Instituto Federal de Goiás - Câmpus Goiânia
Bacharelado em Sistemas de Informação
Programação Orientada a Objetos I

Nota

Nome do Aluno: _____ Data: __/__/____

Prof. Renan Rodrigues de Oliveira

Trabalho em Sala
Revisão para Prova Prática

1. Analise as questões abaixo, determinando se cada uma das afirmações é verdadeiro (V) ou falso (F). Se falso, explique por quê.
   a) Quando um método de subclasse sobrescrever um método de superclasse, o método sobrescrito da superclasse nunca mais poderá ser acessado.
   b) O relacionamento tem um é implementado via herança.
   c) O propósito principal de uma classe abstrata é fornecer uma superclasse apropriada a partir da qual outras classes podem herdar e, assim, compartilhar um design comum.
   d) Quando uma subclasse redefinir um método de superclasse utilizando a mesma assinatura, diz-se que a subclasse sobrecarrega esse método de superclasse.
   e) A primeira tarefa de qualquer construtor de subclasse é chamar o construtor de sua superclasse direta, explícita ou implicitamente, para assegurar que as variáveis de instância herdadas da superclasse são inicializadas.
   f) As classes abstratas podem ser utilizadas para instanciar objetos, porque são completas.
   g) Métodos abstratos fornecem implementações que podem ser sobrescritos na hierarquia de classes.
   h) Um objeto de uma superclasse pode ser tratado como um objeto de uma subclasse.
   i) O relacionamento é um se aplica entre a classe e suas superclasses, não vice-versa.
   j) Uma interface especifica quais operações são permitidas, mas não como as operações são realizadas.
   k) Uma interface Java descreve o conjunto de métodos, variáveis e implementações que pode ser chamado de um objeto.
   l) Todos os métodos de uma classe abstract devem ser declarados como métodos abstract.

m) Um objeto de uma classe que implementa uma interface pode ser pensado como um objeto desse tipo de interface.

n) Se uma superclasse declarar um método como *abstract*, outra subclasse abstrata deverá implementar esse método.

o) Uma classe de associação é uma classe que está ligada a outra classe, em vez de estar ligada a uma associação.

2. Responda as seguintes questões:

   a) Discuta de que maneira a herança promove a reutilização de software, economia de tempo durante o desenvolvimento de software e ajuda a evitar erros.

   b) Alguns programadores preferem não utilizar o *protected*, porque acreditam que ele quebra o encapsulamento da superclasse. Discuta os méritos relativos de utilizar acesso *protected* vs. utilizar acesso *private* em superclasses.

   c) O que são métodos abstratos? Descreva uma circunstância em que um método abstrato seria apropriado.

   d) Compare e contraste classes abstratas e interfaces. Por que você utilizaria uma classe abstrata? Por que você utilizaria uma interface?

   e) O que são classes de associação?

   f) O que são relacionamentos de dependência. Cite exemplos.

   g) Caracterize sobrescrita e sobrecarga de métodos.

   h) É correto afirmar que uma classe pode estender uma interface?

3. Procure e explique as utilizações não válidas de extensões e implementações. Os itens a seguir mostram exemplos de declarações válidas e inválidas de interfaces e classes.

```
a) class Foo { }

b) class Bar implements Foo { }

c) interface Baz { }

d) interface Fi { }

e) interface Fee implements Baz { }
```

```
f) interface Zee implements Foo { }

g) interface Zoo extends Foo { }

h) interface Boo extends Fi { }

i) class Toon extends Foo, Bar { }

j) class Zoom implements Fi { }
```

4. Dado o código a seguir.

```
public abstract interface Frobnicate {

     public void twiddle (String s);
}
```

Identifique as classes corretas. Caso a implementação não esteja correta, explique por quê.

a)
```
public abstract class Frob implements Frobnicate {
        public abstract void twiddle (String s) { /**/ }
   }
```

