UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CÂMPUS PATO BRANCO

Disciplina: **Programação Orientada a Objetos (PO24CP-4CP)**
Profª. Luciene de Oliveira Marin
lucienemarin@utfpr.edu.br

Lista de exercícios - Associação entre classes

Exercício #01 - Diagramas de classe UML

a) Faça um diagrama de classes UML para representar um sistema para registrar informações sobre filmes, atores que participaram, diretores e prêmios recebidos.

b) Faça um diagrama de classes UML para representar um sistema para venda de passagens aéreas.

c) Faça um diagrama de classes UML para representar um sistema de gestão de bibliotecas, onde é necessário registrar os livros, usuários, empréstimos, etc.

Exercício #02 - Classe DataHora

Considere a classe **DataHora** (Listagem 1) e as classes **Data** e **Hora** cujas instâncias são usadas na sua composição. Escreva, se ainda não existir na classe **Data**, um método **éIgual** que receba como argumento uma instância da própria classe **Data** e retorne o valor booleano `true` se a data representada for igual à data passada. Faça o mesmo para a classe **Hora**. Escreva também na classe **DataHora** um método **éIgual** que receba outra instância da própria classe **DataHora** como argumento e que seja executado delegando a comparação aos métodos das classes Data e Hora. Veja também a Listagem 2.

Listing 1: Classe DataHora.java

```
/**
 * A classe DataHora reutiliza as classes Data e Hora através de delegação.
 * A data e hora são representadas por instâncias das respectivas classes que estão
 * embutidas na classe DataHora, e toda a interação entre esta classe e as embutidas
 * é feita através da chamada de métodos das classes embutidas. Esta classe demonstra
 * o conceito de reutilização de classes através de delegação ou composição.
 */
class DataHora // declaração da classe
  {
  /**
   * Declaração dos campos da classe. Estes campos são declarados como privados
   * para que não possam ser acessados de fora da classe.
   */
  private Data estaData; // uma referência à instância da classe Data representa
                         // o dia, mês e ano
  private Hora estaHora; // uma referência à instância da classe Hora representa
                         // a hora, minuto e segundo

 /**
  * O construtor para a classe DataHora, que recebe argumentos para inicializar
  * todos os campos que esta classe indiretamente contém, e chama os construtores
```

```
  * das classes Data e Hora para inicializar os campos das instâncias destas classes.
  * @param hora a hora
  * @param minuto o minuto
  * @param segundo o segundo
  * @param dia o dia
  * @param mês o mês
  * @param ano o ano
  */
  DataHora(byte hora,byte minuto,byte segundo,byte dia,byte mês,short ano)
    {
    estaData = new Data(dia,mês,ano);
    estaHora = new Hora(hora,minuto,segundo);
    }

 /**
  * O construtor para a classe DataHora, que recebe argumentos para inicializar
  * os campos que representam uma data. O construtor também inicializará os campos
  * que representam uma hora, considerando que todos valem zero (meia-noite).
  * De novo, os construtores das classes embutidas nesta serão chamados.
  * @param dia o dia
  * @param mês o mês
  * @param ano o ano
  */
  DataHora(byte dia,byte mês,short ano)
    {
    estaData = new Data(dia,mês,ano);
    estaHora = new Hora((byte)0,(byte)0,(byte)0); // cast necessário
    }

  /**
   * O método toString não recebe argumentos, e retorna uma string contendo os valores
   * dos campos da classe formatados. Os valores são obtidos através da chamada
   * implícita aos métodos toString das instâncias das classes embutidas.
   * @return uma string com os valores dos campos formatados.
   */
  public String toString()
    {
    return estaData+" "+estaHora;
    }

  } // fim da classe DataHora
```

