**PRÁTICAS DE PROGRAMAÇÃO ORIENTADA A OBJETOS**

*Prof. Luiz Merschmann*

Os pilares da POO são abstração, encapsulamento, polimorfismo e herança.

Abstração é trabalhar objetos com as características principais, sem observar detalhes.

Objeto é a abstração de uma entidade abstrata ou concreta do mundo real.

- Atributos: características.

- Métodos: comportamentos.

\* Construtores, destrutores (garbage collector no Java), getters e setters.

\* Sobrecarga é quando um método de características semelhantes é implementado mais de uma vez com mesmo nome, mas número, tipo ou ordem dos parâmetros diferente.

- Estado: características em um determinado momento.

- Instanciação = **new**.

- Identidade de um objeto é o endereço na memória Heap.

Mensagem é a requisição de dados de um objeto por outro por meio de uma chamada de método.

Classe é um molde para criação de objetos (abstração de um conjunto de objetos).

== compara endereço de memória, para comparar estado de objetos fazer “x.equals(y)”.

Encapsulamento é um conceito que protege informações de classes (“se e como os atributos podem ser alterados”).

- Atributos devem sempre ter modificador “private”.

- Métodos private para separar métodos grandes ou trechos iguais de métodos diferentes.

- Armazenar return de get em variável e alterá-la também altera objeto. Usar .clone() (pode ser necessário sobrescrever método – Classe extends Object implements Cloneable).

Atributos estáticos são atributos de classe, existe um único valor que representa todos os objetos de uma classe, geralmente incrementado no construtor.

Modificador final para declaração de constantes (não muda depois da inicialização). Se inicialização for na declaração, declarar como atributo estático. Objeto final apenas não troca de endereço de memória. Este modificador também pode ser usado em métodos e classes e, nestes casos, os métodos não podem ser sobrescritos e as classes não podem se tornar superclasses, respectivamente.

Métodos estáticos são chamados de métodos de classe, não precisam ter um objeto da classe instanciados para serem chamados. Métodos estáticos não podem chamar outros métodos e atributos não estáticos.

Composição é um relacionamento todo/parte onde um objeto de uma classe contém objetos de outras como atributos. “Se o todo deixar de existir, não faz sentido pensar nas partes separadamente.” Ex: pedido e itens.

- Relacionamento do tipo “tem um/tem vários”.

- Método get deve retornar clone, não fazer isso fere o encapsulamento.

Agregação é um relacionamento todo/parte ou “tem um”, mas diferente da composição, o todo não controla o tempo de vida das partes, caso ele deixe de existir, as partes continuam existindo.

- Ex: grupo de estudo e alunos.

- Geralmente implementado como uma estrutura de dados (atributo) que contém objetos de uma classe parte.

Associação é um relacionamento onde as duas partes coexistem sem dependência, ou seja, um objeto não é propriedade de outo.

- Geralmente um objeto chama métodos de outro.

Javadoc para gerar a documentação de um código. A documentação é essencial para se ter uma maneira fácil de entender o código, e exclui a necessidade de lê-lo para saber como funciona uma determinada classe ou método.

- Para usar a ferramenta Javadoc, basta que o código esteja bem comentado, em um formato especifico:

/\*\*

\* Comentar assim.

\*/

- IDEs comumente possuem integração com a ferramenta.

- Usando o terminal, executar comando: $ javadoc –d *nomeDaPastaDestino* \*.java

- É possível deixar a documentação mais completa utilizando, entre “javadoc” e “-d” o comando “–link <https://docs.oracle.com/javase/8/docs/api/>” para adicionar a documentação oficial do java.

- Locais importantes de usar a documentação: classes, métodos públicos.

- Palavras chaves são usadas para aumentar a completude da documentação e sempre começam com @. Ex: @author, @version ,@param, @return e @throws.

Diagrama UML é construído para se ter uma visão gráfica dos relacionamentos entre classes.

- Retângulos compostos por nome da classe, atributos (: tipo) e métodos ((param): return) + setas que representam os relacionamentos.

- Visibilidade: (-) private, (#) protected, (~) pacote, (+) public

- Flexível, pode colocar a quantidade de detalhes que julgar necessário, apenas nome da classe é obrigatório. (...) para representar que há mais atributos ou métodos.

- Relacionamentos: composição (losango preenchido no todo, a parte não deve constar nos atributos do todo), agregação (losango vazado no todo, a parte não deve constar nos atributos do todo), associação (linha ligando, opcionalmente pode-se colocar uma seta no sentido da chamada de método), associação unária (linha ligando classe a ela mesma com nome que representa o relacionamento), herança (triângulo vazado na superclasse), interface (seta vazada com linha tracejada)

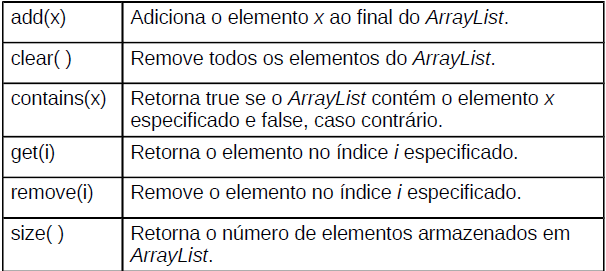
- Multiplicidade define o número de objetos em cada extremidade do relacionamento (n..\* | n >= 0, \* = muitos).

