MC322/MC336
Segundo semestre de 2015

Trabalho 2

Professor: Fábio Luiz Usberti (fusberti@ic.unicamp.br)
PED: Rafael Arakaki (rafaelkendyarakaki@gmail.com)

---

1 Objetivo

Estudo e implementação dos conceitos de programação orientada a objetos, abordados em aula, para uma aplicação na área de jogos.

2 Regras do Jogo

As regras do **LaMa** são:

1. Cada jogador inicia com um Herói com trinta pontos de vida. Vence o jogador que reduzir a vida do Herói do oponente a zero.
2. O baralho possui trinta cartas, sendo 22 Lacaios e 8 Magias.
3. O primeiro jogador inicia com 3 cartas, enquanto o segundo jogador inicia o jogo com 4 cartas.
4. Todo o início de turno cada jogador compra uma carta do baralho, de maneira aleatória.
5. No turno i, cada jogador possui $\min(i, 10)$ de mana para ser utilizado em uso de cartas ou poder heróico, não podendo ultrapassar este valor. O limite de mana é acrescido até o décimo turno e então não aumenta mais. **Exceção**: O segundo jogador possui dois de mana no primeiro turno ao invés de um de mana.
6. Existem dois tipos de carta: Lacaios e Magias.
    - Lacaios são baixados à mesa e não podem atacar no turno em que são baixados, apenas no turno seguinte (se ainda estiverem vivos!).
    - Magias possuem efeitos imediatos, ao serem utilizadas causam dano na vida de um único alvo inimigo ou em todos os alvos inimigos (efeito em área). As magias de alvo podem ter como alvo o Herói do oponente ou um dos Lacaios na mesa do oponente. As magias de efeito em área causam dano nos Lacaios do oponente e no Herói do oponente.
7. Não é possível ter mais de sete lacaios em campo.
8. Um lacaio só pode atacar no máximo uma vez por turno, assim como o poder heróico também só pode ser utilizado no máximo uma vez por turno.
9. O jogador pode utilizar, uma vez por turno, o poder heróico de seu Herói ao custo de duas unidades de mana. O poder heróico faz o Herói atacar um alvo qualquer com um de dano. Se o alvo do poder heróico for um Lacaio, o Herói que atacou receberá de dano o ataque do Lacaio alvo.

10. Cada Lacaio pode escolher atacar um alvo por turno. Os possíveis alvos são: o Herói do oponente e os Lacaios em mesa do oponente. Se o Lacaio **x** atacar o Lacaio **y**, o Lacaio **y** tem sua vida diminuída pelo poder de ataque do Lacaio **x**, e o Lacaio **x** tem sua vida diminuída pelo poder de ataque do Lacaio **y**. Se a vida de um Lacaio chegar a zero, ele morre e é retirado da mesa.

11. Se um Lacaio atacar um Herói, a vida do Herói é reduzida pelo poder de ataque do Lacaio mas o Lacaio não sofre nenhum tipo de dano.

12. O jogador pode possuir até dez cartas na mão. Se já possuir dez cartas ao início de seu turno, a nova carta que seria comprada é descartada.

13. Se em um dado turno não for possível a um jogador comprar cartas porque o Baralho já está esgotado, o jogador recebe um de dano. No próximo turno receberá dois de dano, depois três, e assim por diante.

3 Descrição do Trabalho

A atividade proposta para este trabalho será a implementação de um Motor para o jogo **LaMa** (**La**caios e **Ma**gias).

3.1 Classe Motor (abstrata)

O Motor é responsável pelo controle das jogadas de cada jogador e mantém a informação de tudo o que acontece no jogo. Cabe ao Motor verificar a validade das jogadas dos Jogadores e, em caso de jogadas inválidas, reportar um erro correspondente explicitando qual regra que o Jogador violou com sua jogada.

