Anotações Alura

Java Poliformismo: entende herança e interfaces

* Atributos normalmente são privados.
* Dica: colocar apenas atributos que são realmente necessários deixando a classe o menos possível
* Se vc n cria nenhum construtor, o compilador insere um padrão (sem nada); Um construtor sempre tem o nome da classe, mas diferente de um método ele não retorna nada.
* Correto, o construtor padrão é aquele que não tem nenhum parâmetro.
* Correto, um construtor é chamado na inicialização/criação do objeto.
* Muitos if’s é sempre uma má pratica (tendem a nunca parar de crescer);
* Classe mãe ou Base class ou Super class
* Classe filha é a que herda
* Extends é o processo de herdar, indicando que class filha é um tipo da classe mãe
* “protected” significa publico para os filhos;
* Protected esta entre o private e o public
* O “this” aponta para esta classe. No caso de classe filha o atributo esta na classe mae, de modo que eh boa pratica trocar o “this” por “super”.
* Reescrita/sobrescrita: quando um método existente na classe mae é reescrito na classe filha
* Primeiro pilar da herança eh a reutilização;
* Polimorfismo é um objeto que pode ser referenciado através de uma referencia do mesmo tipo ou de uma referencia mais genérica(mae);
* Mesmo eu usando uma referencia genérica, ao executar o código sempre sera executado o método especifico;
* Na reutilização (herança) vc herda atributos, mas não herda os construtores. O construtor eh so DESTA classe;
* O Java sempre chama o construtor padrão, nunca construtor específicos;
* No caso, para chamar um construtor especifico, sera necessário iniciar o objeto indicando os parâmetros do construtor especifico, chamando no corpo também com o “super(parâmetros)” dentro.
* Abstract na classe impede de instancia objetos da classe
* Abstract no método eh para indicar método sem corpo, ou seja sem implemento, fazendo com o código busque nas “filhas” a implementacao
* O Java não faz herança múltipla;
* Interface eh uma classe abstrata com todos os métodos abstratos, ou seja, tem nada concreto.
* Na interface eliminamos td q eh concreto e deixamos apenas as assinaturas dos métodos;
* Void é usado quando o método não retorna nada. main (): nome do método, se tem os parênteses então é um método. String [] args: args é o nome da variável local do método e String [] o tipo de dado, este tipo é um vetor de Strings.
* Mesmo que em Java eu so possa extender a uma classe, eu posso assina/implementar quantas eu quiser;
* Interfaces é uma possibilidade para o poliformismo;
* Só vale a pena usar reutilização de código (herança) se for pra usar o poliformismo junto;
* Da mesma forma se não for usar poliformismo, melhor usar apenas interfaces;
* Se vc n quer usar reutilização, pode-se usar composição;
* Public e abstract já eh patrão na implementação de método numa interface, logo, n precisa usar.
* Uma thread eh uma linha de execução, podendo definir uma prioridade (método setPriority(Thread.MAX\_PRIORITY));
* Um programa em java sempre começa do método main
* Stack = pilha

Exceções:

try {

//o bloco try sinaliza que esse trecho de código é perigoso. E se acontece uma exceção, então ele executa o catch

} catch(exceção\_encontrada ex) {

Sysout(“exceçãoencontrada”)

}

No mundo java, o local onde os objetos são criado são chamados de HEAP (“RIP”);

“Jogar uma bomba” so funciona com exceções (throw):

**throw new** ArithmeticException(**"deu errado"**);