UFABC	Universidade Federal do ABC Disciplina: Programação Orientada a Objetos 2Q-2019

Atividade 5: Interfaces / Polimorfismo

Obs.: Para entrega: a pasta src com os arquivos .java de cada um dos projetos (os exercícios relacionados podem estar no mesmo projeto). Renomeie cada pasta de modo a identificar o exercício correspondente. Submeta uma pasta compactada na atividade do Tidia.

Exercício 1

Vamos modificar o Exercício 2 da aula passada (enunciado abaixo) para incluir classes e métodos abstratos. Quais classes e métodos poderiam ser modificados para abstratos?
Você pode aproveitar o código disponível em Repositório → Soluções Exercícios.

Você ficou encarregado de criar um sistema para calcular a área e o perímetro de **figuras planas convexas**. Seu trabalho é:

- Criar uma hierarquia de classes que inclua **círculo**, **triângulo**, **quadrado** e **retângulo**.
- As figuras triângulo, quadrado e retângulo devem ser representadas pelo comprimento de seus lados. O círculo será representado pelo seu raio.
- Crie métodos para retornar as medidas de **área** e **perímetro** (note que para o triângulo, é possível calcular a área conhecendo apenas seus lados com a fórmula de Heron).
- Sobrecarregue o método **toString()** para retornar informação detalhada sobre cada figura (tipo, área e perímetro).

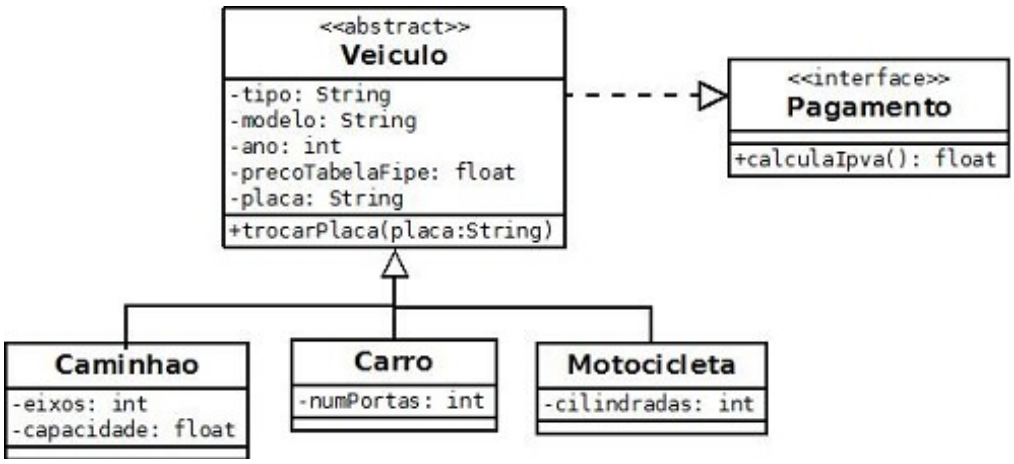
Atenção: todas as figuras devem estar no pacote **geometria**.
Crie um novo pacote chamado **criafiguras** e crie o programa principal que deve ter um vetor de figuras planas. Nesse vetor, instancie uma figura de cada tipo e imprima na tela as informações de cada figura.

Exercício 2

Considere o seguinte diagrama de classes para um sistema simplificado para uma loja de veículos usados. Implemente essas classes considerando seus respectivos atributos, métodos e relacionamentos. Observe que os métodos **get/set** e construtores foram omitidos por simplicidade, porém estes devem constar em sua implementação. Inclua também um método para imprimir os dados de um veículo (**toString()**).
O método **calculaIpva()** faz o cálculo do IPVA considerando o preço de mercado do veículo (atributo **precoTabelaFipe**) multiplicado pela alíquota, que é diferente para cada tipo de veículo, conforme a seguir:

- motos: 2%
- carros: 3%
- caminhões: 1,5%

Considere que veículos com mais de 20 anos de fabricação estão isentos deste imposto.
Crie a classe principal que deve conter uma lista de veículos (vetor). Crie pelo menos 5 veículos e armazene-os no vetor. Apresente os dados e o valor do IPVA para cada veículo e o total que a loja terá que pagar (apresente os valores totais mensal e anual).



Exercício 3

Um sistema usado na Empresa de Entregas ABC depende da implementação de uma classe para guardar objetos da classe Encomenda. Contudo, um funcionário perdeu o código fonte da classe! Por sorte, os projetistas do sistema guardaram a interface que essa classe implementava:

```
public interface GuardaEncomenda{
	void adiciona(Encomenda e); //adiciona uma nova encomenda (não aceita
                                                        //  encomenda duplicada)
	int getSize(); //retorna o número de Encomendas (não nulas)
	void ordena(); //ordena as Encomendas (ponteiros para null vêm por último)
}
```

O seu objetivo é fazer uma nova implementação (seguindo esta interface). O nome da sua classe deve ser **MeuGuardaEncomenda** (para simplificar o exercício, você pode usar um vetor de Encomendas de tamanho máximo 1000 para guardar os objetos).
A classe **Encomenda** possui o método **compareTo()**, que pode ser usado para comparação de instancias da classe Encomenda:

```
public int compareTo(Encomenda e)
```

Retornos do método:
-1: objeto passado como argumento > instância atual
0: objeto passado como argumento = instância atual
1: objeto passado como argumento < instância atual

Além de implementar a interface, o objeto **MeuGuardaEncomenda** também sobrescreve o método **toString()**:

```
@Override
public String toString(){
	String ret = "";
	for(int i = 0; i < tamanho; ++i){
		ret = ret + vetorEncomendas[i] + ", ";
	 }
	 return ret;
}
```

Para testar sua implementação, baixe os arquivos (*bytecode*) **SistemaABC.class**, **Encomenda.class** e **GuardaEncomenda.class** e teste sua implementação (disponíveis na própria atividade). O

resultado deve ser o seguinte:

Enc.#5, Enc.#36, Enc.#40, Enc.#41, Enc.#56, Enc.#62, Enc.#78, Enc.#79, Enc.#83, NUM:99230

Exercício 4

O professor ABC resolveu escrever o código de um sistema para um plano de aulas (link "PlanoErrado.zip", na própria atividade). Entretanto, o código está com vários erros... Você poderia ajudar o professor a corrigir seu programa?

Obs.: Faça apenas mudanças pontuais, não é preciso reestruturar o programa!

Importante: você pode alterar modificadores de acesso, mas altere o menos possível e tente manter o encapsulamento! E não adicione atributos!

A saída do programa deveria ser essa:

```
Plano da disciplina: POO
- [Teorica] Introducao
- [Prova] Avaliacao 1
- [Pratica] Classes
- [Teorica] Construtor
- [Teorica] Polimorfismo
- [Projeto] Avaliacao 2
- [Prova] Avaliacao 3
Formula atual = 0.5 x Avaliacao 1 + 0.5 x Avaliacao 2 + 0.8 x Avaliacao 3

Plano da disciplina: POO
- [Teorica] Introducao
- [Pratica] Classes
- [Teorica] Construtor
- [Teorica] Polimorfismo
- [Projeto] Avaliacao 2
Formula depois de cancelar provas = 0.5 x Avaliacao 2
```