O aluno deverá implementar uma classe que herdará da classe **Motor** (abstrata). Esta classe deverá ser chamada **MotorRA**xxxxxx (onde "xxxxxx" é o RA do aluno). A classe **Motor** possui um construtor, dois métodos abstratos e dois métodos concretos *finais*. São os métodos da classe **Motor**:

- Motor(): Método construtor. Inicializa os atributos da classe. A classe **MotorRA** deverá chamar este construtor através do método super().
- executarPartida() *(abstrato)*: É o método que é executado após o objeto Motor ser instanciado e realiza uma partida entre dois jogadores. Este método deve retornar um objeto da classe **GameStatus** que possui os dados de resultado da partida (quem venceu, se houve algum erro ou não, a jogada que causou erro, a mesa ao fim da partida, etc).
- processarJogada() *(abstrato)*: Neste método deverão ser processadas uma a uma as jogadas que um jogador realizar em um certo turno. É obrigatório que a classe **MotorRA** implemente este método e utilize-o para realizar o processamento das jogadas.
- imprimir() *(final)*: Este método é responsável por imprimir mensagens de "log" no terminal e também em um arquivo (dependendo dos parâmetros verbose e saidaArquivo do Motor).
- gerarListaCartasPadrao() *(final)*: Este método gera uma lista de cartas do baralho padrão LaMa.

A Tabela 1 lista os atributos da classe **Motor**. Os atributos deverão ser utilizados durante a implementação da classe MotorRA. Com exceção dos atributos *verbose* e *saidaArquivo*, todos os atributos possuem escopo *protected* e portanto podem ser acessados diretamente pelas classes herdeiras (como é o caso de MotorRA). É responsabilidade da classe **MotorRA** manipular estes atributos de maneira correta.

Para uma implementação correta deste trabalho é necessário que esses atributos sejam inicializados no construtor através do uso do método *super()*.

Tabela 1: Atributos da classe Motor

Atributo	**Tipo**	**Descrição**
jogador1	Jogador	Classe Jogador do primeiro jogador.
jogador2	Jogador	Classe Jogador do segundo jogador.
baralho1	Baralho	Baralho do primeiro jogador.
baralho2	Baralho	Baralho do segundo jogador.
maoJogador1	ArrayList<Carta>	Cartas na mão do primeiro jogador.
maoJogador2	ArrayList<Carta>	Cartas na mão do segundo jogador.
lacaiosMesa1	ArrayList<Carta>	Cartas lacaio na mesa do primeiro jogador.
lacaiosMesa2	ArrayList<Carta>	Cartas lacaio na mesa do segundo jogador.
vidaHeroi1	int	Vida do herói 1.
vidaHeroi2	int	Vida do herói 2.
manaJogador1	int	Mana do primeiro jogador.
manaJogador2	int	Mana do segundo jogador.
verbose	int	Flag para verbosidade ligada (1) ou desligada (0).
tempoLimitado	int	Flag para limite de tempo 300ms ligado (1) ou desligado (0).
saidaArquivo	FileWriter	Escritor para a saída em um arquivo, ativado (!=null) ou desativado (=null).

4 Demais classes

4.1 Classe Jogador

O Motor deverá se comunicar com a classe **Jogador** através dos métodos construtor Jogador() e **processarTurno()**. O método construtor é executado somente uma vez, fornecendo a mão inicial e a informação se a classe Jogador é primeiro ou segundo jogador na partida. Em seguida serão feitas várias chamadas ao método **processarTurno()** para cada turno daquela partida.

4.2 Classe Carta

A classe Carta é descrita na Tabela 2. Todos os atributos são privados e seus valores podem ser recuperados por meio dos métodos get e set. Por convenção, se o nome de um atributo é composto como vidaMesa, então o método get correspondente se chamará getVidaMesa (note que a letra v virou maiúscula).

Existem dois construtores para a classe Carta, um é utilizado quando a carta é do tipo lacaio e outro quando a carta é do tipo magia.

O construtor da carta lacaio é dado por: public **Carta**(int id, String nome, TipoCarta tipo, int ataque, int vida, int vidaMesa, int mana).

O construtor da carta de magia é dado por: public **Carta**(int id, String nome, TipoCarta tipo, TipoMagia magiaTipo, int magiaDano, int mana).

4.3 Classe Mesa

A classe Mesa é descrita na Tabela 3. Todos os atributos são privados e seus valores podem ser recuperados por meio dos métodos get e set. A convenção para nomes de get e set é a mesma aplicada para a classe Carta.

