UDESC - Fundação Universidade do Estado de Santa Catarina
Professor: Fabiano Baldo
Disciplina: Programação Orientada a Objetos

A lista deve ser entregue até as 23h59 do dia definido na atividade do Moodle, os arquivos devem ser compactados em um arquivo *.zip* ou *.tar*. O arquivo compactado deverá conter o projeto apenas os arquivos **.java**. **Não serão aceitos projetos com os códigos-fonte no formato *.class*!**

Lista 4: Interface e Exceções

Exercício 1:

A partir do UML apresentado na Figura 1, implemente as classes Soma, MDC, Mod e Multiplicação, utilizando apenas métodos iterativos. Isto é, para as operações de multiplicação, MDC e Mod, métodos prontos da linguagem não serão consideradas.

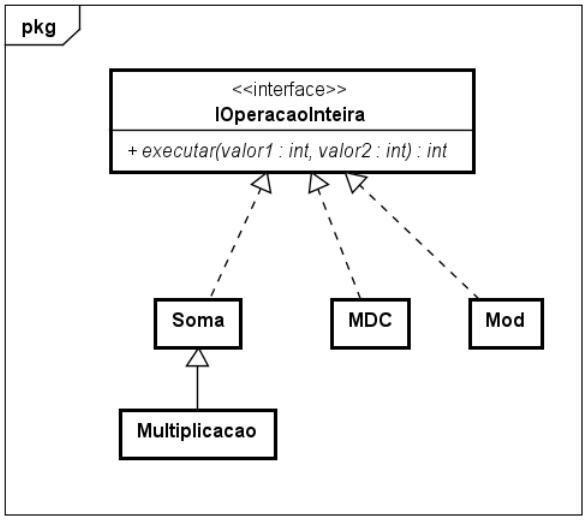

Figure 1: Diagrama UML representando a hierarquia de classes de operações matemáticas.

A classe Multiplicação, deve implementar o método **executar()** utilizando da implementação de sua superclasse. Crie também uma classe Main contendo um método **main()** e uma lista contendo um objeto de cada uma das classes da Figura 1. Gere valores aleatórios e invoque os métodos **executar()** dos objetos da lista.

Exercício 2:

A Figura 2 apresenta um diagrama de classes para um sistema de distribuição de processos. Implemente as classes descritas no diagrama de acordo com a descrição a seguir.

As classes PilhaVaziaException, PilhaCheiaException e ProcessoSemJuizException estendem a classe Exception, contendo uma mensagem semelhante ao nome da própria classe. A classe

