Jogo: Suponha um jogo de par ou ímpar. O jogo inicia com um jogador escolhendo se deseja par ou ímpar e com os jogadores escolhendo seus valores e apresentando-os. A soma dos valores determinará o resultado do jogo e uma resposta deve ser enviada a cada jogador informando quem ganhou e quem perdeu.

Caso de uso: jogar par ou ímpar

1) Cada jogador se cadastra informando seus dados quais sejam nome, idade, sexo e cidade.
2) Ao identificar um número par de jogadores conectados o sistema sorteia dois jogadores para o início de um novo jogo.
3) Cada jogador informa se aceita ou não o oponente para o par ou ímpar.
4) Em caso de ambos aceitarem, é criado um novo jogo no sistema e cada jogador deverá informar qual o tipo de valor que ele deseja (par ou ímpar).
    a. Em caso de empate na escolha o sistema deverá atender aquele jogador que selecionou primeiro.
5) Após uma mensagem de início, cada jogador deverá informar o valor escolhido e aguardar o resultado.
6) Após o último jogador apresentar seu valor, o sistema calcula o valor total e decide quem foi o vencedor.
7) Para ambos jogadores o sistema deverá notificar o resultado do jogo.

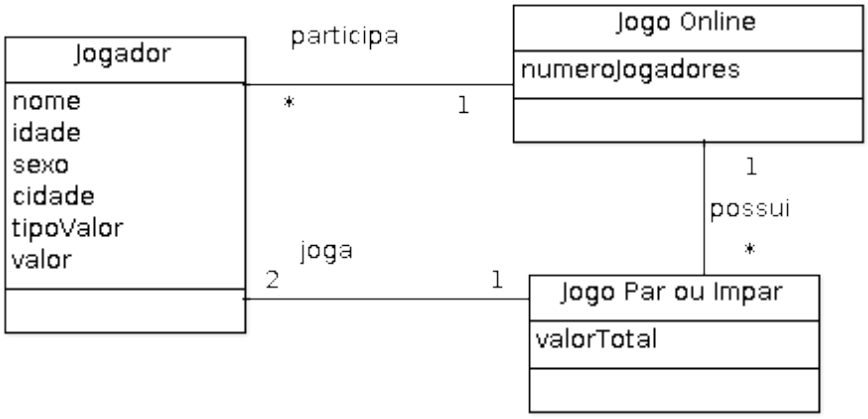
- A qualquer momento um jogador pode solicitar o abandono do jogo.

Resolução:

1) Fazendo uma análise textual da descrição do caso de uso buscando substantivo e frases nominais obtém-se um conjunto de termos (sublinhados na descrição do caso de uso acima).

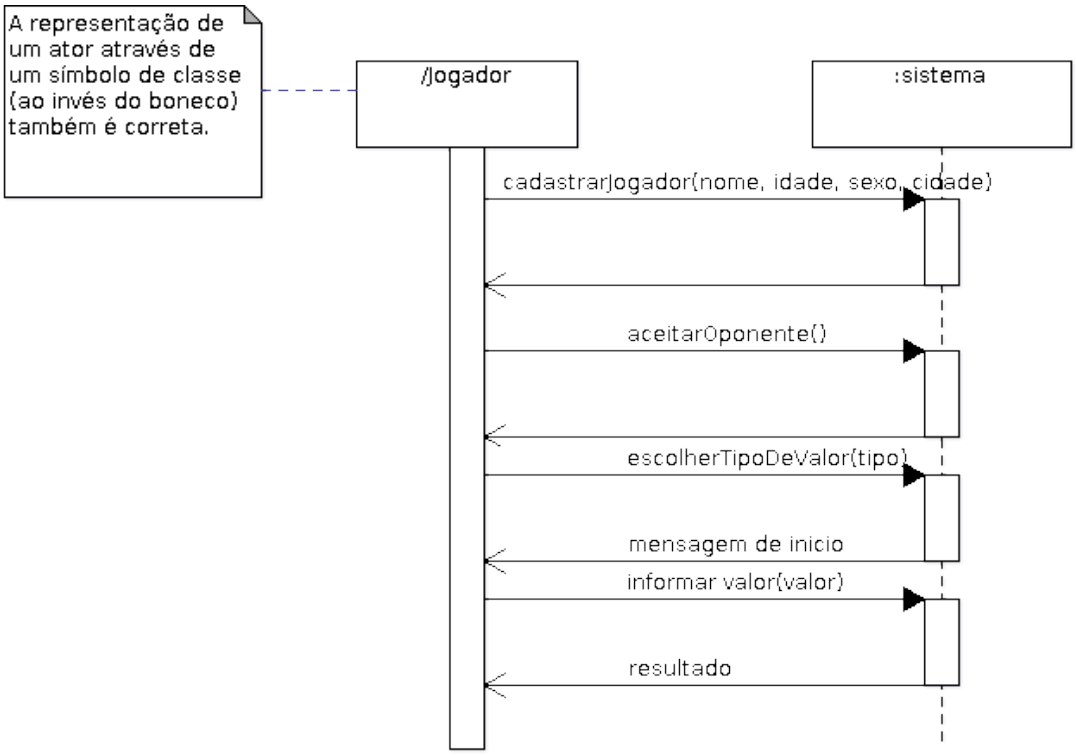
Para esse conjunto de termos devemos definir quais termos são sinônimos, quais termos vão ser considerados como conceitos pertencentes ao domínio (termos que não podem ser descritos com um valor apenas) e termos que serão utilizados para descrições dos conceitos do domínio (termos que podem ser descritos com apenas um valor).