- Atributos e métodos estáticos ficam sublinhados.

ArrayList é um vetor dinamicamente redimensionável.

- import java.util.ArrayList;

- ArrayList<Tipo> exemplo = new ArrayList<Tipo>();



- for(Tipo ex : exemplo) {//execução;}

- Não é possível fazer ArrayList de tipos primitivos. Para isso usar boxing (empacotamento em classe), basta colocar nome do tipo com primeira letra maiúscula.

Herança é um dos pilares da POO e faz com que classes filhas possuam todos os atributos e métodos da superclasse. “É um”.

- Evita duplicação de código, facilitando manutenção.

- public class ClasseX extends ClasseY {...}.

- No construtor, além de inicializar os atributos da subclasse, adicionar “super(@param)” que é o construtor da superclasse.

- Sobrescrita de método ocorre quando métodos de uma mesma classe ou subclasses e superclasses tem a mesma assinatura. Uma boa prática é adicionar “@Override” para referir a sobrescrita e compilador conferir essa informação.

Polimorfismo é a possibilidade de um método ou variável ser definido de várias formas dependendo do momento, como tipo dinâmico de uma variável ou busca de sobrescrita de método. Pode-se atribuir à variáveis de um tipo de superclasse, tipos de subclasses e trocar ao longo do tempo de vida.

- Tipo estático de uma variável é o usado na declaração. Ex: Item i;

- Tipo dinâmico é a vinculação atual. Ex: i = new Cd();

Acoplamento é a conectividade entre classes, ou seja, o quanto elas se relacionam por chamadas de métodos. Deseja-se enfraquecer esse parâmetro, pois a dependência de classes gera dificuldade de manutenção.

Coesão refere-se a quantidade e diversidade de tarefas que um método, classe ou pacote executa. Busca-se um alto grau de coesão, ou seja, a unidade executar uma tarefa bem definida, isso propicia reutilização de código.

Más práticas e soluções / boas práticas:

- Duplicação de código: criar novos métodos.

- Atributos públicos: modificar visibilidade para private e criar getters.

- HashMap é uma coleção do tipo chave/valor (dicionário em Python).

- Refatorar é reorganizar as classes e métodos para adaptá-los a alterações.

Classes e métodos abstratos:

- Métodos abstratos não possuem implementação na classe pai e precisam ser sobrescritos nas classes filhas.

- Classes com métodos abstratos também devem ser declaradas abstratas, e não podem instanciar objetos.

- Classes podem ser abstratas mesmo que não possuam métodos abstratos, isso garante a não instanciação de objetos daquela classe.

- Apesar de classes abstratas não serem instanciáveis, são tipos válidos (podem declarar variáveis e referenciar objetos de classes filhas ou fazer, por exemplo, ArrayList<ClasseAbstrata>).

Interfaces:

- Uma interface Java é a especificação de um tipo (na forma de um nome de tipo e um conjunto de métodos) que não define nenhuma implementação para os métodos.

- Métodos não precisam utilizar a palavra chave *abstract*.

- Visibilidade dos métodos não precisam ser declaradas.

- Apenas atributos constantes (public, static e final) são permitidos em uma interface (não precisa declarar public, static e final).

- public class ClasseX imlements ClasseY {...}.

- Existe herança múltipla de interface.

Teste de sistema: testar o sistema completo como usuário final.

Teste de unidade: testar partes individuais do código, que podem ser de vários tamanhos, como um método à um conjunto de classes.

- ClasseXTeste: Criação de classe auxiliar, contendo a main, para executar os métodos da ClasseX. Classe auxiliar não é parte do sistema final.

- Ler código.

- Teste de mesa: fazer passo a passo do código em uma tabela.

- Explicar à alguém: quando você explica seu código à um terceiro estimula a organizar o pensamento e ajuda a perceber erros, além de que a outra pessoa pode encontrar o erro.

- Instruções de impressão: impressões de informações que podem ajudar a detectar em qual parte se encontra o erro.

- Debugger da IDE: escolha de breakpoint e a partir dele é possível ver estados de objetos e valores de varáveis, executar comandos passo a passo e ver pilha de execução.

Métodos genéricos dependem de um tipo parametrizado <T> para funcionar. Ajuda quando dois ou mais métodos fazem a mesma ação para tipos diferentes.

- public static <T> void printArray(T[] inputArray) {...}.

- Aceita mais de um tipo separado por vírgula.

Classes genéricas dependem de um tipo parametrizado <T>.

- Aceita mais de um tipo separado por vírgula.

- Classe Collections.