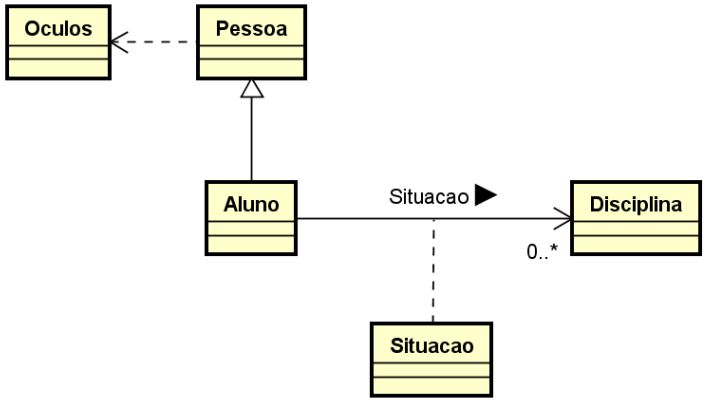
b)
```
public abstract class Frob implements Frobnicate { }
```

c)
```
public class Frob extends Frobnicate {
        public void twiddle (String s) { /**/ }
   }
```

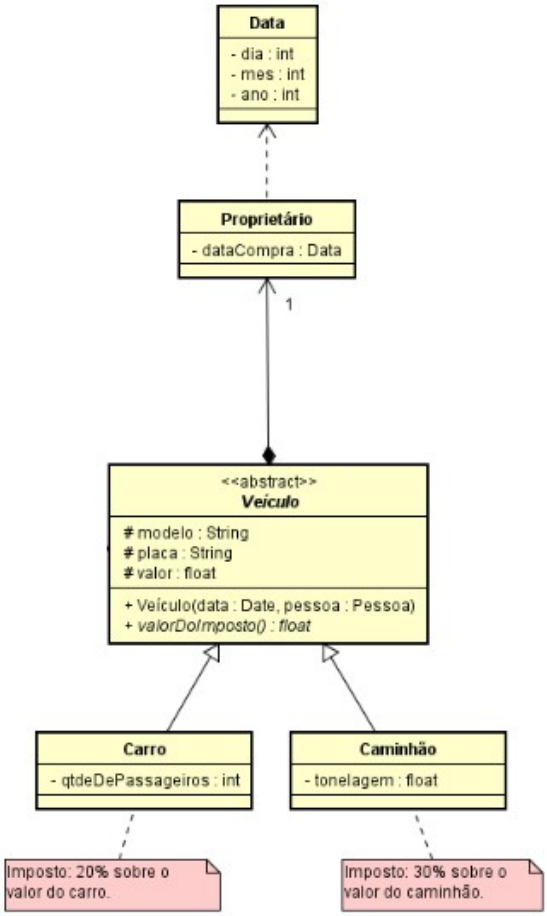
d)
```
public class Frob implements Frobnicate {
        public void twiddle (int i) { /**/ }
   }
```

e)
```
public class Frob implements Frobnicate {
        public void twiddle (String i) { /**/ }
        public void twiddle (int s) { /**/ }
  }
```