Listing 2: Classe DataSimples.java

```
/**
 * A classe DataSimples contém campos e métodos que permitem a manipulação de
 * datas.
 */
class DataSimples // declaração da classe
  {
  /**
   * Declaração dos campos da classe
   */
  byte dia,mês; // dia e mês são representados por bytes
  short ano; // ano é representado por um short

  /**
   * O método inicializaDataSimples recebe argumentos para inicializar os campos da
   * classe DataSimples. Este método chama o método dataÉVálida para verificar se os
   * argumentos são correspondentes a uma data válida: se forem, inicializa os
   * campos, caso contrário inicializa todos os três campos com o valor zero.
   * @param d o argumento correspondente ao método dia
```

```
 * @param m o argumento correspondente ao método mês
 * @param a o argumento correspondente ao método ano
 */
void inicializaDataSimples(byte d,byte m,short a)
  {
  if (dataÉVálida(d,m,a)) // se a data for válida, inicializa os campos com os
                          // valores passados como argumentos
    {
    dia = d; mês = m; ano = a;
    }
  else // caso contrário, inicializa os campos com zero
    {
    dia = 0; mês = 0; ano = 0;
    }
  } // fim do método inicializaDataSimples

/**
 * O método dataÉVálida recebe três valores como argumentos e verifica de maneira
 * simples se os dados correspondem a uma data válida. Se a data for válida, retorna
 * a constante booleana true, caso contrário, retorna a constante booleana false.
 * Vale a pena notar que este algoritmo é simples e incorreto, um dos exercícios
 * sugere a implementação do algoritmo correto.
 * @param d o argumento correspondente ao método dia
 * @param m o argumento correspondente ao método mês
 * @param a o argumento correspondente ao método ano
 * @return true se a data for válida, false se não for válida
 */
boolean dataÉVálida(byte d,byte m,short a)
  {
  if ((d >=1) &&       // se o dia for maior ou igual a 1  E
      (d <= 31) &&     // se o dia for menor ou igual a 31  E
      (m >= 1) &&      // se o mês for maior ou igual a 1  E
      (m <= 12))       // se o mês for menor ou igual a 12 ENTÃO
    return true;       // a data é válida, retorna true
  else
    return false;      // a data não é válida, retorna false
  } // fim do método dataÉVálida

/**
 * O método éIgual recebe uma instância da própria classe DataSimples como argumento
 * e verifica se a data representada pela classe e pela instância que foi passada é
 * a mesma. A comparação é feita comparando os campos da data um a um.
 * @param outraDataSimples uma instância da própria classe DataSimples
 * @return true se a data encapsulada for igual à passada, false caso contrário
 */
boolean éIgual(DataSimples outraDataSimples)
  {
  if ((dia == outraDataSimples.dia) &&    // se os dois dias forem iguais E
      (mês == outraDataSimples.mês) &&    // se os dois meses forem iguais E
      (ano == outraDataSimples.ano))      // se os dois anos forem iguais então
    return true;       // a data é igual, retorna true
  else
    return false;      // a data é diferente, retorna false
  } // fim do método éIgual

/**
 * O método mostraDataSimples não recebe argumentos nem retorna valores. Este método
 * somente imprime os valores dos campos, formatados de forma que uma barra ("/")
 * seja impressa entre eles. Quando o valor do ano for impresso, uma quebra de
 * linha também será impressa.
 */
void mostraDataSimples()
```

```
    {
    System.out.print(dia);    // O método print do campo out da classe System faz com
    System.out.print("/");    // que o argumento passado a ele seja transformado em uma
    System.out.print(mês);    // string e impresso no terminal. O método println faz a
    System.out.print("/");    // mesma coisa, mas adiciona uma quebra de linha ('\n')
    System.out.println(ano); // ao final da string impressa.
    }  // fim do método mostraDataSimples

} // fim da classe DataSimples
```