Tabela 2: Classe Carta

Atributo	Tipo	Descrição	Domínio
ID	int	ID único de uma carta	Inteiro positivo
nome	String	Nome da carta	String
tipo	TipoCarta	Define se a carta é do tipo lacaio ou magia	Enum: TipoCarta.LACAIO, TipoCarta.MAGIA
ataque	int	Ataque da carta (lacaio)	Inteiro positivo
vida	int	Vida da carta (lacaio)	Inteiro positivo
vidaMesa	int	Vida na mesa da carta (lacaio)	Inteiro positivo
mana	int	Mana da carta	Inteiro positivo
magiaTipo	TipoMagia	Define se a magia é do tipo alvo ou área	Enum: TipoMagia.ALVO, TipoMagia.AREA
magiaDano	int	Valor de dano (magia)	Inteiro positivo
turno	int	Marca qual o turno que o lacaio desceu à mesa	Inteiro positivo

A Mesa é um objeto que deverá fornecer ao Jogador no início de seu turno uma "fotografia" (estado do jogo) imediatamente antes do turno daquele Jogador, para que ele possa analisar o jogo e tomar as decisões. O objeto mesa entra como argumento do método **processarTurno()**. Assim permite-se que o jogador consulte a Mesa para descobrir, por exemplo, se um dado lacaio ainda está vivo no jogo ou não.

Tabela 3: Classe Mesa

Atributo	Tipo	Descrição	Domínio
lacaiosJog1	ArrayList<Carta>	Lacaios na mesa do herói 1	Cartas que são lacaios vivos na mesa
lacaiosJog2	ArrayList<Carta>	Lacaios na mesa do herói 2	Cartas que são lacaios vivos na mesa
vidaHeroi1	int	Vida do herói 1	Inteiro entre [0,30]
vidaHeroi2	int	Vida do herói 2	Inteiro entre [0,30]
numCartasJog1	int	Número de Cartas jogador 1	Inteiro entre [0,10]
numCartasJog2	int	Número de Cartas jogador 2	Inteiro entre [0,10]
manaJog1	int	Mana disponível neste turno para jogador 1	Inteiro entre [1,10]
manaJog2	int	Mana disponível neste turno para jogador 2	Inteiro entre [1,10]

4.4 Classe Jogada

A classe Jogada é descrita no texto abaixo. Todos os atributos são privados e seus valores podem ser recuperados por meio dos métodos get e set. A convenção para nomes de get e set é a mesma da aplicada para a classe Carta. Os atributos da classe Jogada são os seguintes:

- *tipo*: Um atributo do tipo **TipoJogada** (enumeração) que define se a jogada trata-se de baixar um

Tabela 4: Atributos relevantes por tipo de Jogada

Tipo	cartaJogada	cartaAlvo
TipoJogada.LACAIO	X	
TipoJogada.MAGIA de alvo	X	X
TipoJogada.MAGIA de área	X	
TipoJogada.ATAQUE	X	X
TipoJogada.PODER		X

lacaio à mesa (TipoJogada.LACAIO), utilizar uma magia (TipoJogada.MAGIA), atacar com um lacaio (TipoJogada.ATAQUE) ou utilizar um poder heróico (TipoJogada.PODER).

- *cartaJogada*: Um atributo da classe **Carta** que define qual carta está sendo utilizada. No caso de baixar um lacaio (TipoJogada.LACAIO) é o lacaio que será baixado, no caso de utilizar uma magia é a carta da magia (TipoJogada.MAGIA), no caso de atacar com um lacaio é o lacaio que irá atacar (TipoJogada.ATAQUE). Toma valor **null** no caso de Poder Heróico.

- *cartaAlvo*: Um atributo da classe **Carta** que define qual carta será utilizada como alvo. Ou então toma o valor **null** para ter como alvo o herói do oponente. Se for realizar um ataque (TipoJogada.ATAQUE) ou utilizar uma magia de alvo (TipoJogada.MAGIA) o campo deve conter a carta do lacaio do oponente que será alvo desse dano, ou então **null** para ter o herói como alvo.

Note que alguns campos não são utilizados dependendo da jogada, a Tabela 4 auxilia ao verificar quais campos são utilizados para cada tipo de jogada (X) e quais não são (vazio). Por exemplo, se a jogada é de baixar um lacaio, o tipo será TipoJogada.LACAIO, e o atributo cartaJogada será o lacaio que será baixado à mesa. Porém o atributo cartaAlvo será ignorado e como convenção deve-se utilizar **null** nesta situação.

5 Funcionamento do Motor

O Motor deverá funcionar de maneira a controlar o jogo LaMa, interagindo com os jogadores e validando as suas jogadas a cada turno. Assim que houver um vencedor, ou caso seja encontrada uma Jogada inválida, o MotorRA deverá retornar um objeto da classe **GameStatus** que deverá conter todas as informações preenchidas.

