>>>>>>>> 24/05/2022 <<<<<<<<
Programação Orientada a Objetos
Prof. Thiago Leite

##

Introdução
- Apresentação inicial
Professor e ocupações

- Objetivos do curso
Compreensao dos conceitos de OO; curso baseado no livro do professor, da Casa do Código (ver?!)
Por que usar? - Fundamentos? - Estrutura? - Relações? - Organização? - Próximos passos?
Requisitos - logica de programacao + vontade de aprender + linguagem de programação JDK11 + IDE IntelliJ

2) Por que usar?
- Por que usar?
Antes da POO, havia o padrão estruturado de programação. Estruturado tem representação + simplista; foca operações (funções) e dados; recursos limitados (baixo nível); como fazer.

POO foca tem repres. + realista; modelagem de entidade e interações; mais recursos (alto nível), trabalhar com maior nível de abstração; o que fazer.
Vantagens: coesao de unidades de código; acomplamentos mais flexíveis, gap semantico reduzido, coletor de lixo limpa a memoria do que não é usado.

3) Fundamentos
- Fundamentos
Conceito = análise, projeto e programação de sistemas de software baseado na composição e interação entre diversas unidades de software chamadas de OBJETOS.
Fundamentos = abstração (isolar caracteristicas de objeto, focar no essencial e descartar o acidental); reuso (criar novas unidades de código a partir de existentes); encapsu-lamento esconde complexidades e protege dados.

- Exercício
Exercício: modelar uma entidade livro (tema, autor, editora, edição, ISBN, ano publicação, quantidade de páginas, titulo, assunto) sao caracteristicas mais relevantes. O nicho a que se destina o programa influencia na modelagem da entidade, como no livro, pode ser para leitor/comprador, editora, loja online etc. Cada nicho tem necessidade de diferente sistema.

4) Estrutura básica da OO
Classe, Atributo, Método, Objeto, Mensagem
- Classe - Conceitos

Estrutura que abstrai conj de objetos com caracteristicas similares. Ex.: bola, carro, viagem, comprador etc. Como criar: substantivos, nomes significativos, contexto. Contexto para pessoa: cliente (loja), estudante (curso), leitor (biblioteca). Código Java como define uma classe:

```
class Carro {

}
```

Não é muito usual criar classe vazia, como feito em Python.

- Classe - Exercício
Exercicio: criar a classe Carro.java

- Atributo - Conceitos
Elemento de uma classe responsável por definir sua estrutura de dados. Caracterização de classes e futuros objetos. Ex.: bola -> diametro, carro -> cor, aluno -> nome, livro -> pgs

Atributo x Variável = (A) o que é proprio e peculiar a algo ou alguem X (V) o que pode variar ou mudar, inconstante. Dicas: sustantivos/adjetivo, nomes significativos, contexto, abstração, tipos adequados (int, float, string). Entender do negocio ajuda a definir tipo adequado do atributo. Como pode ser definido um atributo:

```
class Carro {
        int portas;
}
```

- Atributo - Exercício
Definir 3 atributos pra classe Carro: cor, modelo, tanque L

- Método - Conceito
Porção de código (sub-rotina) disponibilizada pela classe e é executado quando é feita a sua requisição. Identifica quais serviços e ações a classe oferece. Ex.: carro-> ligar venda-> calcular total, comprador-> realizar troca
Criação de metodos: visibilidade, retorno, nome, parametros
Dicas: verbos, nomes significativo, contexto. Como se define:

```
class Carro {
        void frear() {
        ...
        }
}
```

Metodo especiais: contrutor e destrutor
(C) constroi objetos a partir das classes, se necessario, provê valores iniciais aos atributos da classe e objetos.

```
class Carro {
        Carro () {
        ...
        }
}
```

Mesmo que não passe parâmetros iniciais e use um construtor só pra criar objeto vazio. Não precisa definir um construtor, a lgm possibilita que seja chamado o construtor pra criar o objeto, mesmo sem codificar o construtor de forma explícita.