Exercício #03 - Pilhas e filas

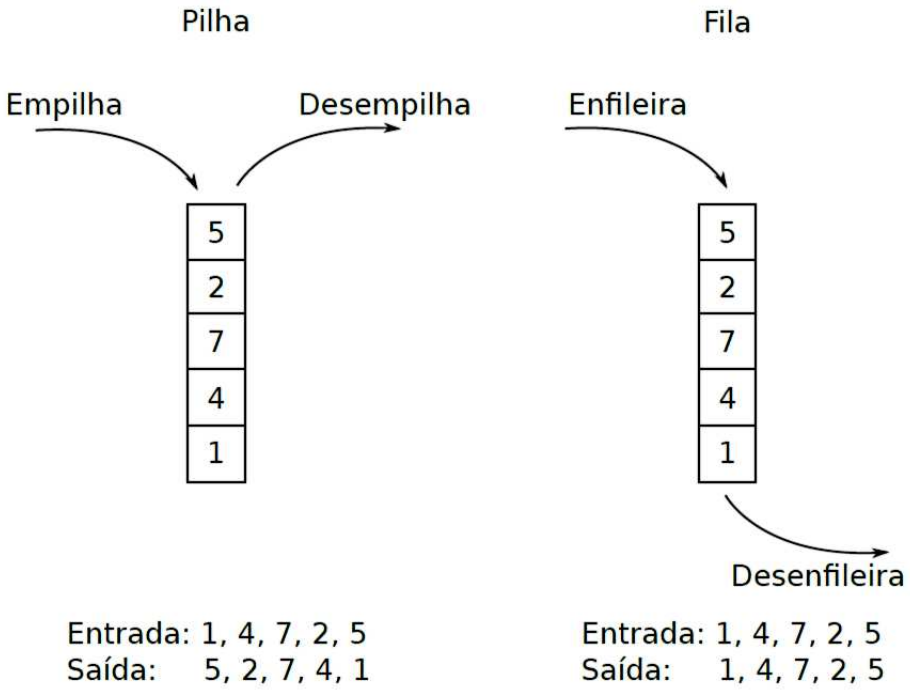

Figura 1: Estrutura de dados: Pilha e Fila

A Figura 1 apresenta duas estruturas de dados: pilha e fila. A estrutura **Pilha** implementa o algoritmo LIFO (*Last In, First Out*), ou seja, o último elemento a entrar na pilha será o primeiro a sair. A estrutura **Fila** implementa o algoritmo FIFO (*First In, First Out*), ou seja, o primeiro elemento a entrar é também o primeiro a sair.

Com base no texto descritivo acima, desenvolva o que se pede:

a) Pedro é um aficionado pela linguagem Java e assina diversas revistas sobre o assunto. Apesar de gostar do assunto, Pedro demora um pouco para ler as revistas e assim as novas revistas que chegam vão sendo empilhadas, uma sobre a outra. Assim que Pedro termina de ler uma revista, ele inicia a leitura da próxima, que é aquela do topo da pilha.

- Modele em UML as classes para representar as revistas e a pilha de revistas, bem como a associação entre essas. Sabe-se que uma revista possui um nome, um número da edição, o mês e o ano da publicação. A pilha de revistas pode armazenar até 50 revistas e a última revista a ser acrescentada na pilha será a primeira a ser retirada. A classe para representar a pilha deverá possuir métodos para as operações de adição, remoção na pilha e para indicar se a pilha está cheia. Após a modelagem, faça a devida implementação em Java.

b) Um salão de cabeleiros pretende implantar um sistema computacional para organizar a fila de atendimento de clientes. Um cliente ao chegar no salão deverá informar ao atendente seu nome e sexo. O cliente ficará então aguardando sua vez na sala de espera.

Assim que um cabeleiro terminar o atendimento a um cliente, ele acionará o sistema para solicitar um novo cliente e dessa forma saberá o nome e o sexo do próximo cliente a ser atendido. A sala de espera comporta no máximo 10 clientes.

Com base no texto acima, identifique as classes (com atributos e métodos) e suas associações e as represente em UML. Após a modelagem, faça a devida implementação em Java.

Exercício #04 - Jogo dos aviões

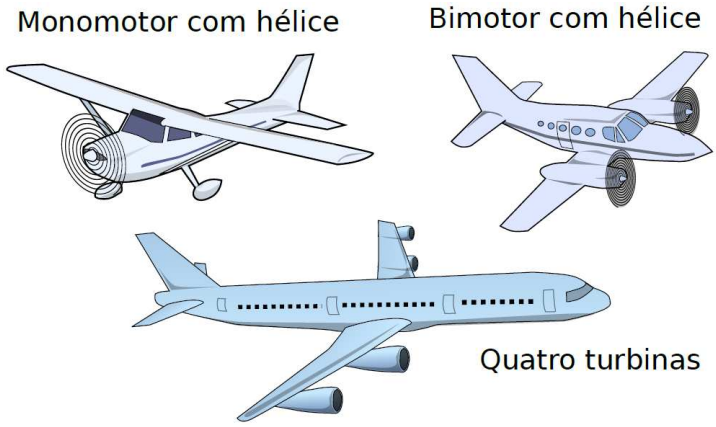

Figura 2: Diferentes tipos de aviões

A Figura 2 ilustra três tipos de aviões presentes em um jogo de computador. Abaixo segue uma descrição sobre os aviões desse jogo.

