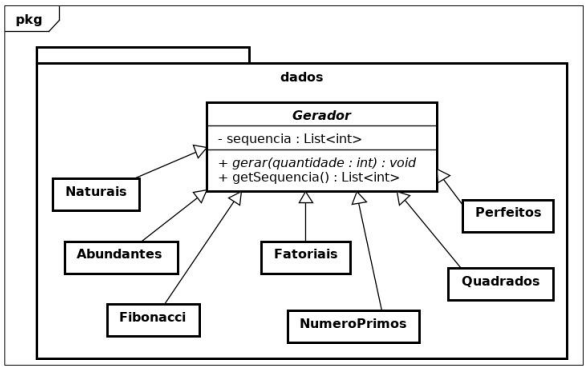
UDESC - Fundação Universidade do Estado de Santa Catarina — Monitor: Vinícius Takeo
Professor: Fabiano Baldo — vtkwki@gmail.com
Disciplina: POO 2020.1 — github.com/takeofriedrich

A lista deve ser entregue até o dia **16/09/2020**, às 23h59, no Moodle, os arquivos devem ser compactados em um arquivo *.zip* ou *.tar*. O arquivo compactado deverá conter o projeto Eclipse ou Netbeans da lista. **Não serão aceitos projetos com os códigos-fonte no formato *.class*!**

Lista 5: Padrões de Projetos em Java

1. A partir da resolução do exemplo da Aula Prática 6: Classes Abstratas em Java, implemente uma classe chamada GeradoresFactory que utiliza do padrão de projetos **Factory Method** para instanciar as diferentes extensões da classe abstrata Gerador de acordo com a figura a seguir:



a) (1.5) A classe GeradoresFactory deve ser um **Singleton** e deve utilizar um Enum TipoGerador a ser passado pelo método **create()** como parâmetro. Esse método deve retornar um objeto do tipo Gerador de acordo com o tipo passado como parâmetro no método.

b) (1.0) Crie um método **main()** e instancie pelo menos cinco objetos de diferentes classes que estendem a classe Gerador. Utilize o método **gerar()** do objeto e exiba no console o resultado.

2. **(2.0)** Utilizando do padrão projetos **Composite**, crie uma classe AgendaComposite que permita a inserção e exibição de novos eventos em uma Agenda. A agenda deve seguir a seguinte estrutura hierárquica:

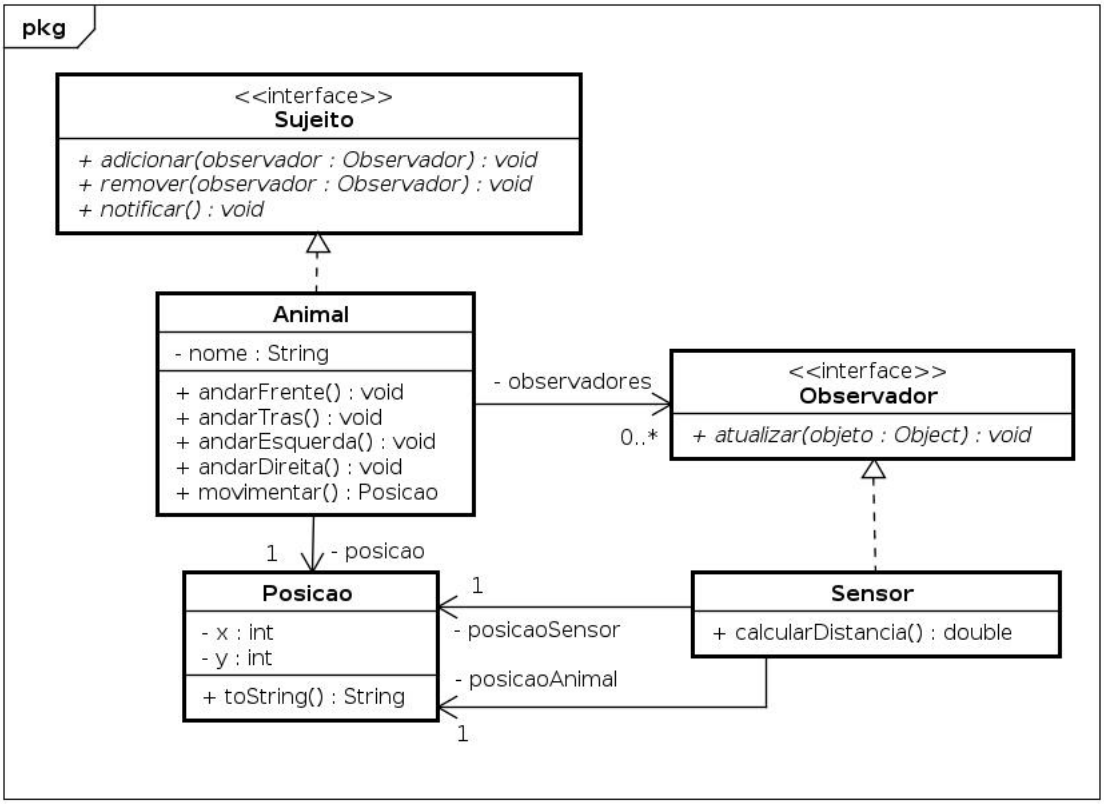
Nivel	Objeto	Atributos
Nivel 0	Ano	int:ano
Nivel 1	Mês	int:mes; String:nome
Nivel 2	Dia	int:dia; String:nome
Nivel 3	Evento	int:hora; String:descrição