Nesse caso chegamos ao seguinte modelo de dominio.



2) Ainda com base na descrição do caso de uso, podemos criar Diagrama de Seqüência de Sistemas (DSS). Para isso “varremos” novamente o caso de uso em busca de evidências de interação entre atores e sistema. Quando isso ocorrer é possível representar tal interação através de uma operação do DSS, também chamada de **operação de sistema**.

Desse modo o DSS para o caso de uso em questão fica sendo:



3) Uma vez que o DSS foi definido é hora de descrever os contratos de operação para cada uma das operações de sistema. Como sabemos, essa descrição é realizada em termos de instâncias (de conceitos) criadas, alteração de estados e associação entre conceitos.

Abaixo estão definidos alguns contratos de operação. Outros contratos foram deixados em branco intencionalmente como exercícios para vocês.

Contratos de operação

Operação 1: cadastrar (*nome*, idade, sexo, cidade)

Referência Cruzada: jogar par ou ímpar

Pré-condições:

- Existe uma instância **jo** de **JogoOnline**.

Pós-condições:

- Uma instância **j** de **Jogador** é criada;
- **J.nome** recebe o valor de *nome*;
- **J.idade** recebe o valor de *idade*;
- **J.sexo** recebe o valor de *sexo*;
- **J.cidade** recebe o valor de *cidade*;
- **J** é associado com a instância **jo**.
- **Jo.numeroJogadores** é incrementado em uma unidade.

Operação 2: aceitarOponente (escolha)

Referência Cruzada: Jogar Par ou Ímpar

Pré-condições:

- Existem instâncias **j1** e **j2** de **jogador**.

Pós-condições:

- Existe uma instância **jpi** de **jogoParOuImpar** criada.
- **Jpi** está associada com **j1** e **j2**.
- **jo.numeroDeJogadores** é diminuído em 2 unidades.

Operação 3: solicitarTipoValor(tipo)

Operação 4: informarValor(valor)

Operação 5: abandonarJogo()

Referência Cruzada: Jogar Par ou Ímpar

Pré-condições:

- Deve existir uma instância **j** de **jogador**.
- Deve existir uma instância **JPI** de **JogoParOuImpar**.
- Deve existir uma instância **JO** de **JogoOnline**.

Pós-condições:

- A associação entre **JPI** e **JO** é desfeita.
- A instância **JPI** é eliminada.
- **JO.númeroDeJogadores** é incrementada em uma unidade.

Atenção!!! Até esse ponto o que estamos fazendo é representar como os **conceitos do domínio** se relacionam (através de instâncias, alterações de estados e associações formadas) durante a execução de cada uma das operações do caso de uso. Não estamos tratando de software ainda! A prova disso é que tais contratos foram descritos com base nos elementos conceituais presentes no modelo de domínio.