É através destas informações que o Motor será posteriormente avaliado quanto à corretude (além de inspeção do código), portanto é muito importante preencher estas informações corretamente. Mais informações para preenchimento dos atributos de **GameStatus** estão na Seção 7.

Além disso o Motor deverá informar com mensagens cada jogada que é realizada por qual jogador e quais os efeitos destas jogadas. Detalhes sobre as mensagens estão na Seção 6.

6 Mensagens de Jogadas

A classe **MotorRA** deverá utilizar o método **imprimir()** para imprimir mensagens referentes a todas as Jogadas e também da transição de turnos.

Estas mensagens poderão ser posteriormente visualizadas para se obter um "log" (relatório) do jogo, igual ao Motor apresentado no Trabalho 1. Não existe um formato fixo para as mensagens, contudo elas devem ser sucintas e dar a quem lê todas as informações pertinentes para entender o que se passa no jogo.

Contribuirão para a nota do trabalho a impressão de mensagens claras e completas do jogo. É importante imprimir mensagens de maneira que qualquer pessoa que conheça o jogo LaMa consiga entender. Em caso de dúvidas, o aluno poderá utilizar como exemplo as mensagens de logs geradas no Campeonato do Trabalho 1 disponíveis no site do docente (`http://www.ic.unicamp.br/~fusberti`).

7 Classe GameStatus

A classe **GameStatus** é uma classe que tem por objetivo guardar informações a respeito do "estado de jogo" quando a partida for encerrada. A partida é encerrada quando um dos jogadores vencer o outro, ou quando ocorrer algum erro (jogada inválida gerada por um dos jogadores).

No caso de uma partida se encerrar normalmente, porque um dos jogadores venceu o outro levando a vida do oponente a zero, o atributo *status* deverá ser zero (0). E o atributo *vencedor* deverá ter 1 ou 2 para representar que o primeiro ou segundo jogador venceu o jogo, respectivamente. Os atributos *jogadaInvalida* e *mesaJogadaInvalida* deverão ser **null** neste caso.

No caso de ocorrer um erro, a vitória deverá ser dada para o jogador que não cometeu o erro (ou seja, o jogador que comete o erro "perde" imediatamente e o outro jogador é declarado vencedor). Por isso, o atributo de *vencedor* deve ser ajustado neste sentido. O atributo *status* precisa ser configurado com um número inteiro maior do que zero, correspondente ao erro que foi encontrado (conferir Tabela 6). O atributo *jogadaInvalida* deve ser preenchido com a Jogada que verificou-se inválida e o atributo *mesaJogadaInvalida* precisa ser corretamente preenchido com um objeto Mesa contendo uma "fotografia" do momento **imediatamente** antes da jogada inválida ter **qualquer** efeito no jogo. Por fim, *mensagemErro* deve ter uma mensagem contendo informações do erro. Esta String também deverá ser enviada para o método imprimir().

Portanto uma jogada precisa ser verificada se não viola nenhuma regra antes de ser executada no jogo. Assim que for verificada uma jogada inválida, o **MotorRA** deverá realizar uma cópia do estado de jogo e criar um objeto **GameStatus** conforme as instruções e retornar este objeto, encerrando a partida. Antes de retornar o objeto GameStatus, a classe **MotorRA** deve também imprimir, através do método imprimir(), a mensagem de erro com informações do erro correspondente. Tais mensagens são de formato livre mas deverão conter **pelo menos** as informações descritas na Tabela 6.

Tabela 5: Classe GameStatus

Atributo	Tipo	Jogo Normal	Jogo encerrado por erro
status	int	0 (zero)	Um inteiro representando o número da regra violada (Tabela 6)
vencedor	int	1 ou 2	Quando o segundo jogador for o responsável pelo erro, 1. Caso contrário, 2.
jogadaInvalida	Jogada	**null**	O objeto Jogada que foi considerado inválido
mesaJogadaInvalida	Mesa	**null**	Um objeto Mesa com uma "fotografia" do estado de jogo **imediatamente antes** da Jogada inválida.
mensagemErro	String	String vazia	Uma String contendo o mesmo texto idêntico à mensagem de erro enviada através do método imprimir() (veja Tabela 6).