A classe AgendaComposite deve conter um método **toString()** que exibe essa estrutura hierárquica. Observe um exemplo de output:

```
2019
    Dezembro
        Quinta 26
            15h00 Voo para Sao Paulo
2020
    Janeiro
        Segunda 13
            10h00 Reuniao
            11h00 Dentista
        Sabado 18
            17h00 Jantar com a familia
    Fevereiro
        Terca 11
            14h00 Voo para Sao Paulo
```

(a) **(1.0)** Crie um método **main()** e instâncie pelo menos um 5 eventos, cada qual pertencendo a um ano, mês e dia. Adicione esses eventos em na AgendaComposite e exiba no console o resultado.

3. A partir do diagrama a seguir implemente o padrão de projetos **Observer** de acordo com os itens:



a) (1.0) Implemente a classe Posição, que representa a posição em que o Animal e o Sensor estão no plano cartesiano. A classe possui dois atributos, x e y, ambos inteiros. Além de seus getters e setters, e dois construtores. Um deles gera as posições em x e y aleatoriamente e o outro as pede como parâmetro. Também implemente o método **toString()** que gera uma String contendo o par x, y, tal como: (1, 3).

b) (1.0) Implemente a classe Sensor e a interface Observador. A classe Sensor possui dois atributos: a posição do animal e a sua posição, além dos seguintes métodos:

- **atualizar():** esse método recebe a posição do animal e atualiza o seu atributo;
- **calcularDistancia():** a partir da posição do animal e da posição do sensor, esse método calcula a distância entre eles usando a seguinte fórmula:
  $Distância = \sqrt{(x_{animal} - x_{sensor})^2 + (y_{animal} - y_{sensor})^2}$

c) (1.5) Implemente a classe Animal e a interface Sujeito. A classe Animal possui os seguintes métodos:

- **adicionar():** adiciona um observador a lista;
- **remover():** remove um observador;
- **notificar():** notifica os observadores passando a posição do animal para eles;
- **andaFrente():** incrementa a posição em y do animal em algum valor e notifica os observadores;
- **andaTras():** decrementa a posição em y do animal em algum valor e notifica os observadores;
- **andaEsquerda():** decrementa a posição em x do animal em algum valor e notifica os observadores;
- **andaDireita():** incrementa a posição em x do animal em algum valor e notifica os observadores;
- **movimentar():** escolhe aleatoriamente uma direção para o animal andar, chamando o correspondente método e retornando uma instância da classe Posição contendo qual foi o deslocamento causado;

d) (1.0) Crie um classe contendo um método main() e instancie um Animal e três sensores. Realize 5 movimentações do animal e exiba no console as posições do animal antes e depois de cada movimento e as distâncias medidas pelos sensores.