- Os aviões podem possuir de 1 até 8 motores. No jogo o avião é visto como um todo, ou seja, depois de criado um avião não é possível desmontá-lo para reaproveitar peças em outros aviões;
- Os motores podem ser constituídos por hélices ou por turbinas;
- Cada avião possui um peso, um número máximo de tripulantes e um número máximo de passageiros;
- Todos os aviões possuem um manche que permite aumentar ou diminuir a velocidade, que por consequência aumenta ou diminui a potência de cada um dos motores;
- Aviões com mais de um motor permitem ao piloto aumentar e diminuir a potência para cada motor de forma individual. Por exemplo, no caso de um bimotor o piloto poderia deixar o motor da direita com 100% de potência e o outro motor com 90% de potência;
- Os aviões apresentam um botão que permitem aos pilotos ligá-los e desligá-los. Sempre que um avião é ligado todos os seus motores ficam com 10% de potência. Toda vez que um avião é desligado todos os seus motores ficam com 0% de potência;
- O piloto pode direcionar o avião para cima, para baixo, para esquerda e para direita. Para isto o piloto deve informar a direção desejada e a intensidade de força aplicada ao comando para que o avião realize a tarefa.

Pede-se:

a) Com base na Figura 2 e em seu texto descritivo, identifique as classes existentes e faça um diagrama de classes UML para representá-las. Deve-se indicar todos os **atributos**, **métodos** de cada classe e utilizar os **modificadores de acesso** de forma a respeitar o princípio do encapsulamento dos dados. No diagrama devem estar representadas as **associações** (agregação ou composição) entre as classes, caso existam.

b) Implemente em Java as classes apresentadas no diagrama UML do item a)

c) Implemente um programa Java e neste crie 3 aviões. Realize algumas interações com cada um desses aviões, como ligar, aumentar velocidade, subir, etc.

Exercício #05 - Foguete

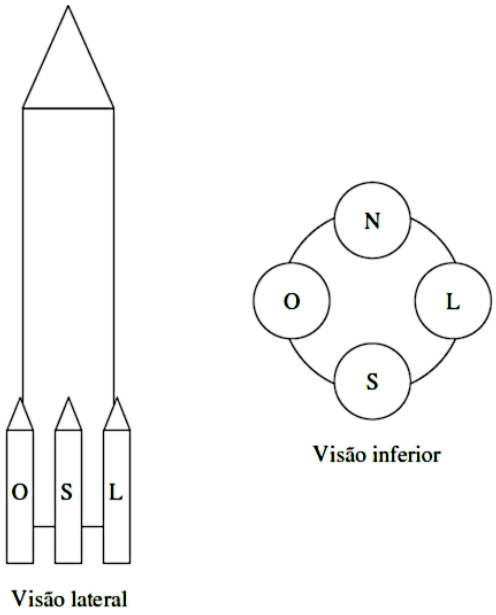

Figura 3: Foguete

A Figura 3 ilustra um **foguete** que possui quatro **propulsores** identificados como: norte (N), sul (S), leste (L) e oeste (O).

- O **centro de controle** para lançamento de foguetes interage diretamente com o foguete para:
  - Verificar o estado do foguete. Por exemplo, qual o nível de combustível, se o foguete está no estado de repouso ou no estado de movimento;
  - Enviar comandos ao foguete. Por exemplo, para dar ignição, para mudar sua direção (norte, sul, leste, oeste), aumentar potência dos **propulsores**, diminuir a potência dos **propulsores** e em caso de emergência para se auto-destruir.
- Cada **propulsor** por sua vez é acionado pelo **foguete** para que este obtenha a potência atual do propulsor (valor que vai de 0 a 100), para aumentar e diminuir a potência.
- Quando o **centro de controle** envia ao **foguete** a mensagem para aumentar ou diminuir a potência, este por sua vez aumenta ou diminui a potência de **todos** os seus **propulsores**.