8 Saída do programa

Atenção: a classe **MotorRA** deve imprimir mensagens apenas através do método **imprimir()**. Qualquer impressão fora deste método será considerada fora do padrão e penalizada de acordo.

9 Critério de Avaliação

O trabalho será avaliado através dos seguintes critérios:

$$NT2 = 0.6 \times NP_1 + 0.1 \times NP_2 + 0.1 \times NP_3 + 0.1 \times NP_4 + 0.1 \times NP_5 \quad (1)$$

Onde:

- NT_2: Nota do Trabalho 2
- NP_1: Nota correspondente ao critério de corretude.
- NP_2: Nota correspondente ao critério de mensagens do motor.
- NP_3: Nota correspondente ao critério de aderência ao enunciado.
- NP_4: Nota correspondente ao critério de comentários.
- NP_5: Nota correspondente ao critério de convenções.

9.1 Critérios de avaliação

1. **Corretude:** O código não deve possuir *warnings* ou erros de compilação. O código não deve emitir *exceptions* em nenhuma situação. Em cada partida que o jogo acabar ou que os jogadores realizarem jogadas inválidas, o MotorRA deverá devolver um objeto **GameStatus** com os atributos exatamente de acordo com a especificação deste enunciado.
2. **Mensagens do Motor:** As mensagens que o Motor emitir com o método imprimir() devem ser sucintas e ao mesmo tempo possuir as informações relevantes para informar cada jogada.

Tabela 6: Tabela de códigos de erros

Código	Descrição do erro	Mensagem de erro deve conter **pelo menos** e de maneira clara
1	Baixar lacaio ou usar magia sem possuir a carta na mão	ID da carta que seria usada/baixada, IDs das cartas na mão.
2	Realizar uma jogada que o limite de mana não permite	O Tipo da Jogada, quanto de mana a jogada custaria, quanto de mana há disponível.
3	Tentar baixar uma carta de magia como carta lacaio	ID e nome da carta que seria utilizada incorretamente
4	Baixar um lacaio já tendo sete outros lacaios em mesa	ID e nome do lacaio que iria ser baixado
5	Atacar com um lacaio invalido de origem do ataque	ID (inválido) da carta lacaio que iria atacar
6	Atacar com um lacaio que foi baixado neste turno	ID da carta lacaio que iria atacar
7	Atacar com um lacaio mais de uma vez por turno	ID da carta lacaio que iria atacar
8	Atacar com um lacaio um alvo inválido	ID do lacaio atacante, ID (inválido) do alvo que seria atacado
9	Tentar usar uma carta de lacaio como uma magia	ID e nome da carta que seria utilizada incorretamente
10	Usar uma magia de alvo em um alvo inválido	ID da magia de alvo, ID (inválido) do alvo
11	Usar o poder heróico mais de uma vez por turno	ID do alvo
12	Usar o poder heróico em um alvo inválido	ID do alvo inválido

Obs: Deve-se escrever na mensagem “Herói X” quando o herói for o alvo, X sendo 1 ou 2.

3. **Aderência ao Enunciado:** A implementação deve realizar o que é requisitado no enunciado, atendendo a todos os avisos e observações.
4. **Comentários:** Os comentários devem ser suficientes para explicar os trechos mais importantes da implementação.
5. **Convenções:** A implementação deve seguir as convenções do Java. O código deve ser identado, modularizado e hierarquizado para melhor organização.

10 Observações

- O trabalho é individual.
- Não é permitido nenhum tipo de compartilhamento de código entre os alunos ou o uso de códigos de terceiros. Uma única exceção permitida consiste nas APIs da linguagem Java. Em caso de plágio, todos os alunos envolvidos serão reprovados com média zero.
- O aluno deve realizar os testes de sua classe no ambiente de programação JavaSE 1.7. Uma vez que os códigos serão testados nesse ambiente.

11 Submissão

O trabalho deverá ser submetido até o dia 20 de novembro pelo Ensino Aberto. A submissão deve ser exclusivamente um arquivo MotorRAxxxxxx.java (onde “’xxxxxx’ é o RA do aluno). Crie uma pasta denominada “Trabalho 2” em seu portfólio do Ensino Aberto e inclua seu código-fonte nessa pasta. **Importante**: Não se esqueça de deixar a visibilidade da pasta do Portfólio como "Compartilhado somente com Formadores".