4) Chega a hora então de fazer a transição entre Análise OO e Projeto OO. Queremos começar a criar nossa solução com base nesses artefatos de análise que já criamos. Estamos no momento de transição entre investigação do problema e proposta da solução. Começaremos a trabalhar no **modelo de projeto**.

Lembrete: o modelo de projeto é o conjunto de diagramas UML que descrevem a nossa solução em seus diferentes aspectos (estáticos e dinâmicos). Podemos então criar diagramas de classes para representar as classes de software (e suas associações) que compõem o projeto, diagramas de seqüência e de colaboração para representar o aspecto dinâmico ou comportamental das classes, etc...

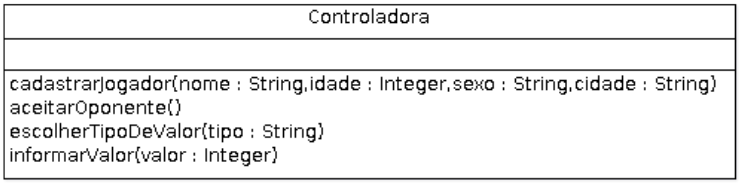
O que buscaremos a partir desse momento é criar um projeto de software que atenda aos requisitos elicitados nos artefatos da análise (previamente criados, tais como modelo de domínio, modelo de casos de uso, especificação suplementar, etc...). No entanto não basta que esse projeto seja correto (i.e. atenda aos requisitos): é necessário que ele seja uma boa solução em termos estruturais, que ele atenda a bons princípios de projetos para facilitar manutenções e evoluções futuras. Em uma frase: estamos querendo um projeto bom e correto.

Para conseguirmos um projeto correto devemos sempre avaliá-lo perante os artefatos da análise. Para conseguirmos um bom projeto, devemos saber

distribuir as responsabilidades entre objetos, sempre avaliando as possíveis soluções em termos de princípios de projeto como coesão e acoplamento. Para isso temos os princípios GRASP como guias para distribuição de responsabilidades entre objetos. Há nove princípios GRASP que devemos saber, quais sejam:

- Coesão alta
- Acoplamento baixo
- Especialista na informação
- Polimorfismo
- Criador
- Controlador
- Indireção
- Variações protegidas
- Invenção pura

Pelo padrão Controlador conseguimos definir um objeto em nosso projeto: é o objeto responsável por capturar os eventos de interface e tratá-los. Tal objeto é então responsável por representar a execução do caso de uso em seus métodos e atributos. Desse modo podemos adicionar a classe Controladora desse caso de uso em nosso projeto.



Note que cada método da classe controladora é uma operação de sistema (descrita no DSS). Portanto, temos que certificar que ao final da execução de cada um desses métodos o projeto de software atenda aos contratos de

operação de sua operação correspondente. Desse modo estamos caminhando rumo a um projeto correto, ou seja, que atenda ao que foi elicitado nos artefatos de análise.

Tomando a primeira operação do DSS e que já está presente na classe controladora vemos que em resumo ela tem a responsabilidade de cadastrar um jogador no jogo online, conforme o contrato de operação descreve.

Nota: o contrato de operação descreve o estado que o sistema vai estar ao final da operação, mas entre a chamada da operação e sua finalização uma série de ações vai ocorrer, objetos vão colaborar uns com os outros, eles vão comunicar através de mensagens entre si, e ao final o contrato deverá ser respeitado.