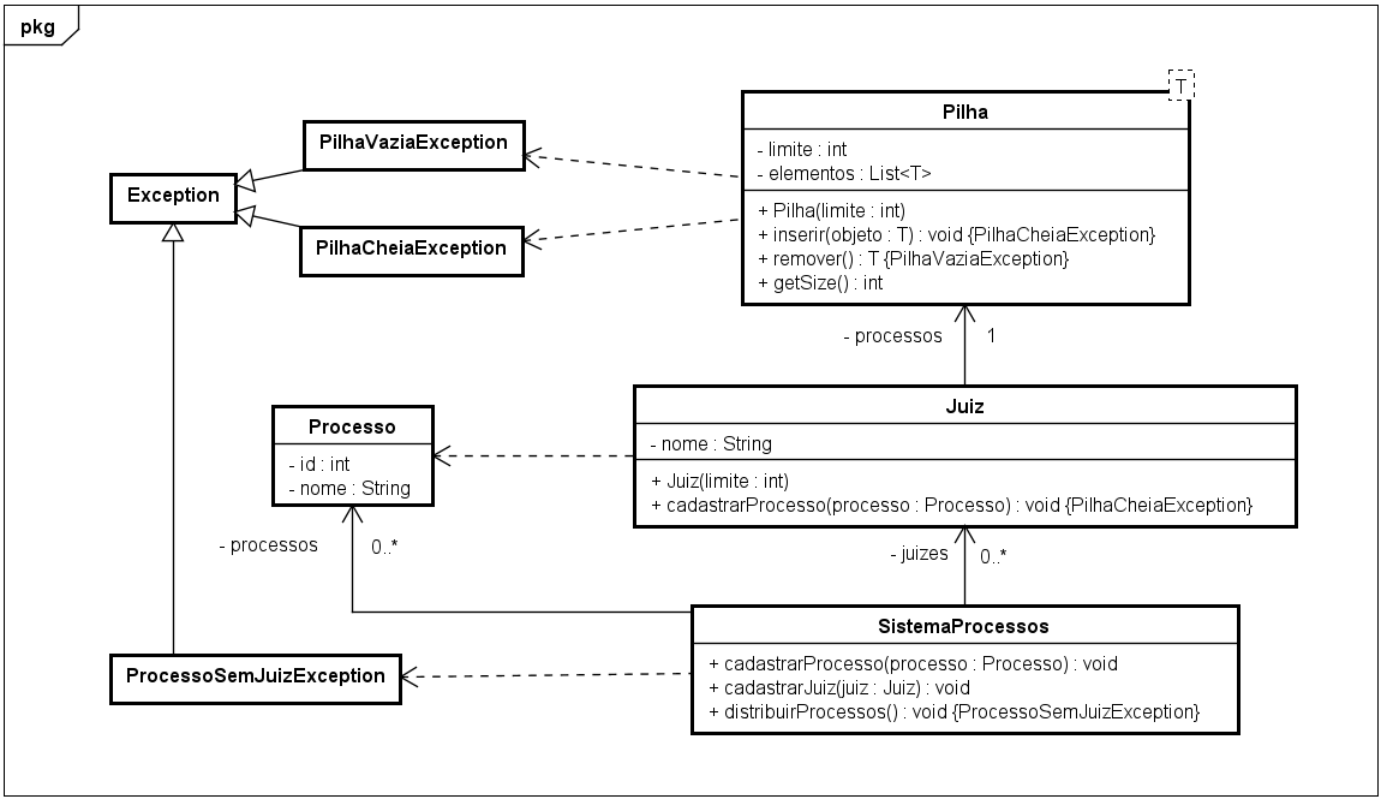

Figure 2: Diagrama UML representando um sistema de distribuição de processos.

Pilha, tipada para um tipo T genérico, possui um atributo inteiro limite, tal valor deve ser passado no construtor da classe e determina o tamanho máximo da pilha. O métodos **inserir()** insere elementos na pilha, se ainda houver espaço, lançando uma exceção do tipo **PilhaCheiaException**, caso contrário. Já o método **remover()** remove um elemento do final da pilha, se houver elementos a serem removidos, caso contrário, o método lança uma exceção do tipo **PilhaVaziaException**.

A classe Processo possui apenas um id e um nome, sendo eles do tipo inteiro e String respectivamente, além dos *getters* e *setters* desses atributos. A classe Juiz, por sua vez, possui um atributo nome do tipo String e um método **cadastrarProcesso()** que recebe como parâmetro um processo e o adiciona a pilha de processos do juiz, repassando a exceção **PilhaCheiaException** caso seja lançada pela Pilha.

Por fim, a classe SistemaProcessos possui uma lista de processos e juízes. Além de métodos para adicionar juízes e processos a essas listas. O método mais importante dessa classe é o **distribuirProcessos()**. Tal método deve distribuir todos os processos da lista de processos de forma randômica aos juízes cadastrados. Como cada juiz possui um limite no número de processos (tal valor deve ser definido no construtor da classe Juiz, podendo ser fixo para o

sistema todo), caso todos os juízes já estejam com suas pilhas cheias e sobrem processos, o método deve lançar uma exceção do tipo **ProcessoSemJuizException**.

Crie uma classe Main contendo um método **main()** e instancie um objeto do tipo SistemaProcessos, além de 3 juízes, todos tendo seu número de processos fixos em 5. Em seguida, instancie 18 processos, cadastre-os no SistemaProcessos, chamando o método **cadastraProcesso()**, e distribua-os, chamando o método **distribuirProcessos()**. Trate as exceções adequadamente.

Exercício 3:

Baseando-se no diagrama da Figura 3, implemente o sistema de arquivos descrito nele.