(D) auxilia na destruição do objeto, é chamado pelo coletor de lixo da lgm pra liberar a memoria. Como definir:

```
class Carro {
        void finalize () {
        ...
        }
}
```

Sobrecarga: mudar assinatura de acordo com a necessidade
Assinatura = nome + parâmetros. Exemplos:
m1 ()
m1 (int i)
m1 (float f)
m1 (String s, long l)
m1 (long l, String s)
Por que usar sobrecarga: POO oferece representação mais fidedigna do mundo real; mesmo método pode ser usado pra implementar diferentes operações relacionadas, através de outros metodos com novos parâmetros. Exemplo: um metodo calcularVenda pode ter métodos com parametros verDesconto, verTotal, compraParcelada, compraAvista. Isto ajuda manter a abstração alvo e facilitar entendimento do software.

- Método - Exercício
Criar metodos pra classe Carro: getter, setter, metodo pra calcular quanto gasta pra encher o tanque do carro, fazer uma sobrecarga do metodo construtor pra criar objetos.

- Objeto e Mensagem - Conceito
Representação de um conceito/entidade do mundo real, que pode ser física ou conceitual e tem significado definido para certo software.
A classe é estática, um molde, é definida pra saber depois como criar e manipular os objetos, que ficam na memória do computador. Objetos são instâncias de classes, que executam e fazem acontecer. Como definir objeto:
Carro carro = new Carro ();
Mensagem é o processo de ativação de um método de um objeto, quando há uma requisição ou chamada ao método que dispara a execução do comportamento descrito na classe. Vai direto pra classe se a requisção é metodo estático. Troca de mensagens entre objetos e classes fazem o sistema funcionar. Como é passada uma mensagem:
Carro carro = new Carro();
carro.<metodo>;

Carro.<metodo>;

Conceitos mais avançados a explorar depois!
Instância x Estático: atributos e métodos podem se comportar sob essas 2 formas.
Estado de um Objeto diz respeito a valores de seus atributos
Identidade de um Objeto importa pra saber como identificar de forma única um objeto.
Representação numérica de um objeto trabalha-se junto com a identificação do objeto.
Representação padrao de um objeto é o que pode identificá-lo, forma mais simples e compreensivel dentre todos os atributos que o compõem.

- Objeto e Mensagem - Exercício
Criar objetos da classe Carro, usar metodos set/get, se aplicaveis, pra definir valores de atributos e exibir os valores get, passe mensagem do calculo de total pra encher o tanque.

Link do video de execução do programa Carro.java:
https://somup.com/c3hTeyt2XM

5) As relações: Herança, Associação e Interface
Conceitos relacionais: herança, associação, interface
- Herança - Conceito
HERANÇA é relacionamento entre classes em que uma subclasse (classe filha, classe derivada) é extensão, um subtipo de outra superclasse (classe pai, classe mãe, classe base). A subclasse reaproveita os atributos e métodos da superclasse, também pode definir seus próprios membros além dos herdados. O resuso é uma consequência da herança, mas não é seu fim, pois a associação também faz reuso. Como fazer:

```
class A extends B {
        ...
}
```

- Herança - Exercício 1
Exercicio: criar classe "Veículo", "Carro2", "Moto" e "Caminhão".
Nem tudo que é definido na classe mae estará na subclasse.

- Herança - Tipos
Tipos: simples e multipla
Her. Simples = classe filha tem só uma classe mae, por ex:
Funcionario => Gerente + Vendedor + Faxineiro
Her. Multipla = classe filha tem 1 ou + classes mae, por ex.:
Estagiário => Estudante e Funcionario
Java não tem herança múltipla, pois pode haver conflito se forem usados atributos e metodos com mesmo nome ou haveria um custo alto para verificar qual a origem dos elementos.

