


A series of overlapping geometric shapes, including triangles and polygons, in shades of teal and purple, located in the top-left corner of the slide.

AceLeraDev Java

Módulo 2 - Programação Orientada a Objetos 1

A series of overlapping geometric shapes, including triangles and polygons, in shades of teal and purple, located in the bottom-right corner of the slide.



- Profissional com mais de 6 anos de experiência, tendo convivido com diversas linguagens (Java, .NET, Ruby). Atuou no desenvolvimento de projetos com Cognitive Services e Cloud dentre esses projetos, um premiado em Cannes. Hoje é uma entusiasta em jogos, colabora com a comunidade auxiliando, apoiando novas meninas na área

- Devs JavaGirl, MTAC.
- Coordenadora da Trilha de Modern Web durante 3 anos e atualmente na trilha Java.

Ementa geral do programa

Módulo 1: Introdução a linguagem de programação

Módulo 2: Programação Orientada a objetos I

Módulo 3: Programação Orientada a objetos II

Módulo 4: Modelagem de banco de dados relacional

Módulo 5: Manipulação de dados em banco de dados relacional

Módulo 6: Criação de APIs REST

Módulo 7: Documentação e autenticação de APIs

Módulo 8: Git, boas práticas e clean code

Módulo 9: Deploy de aplicações

Módulo 10: Preparação para o demo day

Tópicos desta aula:

- Básico da orientação objetos

Tópico 1: Array vs List<>

Tópico 2: Classes

Tópico 3: Objetos

Tópico 4: Encapsulamento


Tópico 5: Exceptions

Tópico 6: Herança



Classes **Revisão**

Além da especificação de atributos, a definição de uma classe descreve também qual o comportamento de objetos da classe, ou seja, que funcionalidades podem ser aplicadas a objetos da classe. Essas funcionalidades são descritas através de métodos.






Orientação Objetos

O paradigma de orientação a objetos foi criado com o intuito de aposentar a procedural procedural.

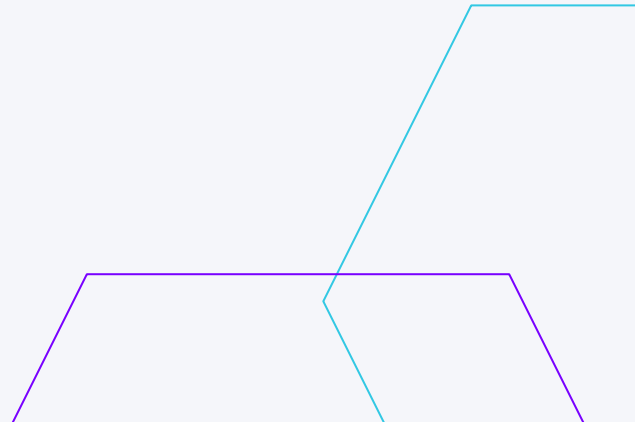
Como devo pensar no paradigma? Como centralizar padrões e isolar comportamentos e delegar responsabilidades.





SHOW THE CODE

Vamos gerar um projeto java.



Encapsulamento

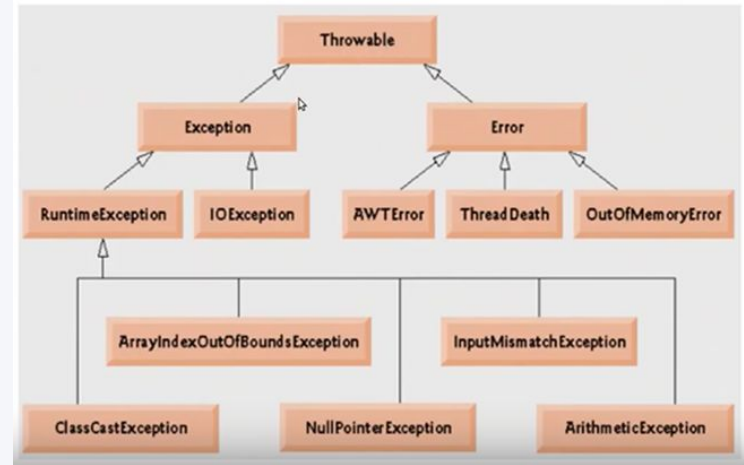
“Encapsulamento vem de encapsular, que em programação orientada a objetos significa separar o programa em partes, o mais isolado possível”

“Um mecanismo da linguagem de programação para restringir o acesso a alguns componentes dos objetos, escondendo os dados de uma classe e tornando-os disponíveis somente através de métodos...”, “... onde o estado de objetos (as variáveis da classe) e seus comportamentos (os métodos da classe) são agrupados em conjuntos segundo o seu grau de relação”

Exceptions vs Error

"...An Error "indicates serious problems that a reasonable application should not try to catch..."

