|  |  |  |
| --- | --- | --- |
| 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 05 – Polimorfismo e Exceções**

**Instruções**

* *Todos os exercícios que envolvem programas devem ser resolvidos por programas em linguagem* ***C++****;*
* *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. Cada classe deve possuir um destrutor específico, que imprima o nome da própria classe;
   4. Crie um *driver* com um vetor de ponteiros para *Forma*, cada ponteiro deve apontar para um dos objetos de cada classe concreta da hierarquia;
   5. O *driver* deve imprimir a classe de origem do objeto apontado por cada ponteiro, o que deve ser determinado em tempo de execução;
   6. 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.
   7. Processando os objetos polimorficamente, o destrutor deve ser invocado através do método delete().



1. Implemente as classes e o relacionamento entre classes descritos no diagrama UML abaixo. Note que apenas as classes do último nível da hierarquia são concretas. No *driver*, crie um vetor de ponteiros para classe *Laurasiatheria*, em cada posição, adicione ponteiros para cada uma das classes concretas, em ordem alfabética.
   1. Utilizando o comportamento polimórfico, utilize os ponteiros criados para invocar o método *Som()* adequado para cada classe apontada;
   2. Determine, em tempo de execução, a classe de cada objeto apontado pelo vetor;



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 ponteiros para *FormaBidimensional*, cada ponteiro deve apontar para 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. Considerando a hierarquia de herança do exercício 5 da lista anterior (reproduzido abaixo), adicione o comportamento polimórfico conforme descrito:
   1. Os métodos de crédito, débito e saldo em conta devem ser virtuais puros na classe base;
      1. Portanto, a implementação dos mesmos não necessita chamar os métodos originais.
   2. Crie um *driver* com um vetor de ponteiros para *Conta*, cada ponteiro deve apontar para 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.

***Exercício 5, lista anterior***

*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. Note que se o valor de débito for maior que o saldo, deve ser impressa uma mensagem de erro. 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. Redefina os métodos de crédito e débito para descontar o valor de tal taxa a cada transação bem sucedida. Os métodos originais da classe Conta devem ser invocados na redefinição. Crie um driver para testar sua hierarquia.*

1. Crie um programa que exagera na alocação dinâmica de memória através do operador *new*. Trate a exceção gerada pela exaustão de memória, utilizando a exceção já existente.
2. Use herança para criar duas classes derivadas da classe *runtime\_error*. Crie um programa *driver* que mostra que, havendo um *catch* que especifica a classe base pode funcionar para as classes derivadas. Em outras palavras, um catch que especifica a classe base também pode capturar as exceções das classes derivadas.
3. Crie um programa que gera pelo menos dois tipos diferentes de exceções e captura todas com o argumento *default* (*...*) para o *catch*.