MC302
Primeiro semestre de 2017

Trabalho 2

Professores: Esther Colombini (esther@ic.unicamp.br) e Fábio Luiz Usberti (fusberti@ic.unicamp.br)
PEDs: (Turmas ABCD) Elisangela Santos (ra149781@students.ic.unicamp.br), Lucas Faloni (lucasfaloni@gmail.com), Lucas David (lucasolivdavid@gmail.com), Wellington Moura (wellington.tylon@hotmail.com)
PEDs (Turmas EF) Natanael Ramos (naelr8@gmail.com), Rafael Arakaki (rafaelkendyarakaki@gmail.com)
PAD: (Turmas ABCD) Igor Torrente (igortorrente@hotmail.com)
PAD: (Turmas EF) Bleno Claus (blenoclaus@gmail.com)

---

1 Objetivo

Estudo e aplicação dos conceitos de programação orientada a objetos, abordados em aula, para uma aplicação na área de jogos utilizando-se da linguagem de programação Java.

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. Existem três tipos de magias: (i) dano em alvo, (ii) dano em área, (iii) magia de buff. As magias de dano em alvo causam dano em um único lacaio do oponente à escolha do jogador utilizador ou no herói do oponente. As magias de área causam dano em todos os lacaios e também no herói do oponente. Por fim, as magias de buff podem aumentar o ataque e a vida de um dos lacaios do jogador que a utilizar. Não é permitido usar magia de buff em lacaio 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 sete cartas na mão. Se já possuir sete cartas ao início de seu turno, a nova carta que seria comprada é descartada.

13. Quando o jogador fica sem cartas para comprar do baralho chamamos esse estado de jogo de *fadiga*. A *fadiga* inicia quando em um dado turno não for possível a um jogador comprar cartas porque o baralho já está esgotado, então o jogador recebe um de dano no início daquele turno. No próximo início de turno receberá dois de dano, depois três, e assim por diante.

14. **Para o Trabalho 2, os lacaios podem possuir efeitos especiais de investida, ataque duplo ou provocar.** (Seção 5).

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 inteiro contendo qual é o jogador vencedor se não houver nenhum erro durante a execução do Motor. O valor de retorno é 1 se o primeiro jogador é vencedor, e 2 se o segundo jogador é o vencedor. Se houver algum erro, o **MotorRA** deverá disparar uma exceção correspondente (Seção 6.2).

- 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.

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<CartaLacaio>	Cartas lacaio na mesa do primeiro jogador.
lacaiosMesa2	ArrayList<CartaLacaio>	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 ligada (1) ou desligada (0).
saidaArquivo	FileWriter	Escritor para a saída em um arquivo, ativado (!=null) ou desativado (=null).
funcionalidadesAtivas	EnumSet<Funcionalidades>	Conjunto contendo as funcionalidades ativas para o Motor

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()*.

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 vidaAtual,

então o método get correspondente se chamará getVidaAtual (note que a letra v virou maiúscula).

Tabela 2: Classe Carta

Atributo	Tipo	Descrição	Domínio
ID	int	ID único de uma carta	Inteiro positivo
nome	String	Nome da carta	String
custoMana	int	Custo de mana da carta	Inteiro positivo

Note que a classe Carta (Tabela 2) é uma classe abstrata. Existem duas classes que herdam da classe Carta: CartaLacaio e CartaMagia que serão descritas a seguir.

Atenção: o método equals() para Carta compara se os IDs das cartas são iguais. A classe Carta implementa a interface Comparable e o somente o atributo ID é utilizado para comparar as cartas. Pode-se supor que qualquer baralho LaMa válido contém IDs únicos para cada jogador e para cada carta (mesmo tratando-se de cartas de mesmo nome).

4.3 Classe CartaLacaio

A classe CartaLacaio é descrita na Tabela 3. Uma vez que a classe CartaLacaio herda da classe Carta, a classe CartaLacaio possui também os atributos e métodos da classe abstrata Carta. Na Tabela 3 são mostrados apenas os atributos específicos da classe CartaLacaio. Todos os atributos são privados e seus valores podem ser recuperados por meio dos métodos get e set.

Tabela 3: Classe CartaLacaio

Atributo	Tipo	Descrição	Domínio
ataque	int	Ataque da carta	Inteiro positivo
vidaAtual	int	Vida da carta durante o jogo	Inteiro positivo
vidaMaxima	int	Vida da carta sem contar danos	Inteiro positivo
efeito	TipoEfeito	Efeito especial do lacaio	Enumeração: NADA, INVESTIDA, PROVOCAR ou ATAQUE_DUPLO
turno	int	(Utilizado pelo Motor)	Inteiro positivo

Atenção: o atributo efeito deverá ser utilizado e considerado pelo **MotorRA** se e somente se o efeito correspondente estiver contido no conjunto do atributo funcionalidadesAtivas inicializado no **MotorRA** (Seção 5).