"...An Exception "indicates conditions that a reasonable application might want to catch..."



Herança

Com a herança é possível criar classes derivadas, subclasses, a partir de classes bases, superclasses. As subclasses são mais especializadas do que as suas superclasses, mais genéricas. As subclasses herdam todas as características de suas superclasses, como suas variáveis e métodos. A linguagem Java permite o uso de herança simples, mas não permite a implementação de herança múltipla. Para superar essa limitação o Java faz uso de interfaces.

Polimorfismo

- Polimorfismo Estático ou Sobrecarga
- Polimorfismo Dinâmico ou Sobreposição

O Polimorfismo Estático se dá quando temos a mesma operação implementada várias vezes na mesma classe. A escolha de qual operação será chamada depende da assinatura dos métodos sobrecarregados.

O Polimorfismo Dinâmico acontece na herança, quando a subclasse sobrepõe o método original. Agora o método escolhido se dá em tempo de execução e não mais em tempo de compilação. A escolha de qual método será chamado depende do tipo do objeto que recebe a mensagem.

Refatoração com polimorfismo

```
1 class TipoEmpregado {  
2  
3     public int quantiaAPagar() {  
4         switch(lerTipo()) {  
5             case TipoDeEmpregado.ENGENHEIRO:  
6                 return _salarioMensal;  
7             case TipoDeEmpregado.VENDEDOR:  
8                 return _salarioMensal + _comissao;  
9             case TipoDeEmpregado.GERENTE:  
10                return _salarioMensal + _bonus;  
11             default:  
12                 throw new RuntimeException("tipo incorreto de empregado");  
13         }  
14     }  
15 }
```

//Essa classe que vai chamar o TipoDeEmpregado dependendo do tipo passado

```
class Empregado {
    int quantiaAPagar() {
        return _tipo.quantiaAPagar(this);
    }
}

class TipoDeEmpregado {

    abstract int quantiaAPagar(Empregado emp);
}

class Engenheiro extends TipoDeEmpregado {
    int quantiaAPagar(Empregado emp) {
        return emp.lerSalarioMensal();
    }
}

class Vendedor extends TipoDeEmpregado {
    int quantiaAPagar(Empregado emp) {
        return emp.lerSalarioMensal() + emp.lerComissao();
    }
}

class Gerente extends TipoDeEmpregado {
    int quantiaAPagar(Empregado emp) {
        return emp.lerSalarioMensal() + emp.lerBonus();
    }
}
```

The slide features decorative geometric lines in purple and teal. In the top-left corner, a purple line forms a large 'V' shape, and a teal line forms a horizontal bar. In the bottom-right corner, a teal line forms a horizontal bar, and a purple line forms a large 'V' shape.

Revisão do que vimos hoje:

Tópico 1: Classes e objetos

Tópico 2: Encapsulamento

Tópico 3: Exceptions vs Errors

Tópico 4: Herança

Tópico 5: Polimorfismo

1. Crie uma classe para representar uma pessoa, com os atributos privados de nome, data de nascimento e altura. Crie os métodos públicos necessários para sets e gets e também um método para imprimir todos dados de uma pessoa. Crie um método para calcular a idade da pessoa.
2. Crie uma classe Agenda que pode armazenar 10 pessoas e que seja capaz de realizar as seguintes operações:

```
void armazenaPessoa(String nome, int idade, float altura);  
void removePessoa(String nome);  
int buscaPessoa(String nome); // informa em que posição da agenda está a pessoa  
void imprimeAgenda(); // imprime os dados de todas as pessoas da agenda  
void imprimePessoa(int index); // imprime os dados da pessoa que está na posição "i" da agenda.
```

3. Crie uma classe denominada **Elevador** para armazenar as informações de um elevador dentro de um prédio. A classe deve armazenar o andar atual (térreo = 0), total de andares no prédio (desconsiderando o térreo), capacidade do elevador e quantas pessoas estão presentes nele. A classe deve também disponibilizar os seguintes métodos:

Inicializa : que deve receber como parâmetros a capacidade do elevador e o total de andares no prédio (os elevadores sempre começam no térreo e vazio);

Entra : para acrescentar uma pessoa no elevador (só deve acrescentar se ainda houver espaço);

Sai : para remover uma pessoa do elevador (só deve remover se houver alguém dentro dele);

Sobe : para subir um andar (não deve subir se já estiver no último andar);

Desce : para descer um andar (não deve descer se já estiver no térreo);

Encapsular todos os atributos da classe (criar os métodos set e get).

Feedback da aula

/