- Herança - Upcast e Downcast
Upcast é subir o tipo na sua hierarquia de classe, como quando um gerente é subido para a classe funcionario. Já o downcast é descer a hierarquia de classe, como descer o

funcionário para a classe vendedor.
Upcast sobe a hierarquia. Downscast aprofunda a hierarquia.
Classe + generica = Funcionario
Classe - generica = Gerente, Vendedor, Faxineiro
Faz upcast: A a = new B (); //transforma B filha em A mãe.
A transformação upcast é implicita, não precisa expor.
Faz downcast: B a = (B) new A (); //transf. A mae em B filha.
A transformação downcast é explicita, precisa dizer pra
quem tranformar. Pode dar erro de compilação ou na execução
da aplicação, naõ é muito útil usar downcast, há um único
caso que não dá problema, no trabalho com a classe Object.

- Herança - Polimorfismo e Sobrescrita
POLIMORFISMO é mesma ação (método)se comportando diferente.
Pode usar polimorfismo com herança e usar herança sem usar
polimorfismo. Por exemplo:
Método "Pagamento processar()" = Boleto processar() +
Credito processar() + Débito processar() + PIX processar()
SOBRESCRITA é mesma ação (método) podendo se comportar
diferente. Por exemplo:
Classe Conta = atributo double saldo e método exibirSaldo()
Subclasses = Corrente + Poupança exibirSaldo() + Especial
exibirSaldo() + Universitaria.
Contas Poupança e Especial têm cálculos a mais no método
sobrescrito pra mostrar rendimento e juros em seus saldos.

Polimorfismo	Sobrescrita
Tem comportamento sempre diferente, método é muito abstrato,
não podemos prever o comportamento padrão. | Há comportamento
padrão que pode ser diferenciado em classe filha e pode ser
reaproveitado da classe mãe.

- Herança - Proposta de Exercício 2
Exercício 1: criar classes "Funcionario", "Gerente",
"Vendedor", Faxineiro", faça upcasts e downcasts.
Exercício 2: análise do comportamento de polimorfismo e
sobrescrita.

- Herança - Resolução de exercício 2 - Parte 1
Criada classe Funcionario.java.
Trabalhar com upcast facilita a aplicação do polimorfismo.

- Herança - Resolução de exercício 2 - Parte 2
Polimorsfismo ocorre quando há objetos diferentes da classe
mae, trabalha com tipo de dado da classe mae e instâncias
de classe filha.
Criado codigo da classe RodarAplicacao3.java.

- Associação - Conceito
ASSOCIAÇÃO possibilita relacionamento entre classes e obje-
tos em que podem pedir ajuda a outras classes/objetos e
representar de forma completa o conceito a que se destinam.
Tipos: estrutural ligado a métodos (composição, agregação)

e comportamental ligado a atributos (dependência).

- Associação - Tipos
Assoc-Estrutural - composição: "Com Parte Todo", ex.:
Pessoa e Endereço. Se Pessoa deixa de existir o endereço
também; a parte só existe com o todo, relação forte de
dependência da existência entre os 2 objetos. Como fazer:

```
class Pessoa {
        Endereco endereço;
}
```

Assoc-Estrutural - agregação: "Sem Parte Todo", ex.:
Disciplina e Aluno. Se Disciplina deixa de existir, o
Aluno pode permanecer vinculado a outra disciplina. Aluno
independe de Disciplina, relação forte de agregação não
de composição. Como fazer:

```
class Disciplina {
        Aluno aluno;
}
```

Agregação	Composição
Relação mais fraca entre a existência de objetos, entidades.
Dependência forte entre objetos, entidades, se um deixa de
existir o outro também.

Assoc-Comportamental - dependência: "Depende de", ex.:
Entidade Compra possui método que usa cupom para frete,
desconto ou reembolso.
[Compra] = Cliente cliente + finalizar(Cupom cupom, ...)
+ finalizar(...).
Na classe Compra, o método finalizar(Cumpom cumpom,...)
depende do objeto Cupom para executar ação. Como fazer:

```
class Compra {
        ...
        finalizar(Cupom cupom, ...) {
        ...
        }
}
```

Herança	Associação
Relação mais rígida, é definida no desenvolvimento quando
cria classe herdada de outra; não muda durante execução da
aplicação. Pergunta de uso: uma coisa é a outra? | Relação
mais flexível, os valores de métodos e atributos podem ser
mudados durante execução do software. Pergunta de uso: uma
coisa usa a outra?