4.4 Classe CartaMagia

A classe CartaMagia é descrita na Tabela 4. Uma vez que a classe CartaMagia herda da classe Carta, a classe CartaMagia possui também os atributos e métodos da classe abstrata Carta. Na Tabela 4 são mostrados apenas os atributos específicos da classe CartaMagia. Todos os atributos são privados e seus valores podem ser recuperados por meio dos métodos get e set.

Tabela 4: Classe CartaMagia

Atributo	Tipo	Descrição	Domínio
magiaTipo	TipoMagia	Define o tipo da magia	Enumeração: ALVO, AREA ou BUFF
magiaDano	int	Valor de dano ou buff	Inteiro positivo

4.5 Classe Mesa

A classe Mesa é descrita na Tabela 5. 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.

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 5: Classe Mesa

Atributo	Tipo	Descrição	Domínio
lacaiosJog1	ArrayList<CartaLacaio>	Lacaios na mesa do herói 1	Cartas que são lacaios vivos na mesa
lacaiosJog2	ArrayList<CartaLacaio>	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]

4.6 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 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.

Tabela 6: Atributos relevantes por tipo de Jogada

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

- *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 6 mostra 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.

O construtor da classe Jogada é: public **Jogada**(TipoJogada tipo, Carta cartaJogada, Carta cartaAlvo).

5 Efeitos

Um novo recurso do jogo LaMa deverá ser implementado: os efeitos especiais de lacaio. Esses efeitos são caracterizados no objeto CartaLacaio pelo atribudo **efeito**. Tais efeitos podem ser três:

- INVESTIDA: um lacaio com esse efeito pode atacar no mesmo turno em que é baixado à mesa.
- ATAQUE_DUPLO: um lacaio com esse efeito pode atacar duas vezes por turno. O lacaio pode atacar dois alvos diferentes. Entretanto, se o primeiro ataque causar a morte do lacaio ele não poderá atacar uma segunda vez.
- PROVOCAR: se houver algum lacaio com esse efeito em campo de um jogador, os lacaios do outro jogador e também o poder heróico deste só poderão ter como alvo o lacaio com efeito provocar. Em outras palavras, não é possível atacar outro alvo diferente do lacaio com **PROVOCAR**. Somente após o(s) lacaio(s) com **PROVOCAR** morrer(em) é que é permitido atacar outros alvos. Magias podem ser utilizadas em qualquer alvo independentemente do efeito **PROVOCAR**.

Os efeitos só terão validade no jogo se o parâmetro **funcionalidadesAtivas** passado ao **MotorRA** no construtor ativar cada um dos efeitos. Exemplo: se **funcionalidadesAtivas** contiver somente as funcionalidades (Seção 5.2) Funcionalidade.INVESTIDA e Funcionalidade.ATAQUE_DUPLO então o MotorRA deverá considerar que os efeitos de INVESTIDA e ATAQUE_DUPLO estão ativos para a partida em questão mas ignorar o efeito de PROVOCAR.

Dica: é recomendável que a implementação dos efeitos especiais de lacaio seja deixado como última tarefa. A não implementação dessa funcionalidade incorre em uma penalização máxima de até dois pontos na nota final (Seção 10.1).

Ligados aos efeitos especiais de lacaio existem duas enumerações descritas nas próximas seções.

5.1 Enumeração TipoEfeito

Trata-se de uma enumeração usada como atributo na classe CartaLacaio. A enumeração pode assumir os seguintes valores: NADA, INVESTIDA, ATAQUE_DUPLO, PROVOCAR.

5.2 Enumeração Funcionalidade

Trata-se de uma enumeração usada como parâmetro no Motor na declaração do atributo **funcionalidadesAtivas**. A enumeração pode assumir os seguintes valores: INVESTIDA, ATAQUE_DUPLO, PROVOCAR.

6 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 o MotorRA deverá retornar um inteiro informando quem venceu o jogo. Ou, caso seja encontrada uma Jogada inválida, o MotorRA deverá disparar uma exceção do tipo **LamaException** (Seção 6.2) contendo as informações pertinentes.

É através destas informações que o Motor será posteriormente avaliado quanto à corretude (além de inspeção do código), portanto é muito importante preencher essas informações corretamente. Mais informações para preenchimento dos atributos de **LamaException** estão na Seção 6.2.

