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 5: Padrões de Projetos

**Exercício 1:**

Crie uma classe em Java que implemente o padrão Iterator, utilizando a interface apresentada em aula. A classe deve receber no construtor um List genérico e permitir a iteração dos elementos, primeiro sobre os elementos de índices pares e depois sobre os elementos de índices impares. Para o exemplo da Figura 1, a saída esperada é A,C,E,G,B,D,F,H.

| A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Figure 1: Exemplo de array.

**Exercício 2:**

A partir da Figura 2 implemente em Java o padrão Factory para as classes descritas no diagrama. Para isso, crie a interface Animal e um enumerável Animais a ser passado para o método **create()** como parâmetro. O método **create()** deve retornar o objeto **Animal** correspondente à subclasse representada pelo valor do enumerado Animais passado como parâmetro. Torne a classe AnimaisFactory um Singleton.

**Exercício 3:**

Crie em Java as classes descritas no diagrama da Figura 3. A classe Servidor possui um atributo IP do tipo String e um método **enviarMensagem()**, que recebe uma mensagem e notifica seus ouvintes (objetos do tipo Cliente) para salvar a mensagem em seus registros. Ainda, ela substitui o valor do atributo ultimaMensagem pela mensagem recém enviada. A classe Servidor implementa a interface Sujeito. Seu método **toString()** deve ser implementado informando o IP do servidor. A sua implementação do método **notificar()** deve passar a última mensagem para os observadores. Toda vez que o servidor enviar uma mensagem os seus observadores devem ser notificados.

Por fim, a classe Cliente implementa a interface Observador. Ela contém o atributo mensagens, que é uma lista de Strings em que cada mensagem contém o IP e a mensagem de

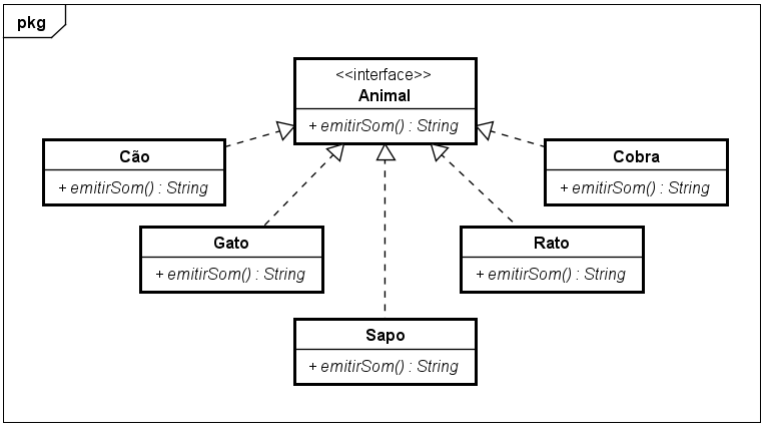


Figure 2: Interface Animal e suas subclasses.

um servidor ao qual o cliente está cadastrado para escutar. Toda vez que um Observador o notificar, o método **atualizar()** deve salvar a mensagem em sua lista.

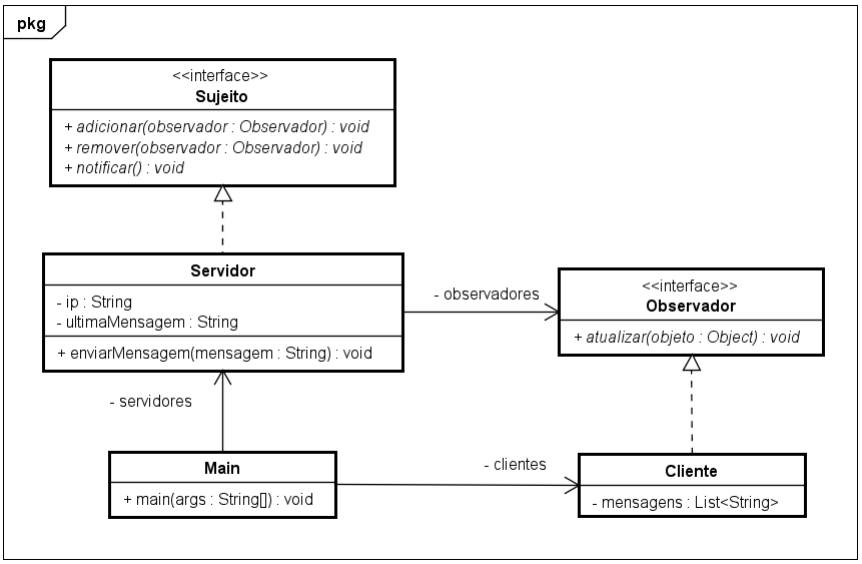


Figure 3: Classes Veículo e Radar implementando o padrão Observer.

Crie um método **main()**, em uma classe Main, e instancie 5 clientes e 3 servidores, gerando IP's aleatórios para eles. Cadastre os clientes nos servidores (pelo menos cada cliente deve estar cadastrado em 2 servidores e não pode estar cadastrado em todos) e envie mensagens (no mínimo 3) a partir dos servidores, exibindo ao final a lista de mensagens dos clientes.