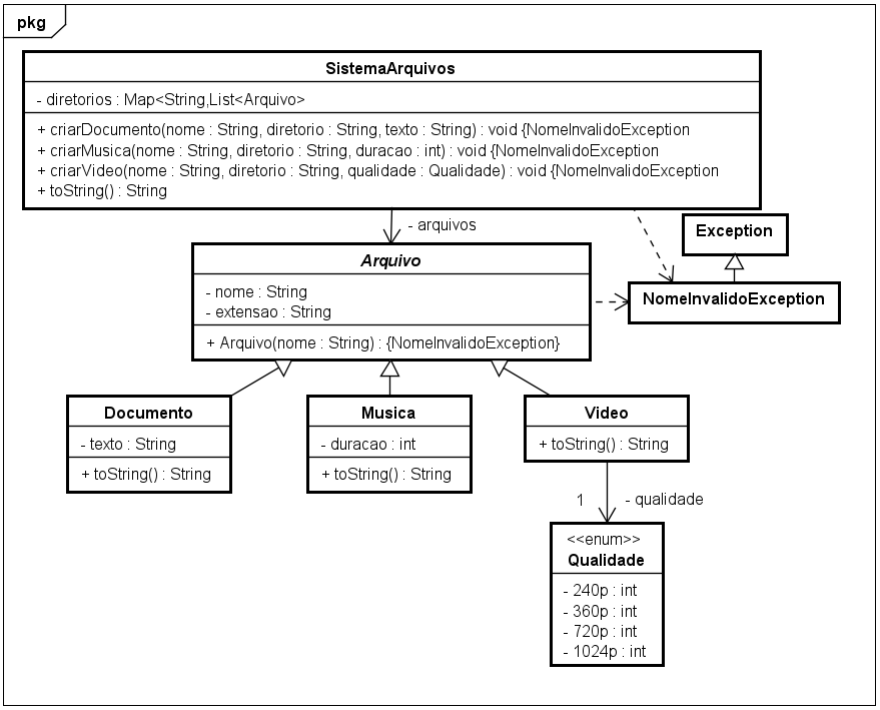

Figure 3: Diagrama UML do Sistema de Arquivos.

A classe NomeInvalidoException estende a classe Exception e deve receber como parâmetro em seu construtor uma String contendo a mensagem do erro. A classe abstrata Arquivo possui dois atributos do tipo String: nome e extensão. Ela é estendida por outras três classes: Documento, Musica e Video, cada qual contendo um atributo. A classe Documento possui um atributo do tipo String chamado texto, além de possuir a extensão **.txt**. Já a classe Musica possui uma duração, um atributo do tipo inteiro, além de possuir a extensão **.mp3**. A classe Video possui um atributo do tipo Qualidade, um enumerável que representa a qualidade de um vídeo, podendo esta ser 240p, 360p, 720p e 1024p. A extensão de um arquivo de vídeo

é **.mp4**. Todas as três classes que estendem a classe Arquivo devem sobreescrever o método **toString()**, tal método deve retornar uma String contendo o nome do arquivo e a extensão, além da duração, no caso das classe Musica, e a qualidade, no caso das classe Video. O construtor dessas classes pode vir a lançar uma exceção do tipo **NomeInvalidoException**. O nome de um arquivo deve seguir as regras a seguir:

- Não pode possuir quebras de linhas;
- Não pode possuir colchetes ou parênteses;
- Nem pode possuir aspas simples ou duplas;
- Deve possuir no mínimo 10 caracteres e no máximo 256.

O construtor da classe Arquivo deve verificar se o nome passado como parâmetro infringe uma dessas regras, caso sim, uma exceção do tipo **NomeInvalidoException** deve ser lançada contendo o problema detectado no nome. A classe SistemaArquivos, por sua vez, possui um Map que mapeia String's para listas de arquivos, tal Map representa os diretórios sendo a String seu nome. Ela possui três métodos para a criação de arquivos, um para documento, outro para musica e outro para video, todas recebem uma String e podem vir a lançar uma exceção do tipo **NomeInvalidoException**, caso o nome do arquivo seja inválido ao instanciar um novo Arquivo e adicionar ao diretório, também passado como parâmetro. O método **toString()** da classe SistemaArquivos, deve retornar uma String na forma:

```
Downloads:

    Imagine Dragons - Radioactive.mp3
    Duracao: 276

    HIMYM S01E01.mp4
    Qualidade: 1080p

    Lista de compras.txt

Area de Trabalho:

    resumo.txt

    notasPOO.txt
```

Crie uma classe Main contendo um método **main()** e instancie um SistemaArquivos. Utilize os métodos de criação de arquivos para criar cinco arquivos variados e adicione-os a dois diretórios diferentes. Trate as exceções adequadamente na **main()**.

Exercício 4:

Implemente em Python um programa que possui as classes Ponto2D, Reta, EspaçoGeometrico e ObjetoSobrepostoException tal como descrito na Figura 4.

