|  |  |  |
| --- | --- | --- |
| Logo_UFOP | **Universidade Federal de Ouro Preto – UFOP****Instituto de Ciências Exatas e Biológicas – ICEB****Departamento de Computação – DECOM****Disciplina: Programação Orientada a Objetos****Professor: Marco Antonio M. Carvalho** | decom_novo |

**Lista de Exercícios 10 – Polimorfismo**

**Instruções**

* *Todos os exercícios que envolvem programas devem ser resolvidospor programas em linguagem* ***Java****;*
* *Na solução dos exercícios, devem ser utilizados os conceitos listados no cabeçalho desta lista;*
* *Eventuais dúvidas podem ser sanadas com o professor.*
1. Implemente a hierarquia **Formato** como descrito pelo diagrama abaixo e por este texto:
	1. Cada **FormaBidimensional**deve conter um método *getArea()* que calcula sua área;
	2. Cada **FormaTridimensional** deve conter os métodos *getArea()* e *getVolume()*, para calcular área e volume, respectivamente;
	3. Crie um *driver* com um vetor de referência a objetos da classe *Forma*, cada uma relativa a um dos objetos de cada classe concreta da hierarquia;
	4. O *driver* deve imprimir a classe de origem de cada objeto, o que deve ser determinado em tempo de execução;
	5. O driver deve imprimir, para cada objeto apontado, se trata-se de uma **FormaBidimensional** ou **FormaTridimensional**, o que deve ser determinado em tempo de execução
		1. Se a forma for bidimensional, imprima sua área;
		2. Se a forma for tridimensional, imprima sua área e volume.



1. Adicione um método *desenhar()* à classe *FormaBidimensional* e à hierarquia que a tem como classe base.
	1. O construtor de cada classe concreta deverá especificar o tamanho (em quantidade de caracteres) e o caractere de preenchimento para desenhar cada forma, gerando figuras como as apresentadas abaixo;
	2. Crie um *driver* com um vetor de referências a objetos da classe *FormaBidimensional*, cada uma relativa a um dos objetos de cada classe concreta da hierarquia
		1. Invoque o método *desenhar()* de cada objeto apontado.

XXX A

XXX A A OOO

XXX A A A OOOOO

 A A A A OOO

1. Crie uma hierarquia de herança que um banco possa utilizar para representar dois tipos de conta: poupança e conta corrente. Todos os clientes deste banco podem depositar e sacar dinheiro de suas contas.



A classe ***Conta*** deve possuir um atributo que represente o saldo da conta. Este atributo deve ser inicializado através de um construtor parametrizado que valide o valor enviado como parâmetro. Devem ser criados métodos para mostrar o saldo, para crédito e para débito na conta. Crie um *getter* e um *setter* para o atributo.

A classe **Poupanca** deve possuir um atributo relacionado à variação (rendimento), com métodos *getter*, *setter* e construtor. Crie também um método *CalculaRendimento*(), que informa o valor do saldo multiplicado pela taxa de rendimento.

A classe **ContaCorrente** deve incluir um atributo que represente a taxa cobrada por cada transação de crédito/débito, com *getter*, *setter* e construtor. Sobrescreva os métodos de crédito e débito para descontar o valor de tal taxa a cada transação bem sucedida.

Adicione o comportamento polimórfico conforme descrito:

* 1. Os métodos de crédito, débito e saldo em conta devem ser abstratos na classe base;
	2. Crie um *driver* com um vetor de referências para *Conta*, cada referência deve ser relativa a um dos objetos de cada classe concreta da hierarquia
		1. Invoque os métodos de débito e crédito de cada objeto apontado.
	3. Percorra o vetor e determine em tempo de execução qual é o tipo de cada conta:
		1. Se for uma poupança, calcule seu rendimento através do método *CalculaRendimento()*;
		2. Se for uma conta corrente, apenas mostre seu saldo.
1. Modifique o sistema de pagamento visto em aula para incluir a subclasse **EmpregadoPorPeca**, que herda diretamente da classe *Empregado*. Esta classe representa empregados que recebem por tarefa cumprida, e deve possuir os atributos *valorPorPeca* e *quantidade*. Forneça uma implementação concreta do método *getValorAPagar*(), que calcula o valor a ser pago para o empregado. Modifique a classe **TesteInterfacePagavel** para processor polimorficamente um objeto desta nova classe também.