- A mudança de direção do foguete, acionada pelo centro de controle, implica em modificar a potência de propulsores de forma que o propulsor da direção que se pretende guiar o foguete tenha uma potência inferior ao propulsor oposto a esta direção.
  - Por exemplo, o centro de controle informa para o foguete ser direcionado para leste. Assim, a potência do **propulsor O** deve ser maior que a potência do **propulsor L**. Se o **propulsor L** estiver na potência máxima, então será necessário diminuir a potência de L para um valor abaixo da potência do **propulsor O**. Agora se a potência de L for inferior à potência máxima, então basta aumentar a potência do **propulsor O**.

Pede-se:

De acordo com a Figura 3 e seu texto descritivo,

a) Identifique as classes, com seus principais atributos e métodos, as associações (agregação ou composição) e as represente em um diagrama de classes UML.

b) Implemente em Java as classes que foram identificadas no item anterior.

c) Implemente a classe **CentroDeControle** como sendo um programa Java, crie 2 (dois) Foguetes e envie alguns comandos para que este foguetes, de acordo com o que foi proposto nos itens a) e b) para a classe **Foguete**.

Exercício #06 - Máquinas reais × máquinas virtuais

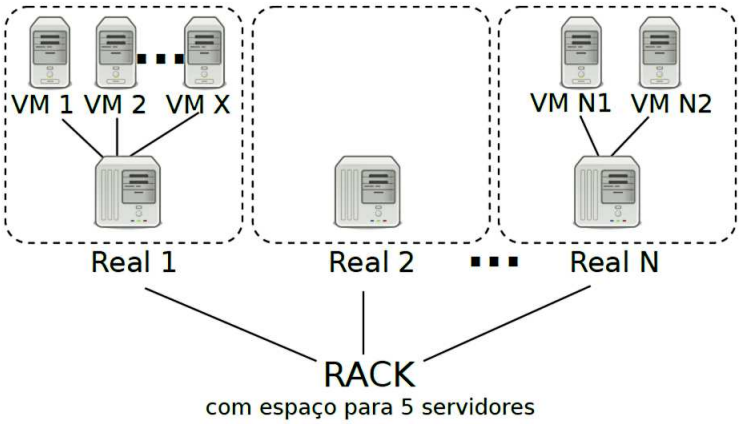

Figura 4: Rack com máquinas reais, as quais hospedam máquinas virtuais

A Figura 4 ilustra a disposição de máquinas reais (computadores) em um armário (RACK). Cada máquina real pode hospedar algumas máquinas virtuais. A seguir algumas características sobre cada tipo de máquina.

Máquina	**Possui**	**Permite**
Real	Um nome, memória RAM, um disco rígido e um conjunto de máquinas virtuais	criar, remover, ligar e desligar máquinas virtuais
Virtual	Um nome, memória RAM e um disco rígido	ligar e desligar

- Uma **máquina real** pode armazenar até 5 máquinas virtuais, porém toda **máquina virtual** criada consome uma quantidade de memória RAM e de disco rígido. Uma **máquina real** não pode possuir máquinas virtuais cujo somatório de memória RAM ou de disco rígido seja superior ao total de memória RAM e de disco rígido da própria **máquina real**. Ou seja, a criação de novas máquinas virtuais está condicionada a existência de recursos na máquina real;
- Os recursos alocados (memória e disco) para uma **máquina virtual** são imediatamente devolvidos para a **máquina real** assim que a **máquina virtual** é removida;
- Uma **máquina virtual** ao ser ligada ou desligada, exibe uma mensagem no terminal indicando seu estado, por exemplo, "Nome da máquina está ligada!";
- Para criar uma **máquina virtual** é necessário informar o nome da máquina, total de memória RAM e o tamanho do disco rígido;
- Um **RACK** pode armazenar até 5 **máquinas reais**. Para criar uma **máquina real** é necessário informar o nome da máquina, total de memória RAM e o tamanho do disco rígido.

Pede-se:

De acordo com a Figura 4 e seu texto descritivo, identifique as classes, com seus principais atributos e métodos, as associações (agregação ou composição) e as represente em um diagrama de classes UML.

a) Identifique as classes, com seus principais atributos e métodos, as associações (agregação ou composição) e as represente em um diagrama de classes UML.

b) Implemente em Java as classes que foram identificadas no item anterior.

c) Implemente um programa Java onde deve ser criado um "RACK" para hospedar até 5 **máquinas reais**. Crie algumas **máquinas reias** e **máquinas virtuais** e ligue e deslige algumas máquinas.