A classe ObjetoSobrepostoException estende a classe Exception e deve receber uma mensagem informando que o objeto está sobreposto a outro. A classe Ponto2D possui dois

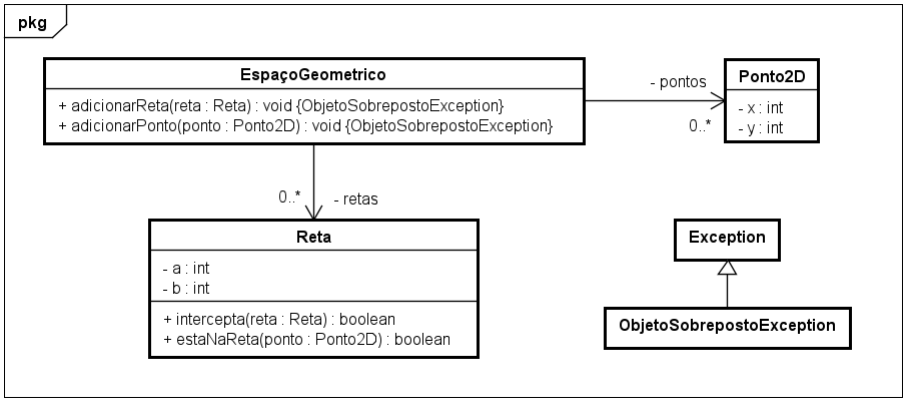

Figure 4: Classes Ponto2D, Reta, EspaçoGeometrico e ObjetoSobrepostoException

atributos que representam a posição em x e em y do ponto no sistema de coordenadas cartesiano. Já a classe Reta possui dois atributos: a e b. Logo, uma reta será uma equação na forma $y = a * x + b$, sendo a e b os atributos da classe. A partir disso, implemente os dois métodos **intercepta()** e **estaNaReta()**. O primeiro deles, o **intercepta()**, recebe outra reta e verifica se existe algum valor de x para o qual as duas retas irão se encontrar (para isso basta igualar as duas equações), caso exista tal valor de x, retorne **true**, caso elas sejam paralelas, retorne **false**. Já o método **estaNaReta()** apenas verifica se o ponto (x, y) pertence a equação de reta. O método deve retornar **true** caso sim e **false** caso não. Na Figura 5 é possível ver exemplos de retas paralelas (vermelha e verde) e retas que se interceptam (verde com azul e vermelha com azul). Na Figura 5 também é possível ver que os pontos A, B e C estão em alguma reta.

Por fim, a classe EspacoGeometrico possui métodos para adicionar retas e pontos a listas internas da classe. O método **adicionarReta()** deve verificar para todas as retas armazenadas no espaço geométrico se a nova reta não intercepta nenhuma delas, caso haja, uma exceção do tipo ObjetoSobrepostoException deve ser lançado. O mesmo vale para o método **adicionarPonto()**, se o ponto a ser adicionado está em alguma das retas uma exceção do tipo ObjetoSobrepostoException deve ser lançada. Instancie os casos apresentados na Figura 5, adicione um ponto D (3,2) e trate as exceções adequadamente.

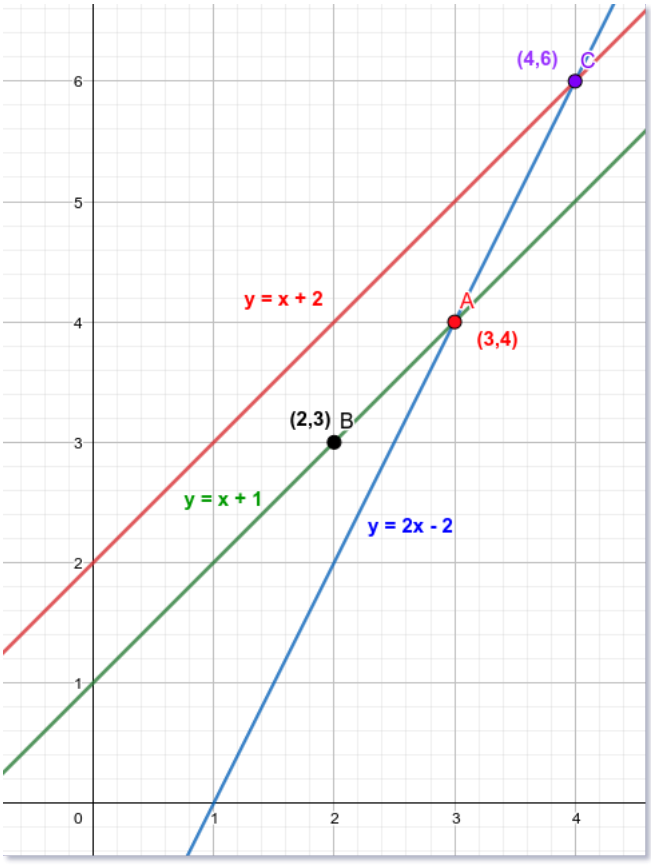

Figure 5: Instanciação de Retas e Pontos.