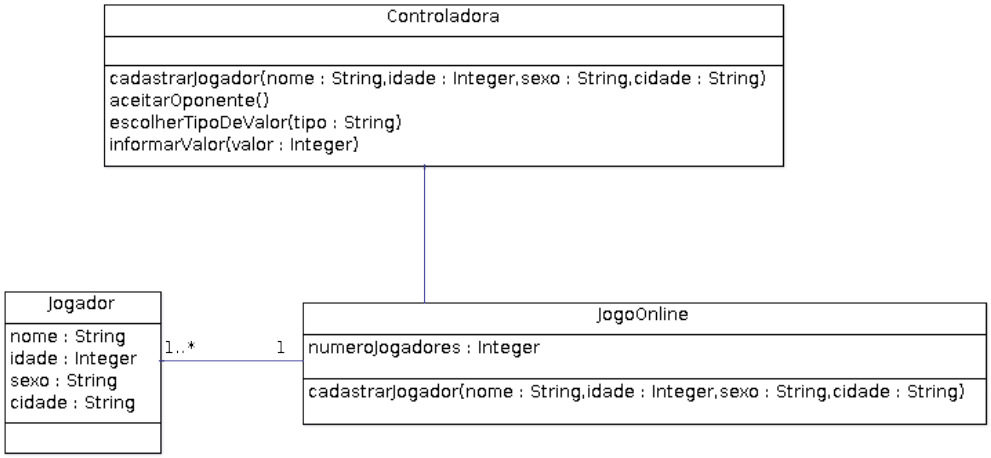
Portanto aplicando os princípios GRASP devemos ser capazes de avançar o nosso projeto com objetos e interações entre esses objetos para a execução da operação, respeitando seu contrato.

Por especialista na informação, temos que a classe Controladora conhece a classe JogoOnline. Isso faz com que exista uma associação direcionada de Controladora para JogoOnline.

Se olharmos para o Modelo de Domínio constatamos que há uma relação um-para-muitos entre JogoOnline e Jogadores. Isso nos sugere a utilização do princípio GRASP Criador. Assim JogoOnline fica sendo o responsável por instanciar o Jogador (os dados de Jogadores foram passados pela interface para a Controladora, que pode então passá-los para JogoOnline realizar a instanciação). Desse modo, sabemos que a classe JogoOnline vai ter uma associação um-para-muitos com Jogador. Podemos então adicioná-las ao projeto.

Nesse ponto vale uma pausa: onde deverão ser armazenados os dados que foram passados pela controladora para JogoOnline? Por especialista na Informação percebe-se que eles devem ser armazenados em Jogador. Portanto vamos definir os atributos de Jogador e consequentemente definir algumas responsabilidades de conhecer dessa classe. Podemos ver que essa distribuição é suportada pelo modelo de domínio.

O contrato da operação CadastrarJogador define ainda que deve ser incrementado o número de jogadores conectados. De quem é essa responsabilidade? Novamente, por Especialista de Informação, percebemos que essa responsabilidade é de JogoOnline. Feito isso conseguimos aumentar nosso diagrama de classes para o projeto. Ele fica sendo então igual a:

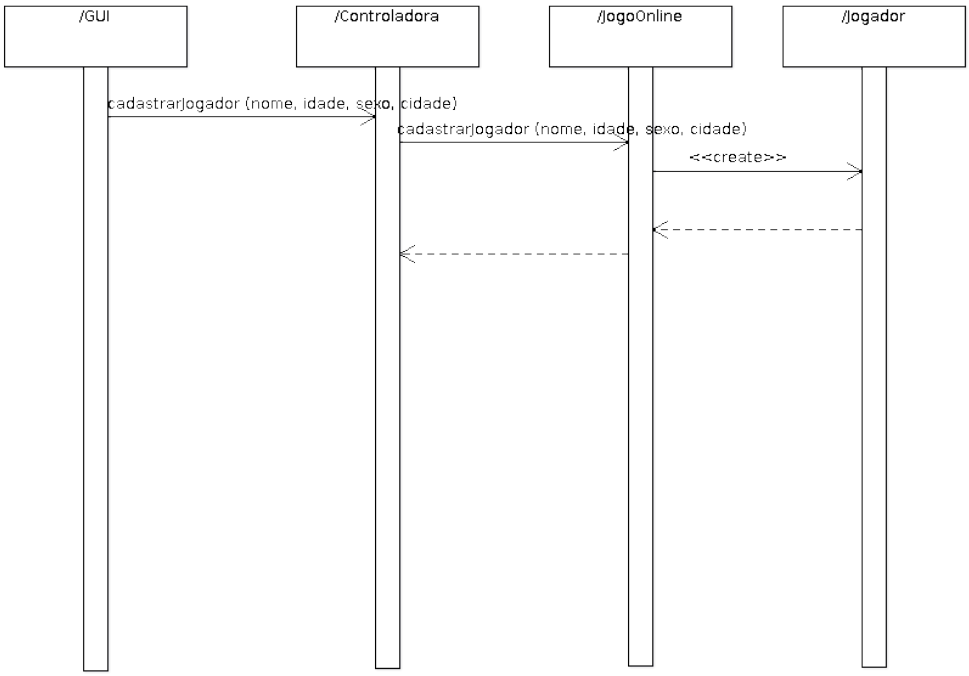


O diagrama acima apresenta o aspecto estático do projeto após avaliar a operação de sistema **cadastrarJogador(...)**. Temos agora que pensar no aspecto dinâmico (ou comportamental) dessa operação. Quando a operação for executada, mensagens serão trocadas entre diferentes objetos e temos que nos preocupar em garantir que, ao final da execução da operação, o contrato de operação de cadastrarJogador(...) seja atendido pelo projeto de software.

Sabemos que:

- a classe **Controladora** é a primeira classe após a interface a receber uma mensagem com os dados informados pelo usuário,
- a classe **Controladora** deve enviar uma mensagem à classe **JogoOnline** contendo os dados do jogador a ser cadastrado.
    - Nesse ponto deve haver então um objeto de JogoOnline instanciado. Mas, e são não tiver? Como fazer? A classe Controladora deve então instanciá-lo!
- A classe **JogoOnline** tem os dados iniciais de **Jogador**, e pelo padrão Criador deve então ser responsável por instanciar o objeto.
- A classe **JogoOnline** tem a responsabilidade de conhecer o número de jogadores online, e por Especialista na Informação tem a responsabilidade de atualizar tal valor.

Com base nisso temos que ser capazes de criar uma cadeia de mensagens entre os objetos de modo e que ao final atenda ao contrato de operação **cadastrarJogador(...)**. Essa cadeia de mensagens pode ser representada através de um diagrama de seqüências ou de colaboração. O diagrama abaixo apresenta uma possível solução.



Esse diagrama mostra algumas coisas sobre o comportamento do projeto quando a mensagem cadastrarJogador(...) é executada. No entanto ela não mostra tudo. Vejamos o que conseguimos compreender desse diagrama e o que não está sendo mostrado.

Podemos ver que a mensagem vinda da interface com usuário é composta dos dados de um jogador. Para evitar a ligação direta entre camada de usuário existe uma classe Controladora responsável por receber tal mensagem e enviá-la à classe de domínio responsável, no caso JogoOnline (note que essa decisão foi tomada com base no padrão Especialista na Informação). A classe JogoOnline recebe os valores vindos como parâmetros e pelo padrão Criador (ela contem os dados iniciais) fica responsável por instanciar Jogador. Algum objeto vai ser retornado e a classe então armazena-o em seu interior (como é feito esse armazenamento? Um vetor de Jogadores? Uma lista? Uma pilha? Enfim, isso não conseguimos ver nesse diagrama e fica a cargo do implementador durante a atividade de codificação). Ao final o método cadastrarJogador(...) fica ainda responsável por atualizar o valor de seu atributo numeroDeJogadores.

Caros alunos, esse foi um exemplo simples para ilustrar a vocês a utilização dos padrões GRASP. Basicamente o que temos que fazer é racionar sobre os

objetos juntamente com o conhecimento de padrões e então dividir as responsabilidades entre os diversos objetos que compõem o nosso software. Como vimos os diagramas capturam aspectos estáticos e dinâmicos do software, e vão sendo complementados à medida que o projeto vai sendo desenvolvido. Isso significa dizer que podemos descobrir uma responsabilidade ainda não capturada ao trabalhar em um diagrama e devemos representá-la nos demais diagramas para manter a consistência do projeto. Exemplo: se ao trabalhar com um diagrama dinâmico identificamos que um objeto B é retornado para o objeto A e vai ser mantido como um atributo em A, devemos então representar essa atribuição no diagrama de seqüências e também representar o atributo no diagrama de classes.

Continuem o projeto desse caso de uso, investigando os outros contratos de operação juntamente com os padrões GRASP e criando os diagramas de Classes e de Seqüência. **Façam isso como exercício e escrevam o raciocínio de vocês para cada aplicação de padrões GRASP.**

Bom trabalho a todos!

Prof. André Lanna