**Exercício 4:**

Certa concessionária de veículos possui uma regra de preços para tornar mais atrativo o preço de seus veículos. Para isso, todo novo veículo possui um preço base. A concessionária aplica acréscimos no preço levando em consideração a cor do veículo, a potência do motor e o tipo de ventilação do veículo. Nas tabelas da Figura 5 é possível ver o valor acrescido para cada uma das opções disponíveis nessa concessionária.

Tendo como base o padrão Strategy representado no diagrama da Figura 4, implemente a lógica de cálculo do valor dos veículos dessa concessionária descrito anteriormente.

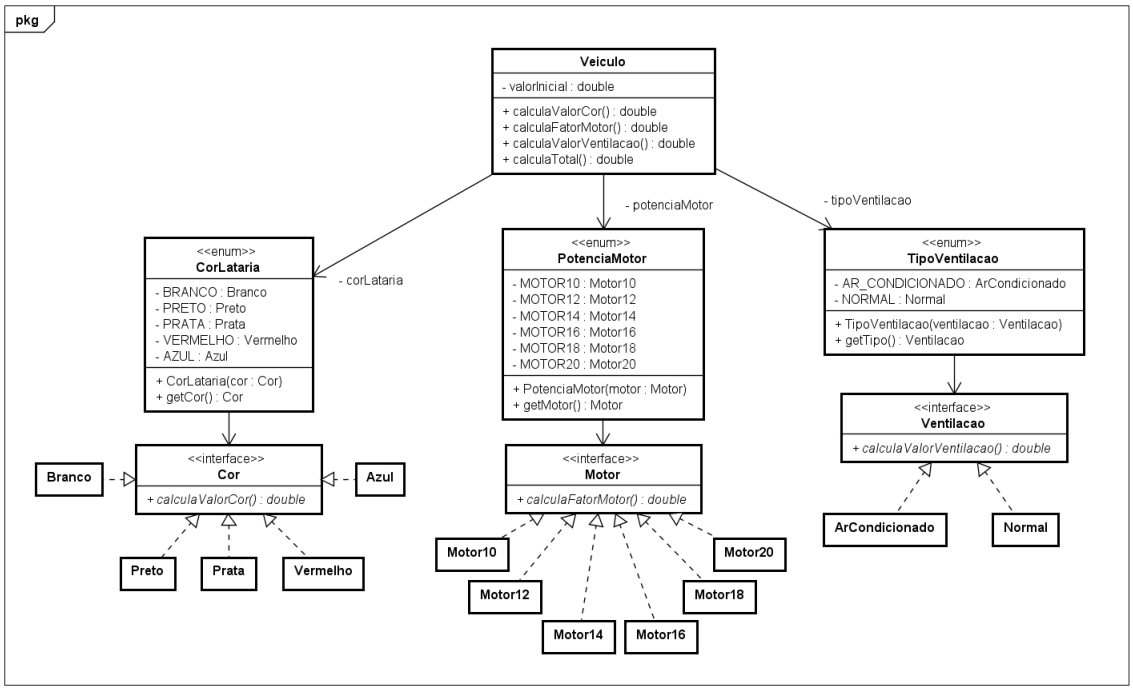


Figure 4: Classe Veiculo e sua lógica de cálculo do preço utilizando o padrão Strategy.

| Cor | Acréscimo |
|---|---|
| Branco | R$0,00 |
| Preto | R$850,00 |
| Prata | R$1100,00 |
| Vermelho | R$1250,00 |
| Azul | R$1150,00 |

| Motor | Acréscimo |
|---|---|
| 1.0 | 0% |
| 1.2 | 6% |
| 1.4 | 9% |
| 1.6 | 11% |
| 1.8 | 14% |
| 2.0 | 17% |

| Ventilação | Acréscimo |
|---|---|
| Natural | R$0,00 |
| Ar-Condicionado | R$1700,00 |

Figure 5: Tabelas utilizadas para o cálculo do preço de um veículo pela concessionária.

Crie um método **main()** e instancie alguns casos de teste, exibindo o valor final do veículo. Utilize como valorInicial o valor de R$40.000,00

## Exercício 5:

Utilizando o padrão de projetos **Composite**, crie em Java ou Python uma classe PaisesComposite que permita a inserção e exibição de novas cidades em uma estrutura hierárquica. A seguinte estrutura hierárquica deve ser seguida:

| Nivel | Objeto | Atributos |
|---|---|---|
| Nivel 0 | Pais | nome:String |
| Nivel 1 | Região | nome:String |
| Nivel 2 | Estado | nome:String ; sigla:String |
| Nivel 3 | Cidade | nome:String ; populacao:int |

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

```
Brasil
    Sul
        Santa Catarina - SC
            Joinville - 597.658 habitantes
            Blumenau - 361.855 habitantes
        Parana - PR
            Curitiba - 1.948.626 habitantes
    Sudeste
        Rio de Janeiro
            Rio de Janeiro - 6.747.815 habitantes
```

Seguindo a implementação proposta nos slides, a Classe PaisesComposite deve limitar a quantidade de países em 1, isto é, apenas o Brasil. Crie uma classe Main e um método **main()** e instancie o Brasil, pelo menos 2 regiões, 3 estados e 5 cidades. Adicione esses objetos na PaisesComposite e exiba no console o resultado.