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 6.1.

6.1 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`).

6.2 Erros de Jogadas Inválidas

A classe **MotorRA** deverá implementar um mecanismo para detectar possíveis jogadas inválidas realizadas pelos jogadores e disparar exceções de acordo com o tipo de jogada inválida.

No caso de uma partida se encerrar normalmente, porque um dos jogadores venceu o outro levando a vida do oponente a zero, nenhuma exceção deverá ser disparada e o método executarPartida deverá retornar 1 se o primeiro jogador venceu o jogo e 2 se o segundo jogador venceu o jogo.

No caso de ocorrer um erro, uma exceção do tipo **LamaException** deverá ser disparada. Os atributos da classe **LamaException** são mostrados na tabela 7. 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 nesse sentido. O atributo *numeroErro* precisa ser configurado com um número inteiro maior do que zero, correspondente ao erro que foi encontrado (conferir Tabela 8). O atributo *jogadaInvalida* deve ser preenchido com a Jogada que

verificou-se inválida. Por fim, o atributo *message* deve ter uma mensagem contendo informações do erro. A mensagem é de formato livre mas deverá conter **pelo menos** as informações descritas na Tabela 8.

Portanto cada jogada precisa ser verificada, em ordem, se não viola nenhuma regra antes de ser executada no jogo. Assim que for verificada uma jogada inválida, o **MotorRA** deverá realizar disparar a exceção contendo todas as informações pertinentes.

Tabela 7: Classe LamaException

Atributo	Tipo	Conteúdo
numeroErro	int	Um inteiro representando o número da regra violada (Tabela 8)
vencedor	int	Quando o primeiro jogador for o responsável pelo erro, 2. Caso contrário, 1.
jogada	Jogada	O objeto Jogada que foi considerado inválido
message	String	Uma String contendo o mesmo texto idêntico à mensagem de erro enviada através do método imprimir() (veja Tabela 8).

7 Independência de Baralho

O MotorRA deverá funcionar para qualquer baralho contendo cartas LaMa válidas e não somente para o baralho padrão disponibilizado.

8 Diagrama

É disponibilizado para o Trabalho 2 um diagrama de classes contendo as principais classes utilizadas nesse projeto. É recomendável a consulta ao diagrama para uma visão mais sistêmica do projeto.

9 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.

10 Critério de Avaliação

O trabalho será avaliado através dos seguintes critérios:

$$NT2 = 0.6 \times NP_1 + 0.2 \times NP_2 + 0.1 \times NP_3 + 0.05 \times NP_4 + 0.05 \times NP_5 \quad (1)$$

Onde:

- NT_2: Nota do Trabalho 2
- NP_1: Nota correspondente ao critério de corretude.

Tabela 8: 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 ou buff 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
13	Existindo um lacaio com provocar, atacar outro alvo	ID do alvo inválido, ID do lacaio vivo com provocar

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

- NP_2: Nota correspondente ao critério de aderência ao enunciado.
- NP_3: Nota correspondente ao critério de mensagens do motor.
- NP_4: Nota correspondente ao critério de comentários.
- NP_5: Nota correspondente ao critério de convenções.

10.1 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 normalmente deverá ser devolvido um inteiro representando o vencedor. Ou se os jogadores realizarem jogadas inválidas, o MotorRA deverá devolver disparar um **LamaException** com os atributos exatamente de acordo com a especificação.
2. **Aderência ao Enunciado:** A implementação deve realizar o que é requisitado no enunciado, atendendo a todos os avisos e observações.

3. **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.

4. **Comentários:** Os comentários devem ser suficientes para explicar os trechos mais importantes da implementação e bem identados, com preferência para o formato JavaDoc [1].

5. **Convenções:** A implementação deve seguir as convenções de nomes de atributos e métodos do Java. O encapsulamento dos atributos deve ser observado.

11 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 na API padrão da linguagem Java [2]. 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 1.8 (Java 8). Uma vez que os códigos serão testados nesse ambiente.

12 Submissão

O trabalho deverá ser submetido até às 23:55 do dia 27 de junho pelo moodle. A submissão deve ser exclusivamente um arquivo **MotorRAxxxxxx.java** (onde "'xxxxxx' é o RA do aluno). **Importante**: em caso de mais de uma classe, todas as classes deverão estar no mesmo arquivo e somente a classe **MotorRAxxxxxx** deve ser pública.

[1] `https://pt.wikipedia.org/wiki/Javadoc`
[2] `http://docs.oracle.com/javase/8/docs/api/`