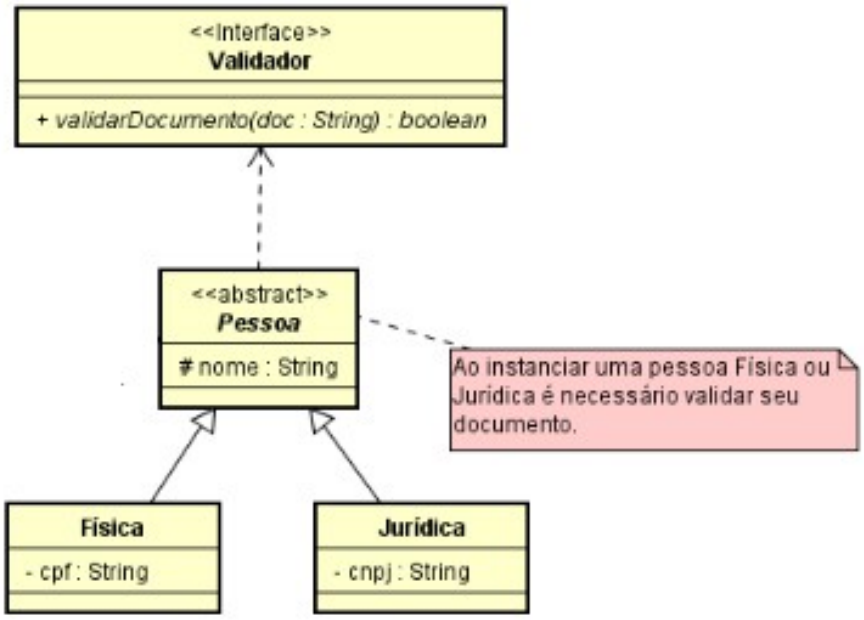
5. Implemente um programa em Java para cada diagrama de classe, criando no mínimo um atributo privado para cada classe. Construa também o programa principal para testar a implementação, instanciando um objeto do tipo aluno com várias disciplinas.



6. Implemente um programa em Java de acordo com o diagrama de classe a seguir. Construa o programa principal, instanciando objetos do tipo Carro e Caminhão.



7. Implemente um programa em Java de acordo com o diagrama de classe a seguir. Construa o programa principal, instanciando objetos do tipo Pessoa Física e Pessoa Jurídica. Utilize os códigos para validação dos Anexos I e II.



ANEXO I

Validar CPF

```
@Override
	public boolean validarDocumento(String doc) {

	    int d1, d2;
	    int digito1, digito2, resto;
	    int digitoCPF;
	    String nDigResult;

	    d1 = d2 = 0;
	    digito1 = digito2 = resto = 0;

	    for (int nCount = 1; nCount < doc.length() -1; nCount++)
	    {
	       digitoCPF = Integer.valueOf (doc.substring(nCount -1, nCount)).intValue();

	       d1 = d1 + ( 11 - nCount ) * digitoCPF;

	       d2 = d2 + ( 12 - nCount ) * digitoCPF;
	    };

	    resto = (d1 % 11);

	    if (resto < 2)
	       digito1 = 0;
	    else
	       digito1 = 11 - resto;

	    d2 += 2 * digito1;

	    resto = (d2 % 11);

	    if (resto < 2)
	       digito2 = 0;
	    else
	       digito2 = 11 - resto;

	    String nDigVerific = doc.substring (doc.length()-2, doc.length());

	    nDigResult = String.valueOf(digito1) + String.valueOf(digito2);

	    return nDigVerific.equals(nDigResult);

	}
```

ANEXO II

Validar CNPJ

```
@Override
```

```
public boolean validarDocumento(String doc) {
char dig13, dig14;
int sm, i, r, num, peso;

	sm = 0;
	peso = 2;
	for (i=11; i>=0; i--) {

		num = (int)(doc.charAt(i) - 48);
		sm = sm + (num * peso);
		peso = peso + 1;
		if (peso == 10)
			peso = 2;
	}

	r = sm % 11;
	if ((r == 0) || (r == 1))
		dig13 = '0';
	else dig13 = (char)((11-r) + 48);

	sm = 0;
	peso = 2;
	for (i=12; i>=0; i--) {
		num = (int)(doc.charAt(i)- 48);
		sm = sm + (num * peso);
		peso = peso + 1;
		if (peso == 10)
			peso = 2;
	}

	r = sm % 11;
	if ((r == 0) || (r == 1))
		dig14 = '0';
	else dig14 = (char)((11-r) + 48);

	if ((dig13 == doc.charAt(12)) && (dig14 == doc.charAt(13)))
		return(true);
	else return(false);
}
```