os contratos.



A solução acima não é considerada uma solução válida. Apesar de apresentar a solução do primeiro contrato (classe concreta ligada a outra classe concreta por meio de um método) e ter todas as classes concretas do diagrama do contrato 3, a solução não apresenta uma classe concreta ligada a outra por meio de atributo pedido no diagrama 2 e 3.



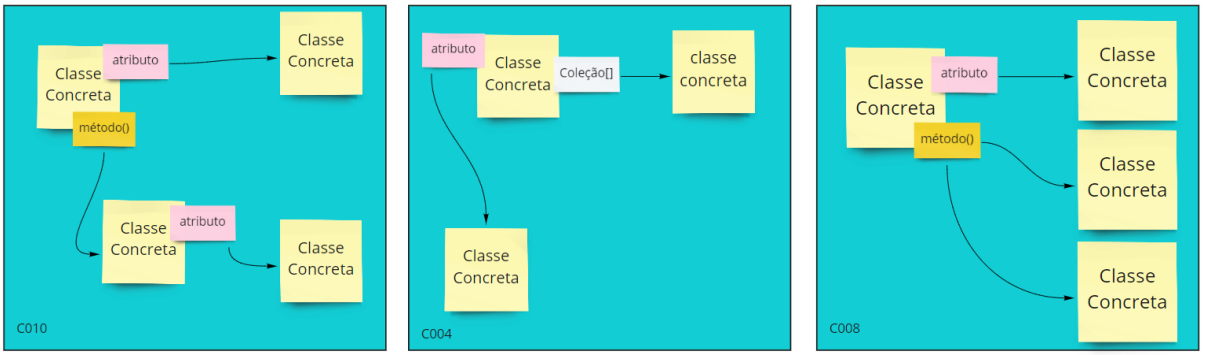
Acima temos uma solução que resolve os 3 contratos, pois apresenta todas as ligações anteriores e a carta Atributo que está presente nos diagramas 2 e 3.

CARTAS DE CONTRATO

As cartas de Contrato são os desafios que os jogadores terão que resolver durante as rodadas. O jogo contém 15 contratos diferentes, com estruturas semelhantes, que estarão

em um único baralho na mesa com a face para baixo, que deverá ser embaralhado no início da partida.

Cada contrato tem a representação dos elementos obrigatórios em sua construção, como a quantidade de cartas de Classe Concreta e a ligação entre elas.



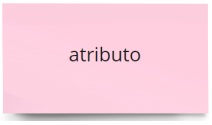
Podemos ter a presença das seguintes estruturas:

Classe Concreta:
É a representação da carta Classe Concreta.

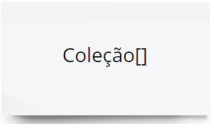
Interface:
É a representação da carta Interface.

Método:
É a representação da carta Método. Ela indica que a ligação entre as extremidades da seta é feita por um método.

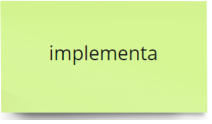
Atributo
É a representação da carta Atributo. Ela indica que a ligação entre as extremidades da seta é feita por um atributo.



Coleção
É a representação da carta Coleção. Ela indica que a ligação entre as extremidades da seta é feita por uma coleção.