4. **Extra (2.0)** Observe as tabelas a seguir que apresentam as políticas de cobrança de ingressos de um determinado cinema:

Tipo de Entrada	Desconto sobre o preço do ingresso
Meia	50%
Inteira	-
Estudante	45%
Doador de Sangue	60%

Tabela 1: Descontos por tipo de ingresso

Dia da Semana	Acréscimo sobre o preço do ingresso
Segunda	-
Terça	-
Quarta	-
Quinta	-
Sexta	-
Sábado	50%
Domingo	50%

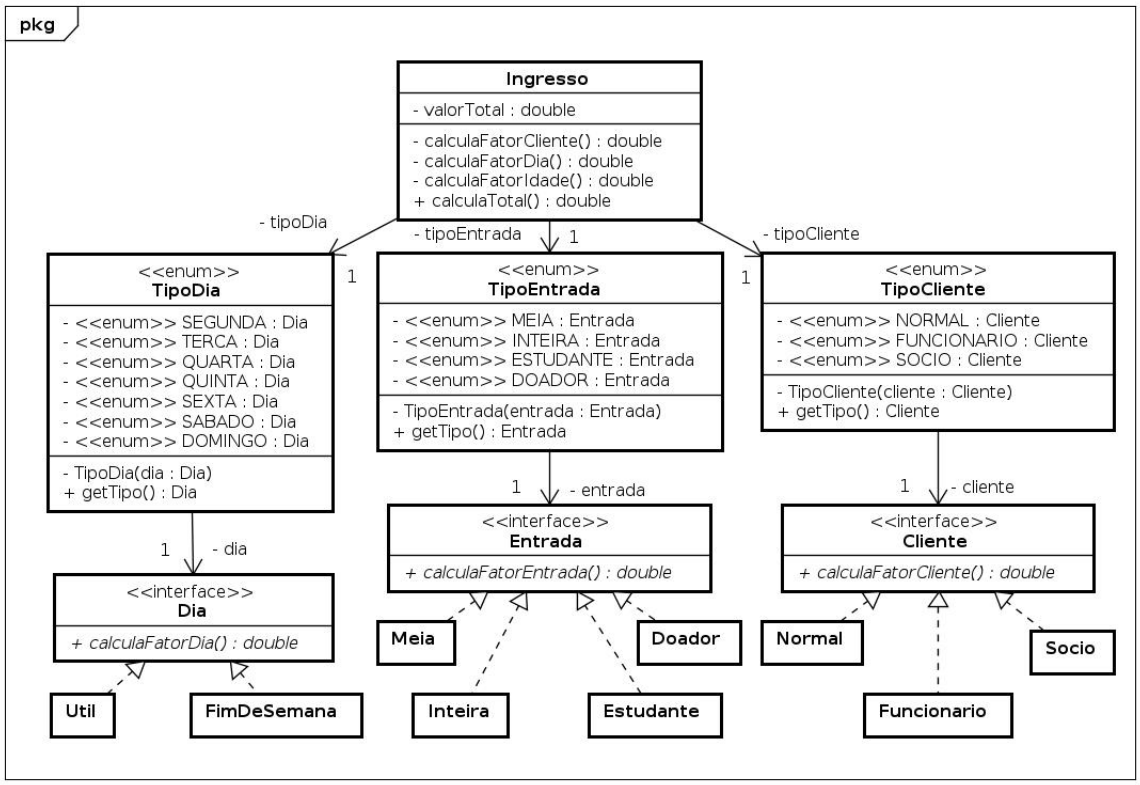
Tabela 2: Cobrança por dia da semana

Tipo de Cliente	Desconto sobre o preço do ingresso
Normal	-
Funcionários	15%
Sócios	25%

Tabela 3: Descontos por tipo de cliente

O cinema possui quatro tipos de entrada: meia, inteira, para estudantes e para doadores de sangue. Entradas do tipo inteira pagam o valor total do ingresso. Enquanto as entradas do tipo meia e para estudantes e doadores de sangue pagam apenas a parcela descrita na tabela. O cinema também leva em consideração o dia em que a sessão irá ocorrer. Aos fins de semana, o preço do ingresso sobe em 50%. Diferentes tipos de clientes recebem diferentes descontos. Funcionários têm direito a 15% e sócios 25%.

O Diagrama UML a seguir apresenta uma possível implementação do padrão **Strategy**. Implemente a regra de negócio apresentada nas tabelas. O método **calculaTotal()** multiplica todos os fatores pelo valor total do ingresso.



(a) Crie um método **main()** e instâncie pelo menos um Ingresso. Escolha um dia, um tipo de entrada e um tipo de cliente e exiba o valor total do ingresso no console.