- Associação - Exercício
Codifique exemplos de slides anteriores sobre associações.
A forma de materializar Composição e Agregação no código é
idêntica, o que muda é o comprtamento dos objetos.

- Interface - Conceito
INTERFACE define um contrato que deve ser seguido pela classe que a implementa, com compromisso de realizar todos os comportamentos da interface. Como fazer:

```
interface A {
        ...
}

class B implements A {
        ...
}
```

- Interface - Exercício
Criar interface "OperacaoMatematica" com 4 métodos das operações básicas: soma, subtração, multiplicação, divisao.

Continuar estudo: tipos de classe Abstrata/Concreta; metodos abstratos; caracteristicas das associações; palavras coringas super,base, super(); relações entre classes e interfaces (extends, implements).

6) A Organização de Pacotes e Visibilidades
- Pacotes - Conceitos
PACOTES são organização física ou lógica para separar classes com responsabilidades distintas.
Separações lógicas seriam separações virtuais. Existem diferentes tipos de classes: utilitárias, de integrar sistema. Como criar:

```
package ...;

import ...;
```

- Pacotes - Exercícios
Observar pacotes de um projeto para exemplificar como impactam na visibilidade. Criado projeto "pacotes".

- Visibilidades - Conceitos
VISIBILIDADE é modificador de acesso com finalidade de determinar até que ponto uma classe, atributo ou método pode ser usado.
Quando temos classes internas, trabalha-se mais com visibilidades em classes. Na maioria dos casos, trabalha-se com classes públicas. Tipos: private, protected, public.

#Private: só dentro da classe. Criar atributos privados pra encapsular, proteger dados e valores. Como fazer:
private

```
private int i;
private void do();
```

#Protected: dentro da classe, mesmo pacote e subclasses
protected

protected int i;
protected void do();

#Public: em qualquer lugar. Como fazer:
public

public int 1;
public void do();

Para possibilitar comunicação entre classes e módulos dentro
da aplicação, precisa chamar metodos publicos ou protegidos.
De acordo com a necessidade, poderão ser usadas as visibili-
dades adequadas de forma combinada.

- Visibilidades - Exercício
Exemplo de projeto para verificar uso de visibilidades.

- Conclusão
Dicas de estudo
Padrões de Projeto (Design Patterns)
Boas práticas: SOLID, código, técnicas de programação etc.
Refatoração
UML
Frameworks
Prática e estudo
[Orientação a Objetos: Aprenda seus conceitos e suas
aplicabilidades de forma
efetiva](https://www.casadocodigo.com.br/products/livro-oo-conceitos)

- Slides
https://docs.google.com/presentation/d/1k5syUGWVoY8yJ2cDLHwhrR0kDBUUG5Ay/edit?rt
pof=true&sd=true
baixei!

- GitHub
ESTRUTURA
https://github.com/tlcdio/OOAula03.1.git --> Carro.java
https://github.com/tlcdio/OOAula03.2.git --> Carro.java
https://github.com/tlcdio/OOAula03.3.git --> Carro.java
https://github.com/tlcdio/OOAula03.4.git --> Carro.java, RodarAplicacao1
RELAÇÕES
https://github.com/tlcdio/OOAula04.1.git --> Veiculo.java, Caminhao.java,
Carro2.java, Moto.java
https://github.com/tlcdio/OOAula04.2.git --> Funcionario.java, Faxineiro,
Gerente, Vendedor, Rodar Aplicacao2
https://github.com/tlcdio/OOAula04.3.git --> ClasseMae.java, ClasseFilha1,
ClasseFilha2, RodarAplicacao3
https://github.com/tlcdio/OOAula04.4.git --> Pessoa.java + Endereco,
Disciplina.java + Aluno, Compra.java + Cupom
https://github.com/tlcdio/OOAula04.5.git --> OperacaoMatematica.java,
Calculadora.java

PACOTES
https://github.com/tlcdio/OOAUla05.1.git --> outropacote (ex1, ex2),
qualqueroutropacote (ex3)
https://github.com/tlcdio/OOAula05.2.git -->