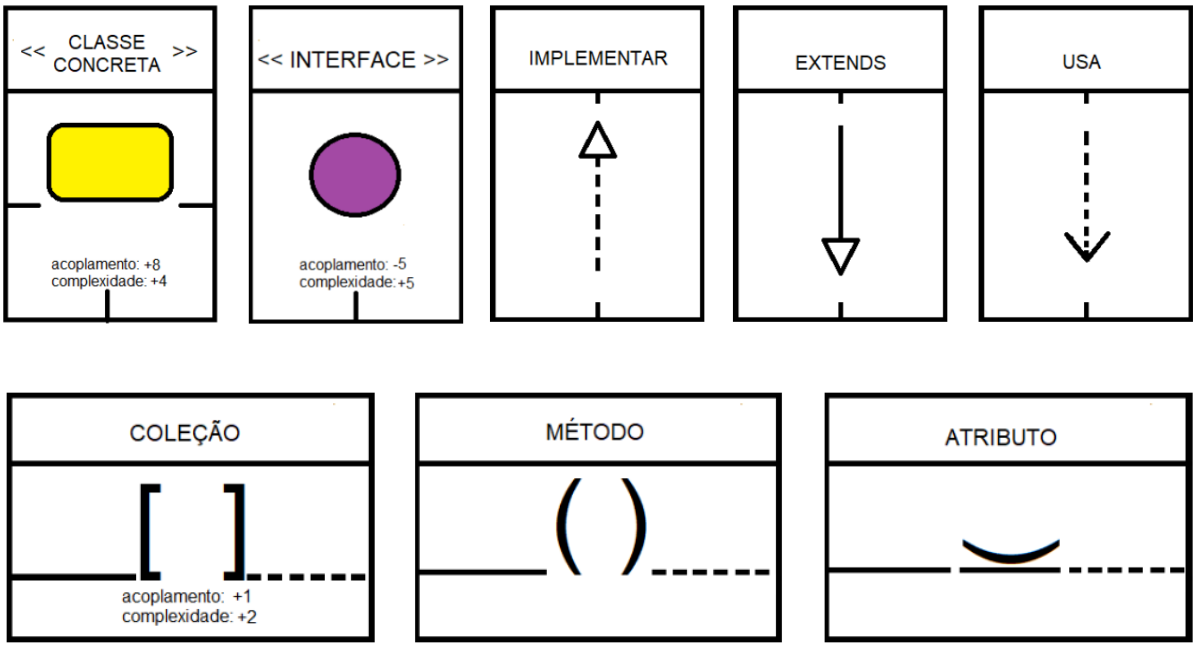


Implementa
É a representação da carta Implementar. Ela indica que a ligação entre as extremidades da seta é feita por implementação.



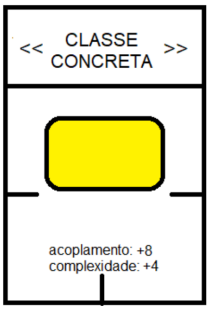
CARTAS DE PROGRAMAÇÃO

Para construção das soluções dos contratos, são usadas as cartas de Programação. Durante o jogo estarão disponíveis 8 tipos de cartas de Programação (Classe Concreta, Interface, Implementar, Usa, Extends, Implementar, Coleção, Método e Atributo) separados em baralhos diferentes com 10 cópias de cada na mesa do jogo.

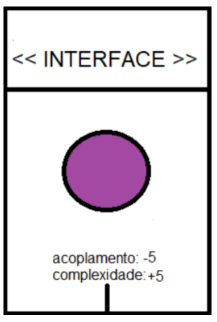


Descrição das cartas de Programação:

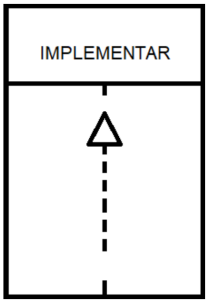
Classe Concreta:
Representação da classe concreta presente no diagrama dos contratos. Cada carta Classe Concreta tem nível de Acoplamento +8 e Complexidade +4, que deverão ser considerados no cálculo final do Acoplamento e Complexidade.



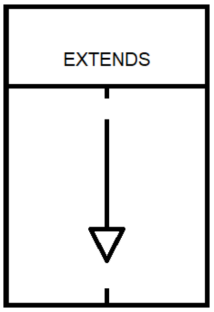
Interface:
Representa a interface no diagrama dos contratos. Ela pode ser usada independentemente do diagrama do contrato (desde que cumpra os requisitos). Cada carta Interface tem nível de Acoplamento +5 e Complexidade -5, que deverão ser considerados no cálculo final do Acoplamento e Complexidade.



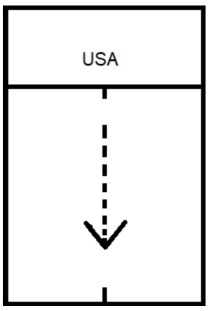
Implementar:
Representa a ligação entre uma Classe Concreta e uma Interface no diagrama dos contratos. Essa carta só pode ser usada para ligar uma carta de Classe Concreta a uma carta de Interface. Cada carta Implementar não tem nível de Acoplamento e Complexidade, e não deverão ser considerados no cálculo final do Acoplamento e Complexidade.



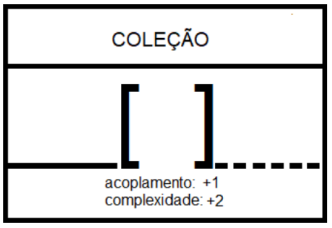
Extends:
A carta Extends não tem representação no diagrama do contrato. Essa carta só pode ser usada para ligar uma carta de Classe Concreta a outra carta de Classe Concreta em indicação de herança. Cada carta Implementar não tem nível de Acoplamento e Complexidade, e não deverão ser considerados no cálculo final do Acoplamento e Complexidade.



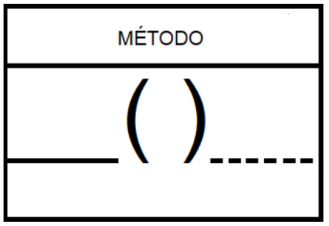
Usa:
Representa a função de usar no diagrama do contrato. Essa carta só pode ser usada para indicar uso de uma carta de Classe Concreta a outro tipo de carta. Cada carta Implementar não tem nível de Acoplamento e Complexidade, e não deverão ser considerados no cálculo final do Acoplamento e Complexidade.



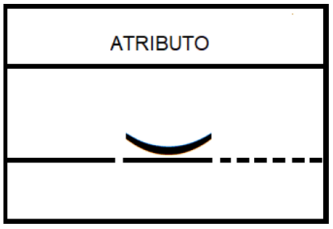
Coleção:
Representação da Coleção presente no diagrama do contrato. Essa carta só pode ser usada para ligar uma carta de Classe Concreta a outra Classe Concreta ou Interface, demonstrando a ligação entre elas. A parte com a minha completa da esquerda, deve estar vindo da carta de origem, e a linha pontilhada da direita deve estar apontada para a carta que vai receber a ligação. Cada carta Coleção tem nível de Acoplamento +1 e Complexidade +2, que deverão ser considerados no cálculo final do Acoplamento e Complexidade.



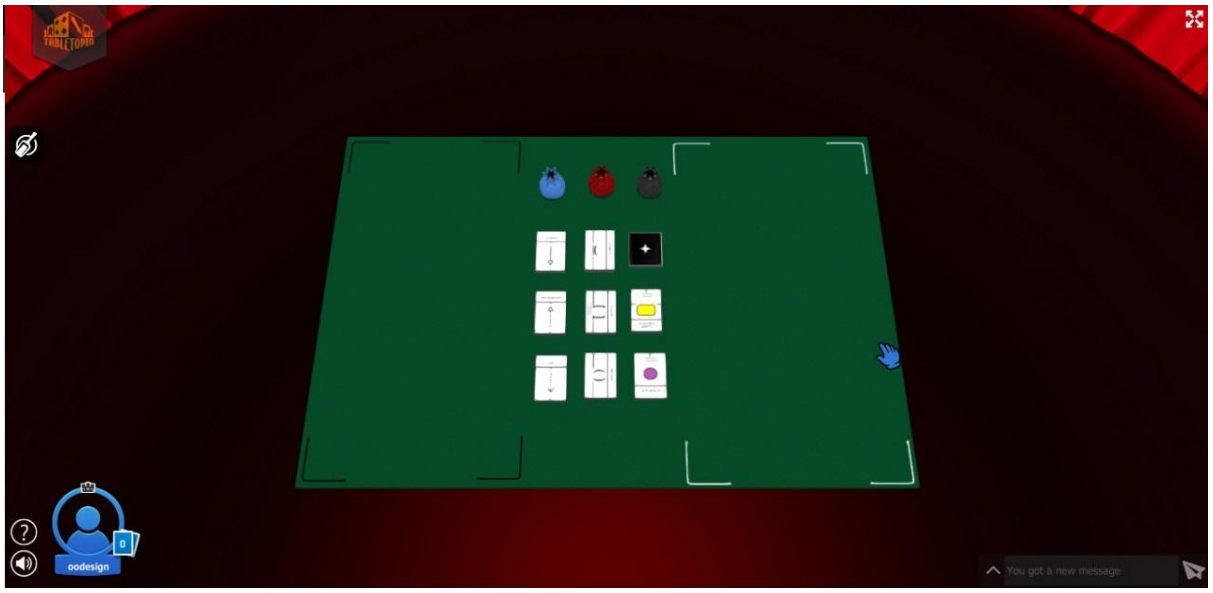
Método:
Representação do Método presente no diagrama do contrato. Essa carta só pode ser usada para ligar uma carta de Classe Concreta a outra Classe Concreta ou Interface, demonstrando a ligação entre elas. A parte com a minha completa da esquerda, deve estar vindo da carta de origem, e a linha pontilhada da direita deve estar apontada para a carta que vai receber a ligação. Cada carta Método não tem nível de Acoplamento e Complexidade, e não deverão ser considerados no cálculo final do Acoplamento e Complexidade.



Atributo:
Representação do Atributo presente no diagrama do contrato. Essa carta só pode ser usada para ligar uma carta de Classe Concreta a outra Classe Concreta ou Interface, demonstrando a ligação entre elas. Cada carta Atributo não tem nível de Acoplamento e Complexidade, e não deverão ser considerados no cálculo final do Acoplamento e Complexidade.



Apêndice 17



Mesa do jogo durante a aplicação