Desenvolvimento básico em Java

> Curso

Preparando o ambiente para programar em Java:

> Módulo

Java - Instalação e Ambiente:

> Aula

Programas Instalados:

- Java JDK e JRE;
- Maven;
- Graddle;
- Intellij IDE;

Java - Criação de Projetos:

> Aula

- Aprendemos a estrutura de um projeto Java, e a interface da IDE Intellij;

O que precisamos saber sobre Java

> Módulo/Aula (Módulo de aula única)

O que é Java:

- É uma linguagem criada em **1995** pela **Sun Microsystems**, que anos depois foi adquirida pela **Oracle**.

Compilação do Java:

- Outras linguagens geralmente são **compiladas** para **código nativo**.
- Com o Java, o código é compilado para um **bytecode**, que por sua vez, é interpretado por uma **máquina virtual**.

O que é o compilador:

- É um programa que transforma o **código fonte**, para **código objeto**.
- Traduz o programa de uma **linguagem textual para linguagem de máquina**.

O que é o bytecode:

- É o **resultado da compilação** Java.
- Ele **é interpretado** e executado pela **Máquina Virtual Java, JVM**.

O que é a JVM:

- Primeiramente, **VM ou Virtual Machine**, é um **software que simula uma máquina física**, executa programas, gerencia processos, memória e arquivos.
- A **JVM ou Java Virtual Machine** é a Maquina Virtual que **interpreta bytecode** compilado a partir de código Java.

O que é a JRE:

- **Java Runtime Environment**, Ambiente de Execução do Java.
- Composta pela JVM, bibliotecas e APIs etc.
- Parte **responsável pela execução do software** Java.

O que é JDK:

- **Kit de Desenvolvimento Java.**
- Inclui o compilador, as bibliotecas e a JRE.

O que é Java SE ou JSE:

- Java Standard Edition.
- **Distribuição mínima** da plataforma de desenvolvimento.

O que é Java EE ou JEE:

- Extensão do Java EE.
- Java Enterprise Edition.
- Inclui mais **features, para uso profissional**.

O que é Jakarta EE:

- A **Oracle** não estava dando a devida atenção para o constante desenvolvimento do Java;
- Então, ela **doou** todo o **código do Java EE** para a **Eclipse Foundation**.
- A **nova marca** foi registrada como **Jakarta EE**.

Características da linguagem

> Módulo

Iniciando um Projeto Java:

> Aula

Assuntos:

- Classes;
- Tipos
- Modificadores de acesso
- Interfaces
- Enums

Classes:

- Todo programa Java roda em cima duma classe.
- O nome da classe deve ser o mesmo do arquivo.java
- O nome da clase deve sempre ser em TitleCase
- As classes são contém:
    - Método Main;
    - Construtores;
    - Argumentos;
    - Parâmetros.

Tipos primitivos, wrappers, não primitivos e tipagem forte e estática:

> Aula

Tipos:

- **Primitivos:** não aceitam null, possuem valores default
    - Byte - 0;
    - Char - '\u0000';
    - Short - 0;
    - Int - 0;
    - Long - 0L.
    - Float - 0.0f;
    - Double - 0.0d;
    - Boolean - false.
- Wrappers: Objetos que rpresentam tipos primitivos no Java.
    - AutoBoxing
    - UnBoxing
- Não primitivos;
    - String;
    - Number;
    - Objects;
- Tipagem forte e estática.
    - var - Inferência de tipo;
    - Tipagem de variáveis verificadas na compilação

Modificadores de Acesso:

> Aula

- Responsáveis por habilitar ou não o acesso de outros programas á classe.

Public:

- Pode ser acessado por qualquer classe.

Private:

- Só pode ser acessado pela classe de origem.

Protected:

- Pode ser acessível por classes do mesmo pacote ou herdado.

Default:

- Pode ser acessível por classes do mesmo pacote.

Abstract:

- Não pode ser aplicado à variáveis, apenas em classes e métodos.
- Não pode ser instanciada;
- Caso dentro duma classe, houver um método abstrato, a classe também deve ser declarada como abstract.

Static:

- Usada para criação de variáveis;
- Pode ser instanciada;

Final:

- Não permite estender nos métodos,
- Não permite mudar o valor de variáveis.

Métodos abstratos, default e herança múltipla

> Aula

Interfaces:

- Métodos abstratos;
    - Devem ser implementados por todos;
    - Novos métodos quebram as implementações;
- Métodos default;
    - São herdados a todosque implementam;
    - Novos métodos não quebram as implementações;
- Herança múltipla.

Enums:

- Dicionários de dados imutável;
- Não é permitido novas instâncias;
- O construtor é sempre declarado como private;

Características da Linguagem II

> Módulo

Strings e o pacote java.lang

> Aula

Strings:

É uma classe que representa uma sequência de caracteres.

Operações com String:

- **CharAt()** - retorna a caractere na posição passada como parâmetro;
- **length()** - retorna o tamaho da string;
- **trim()** - remove os caracteres em branco nas extremidades.
- **toLowerCase() / toUpperCase** - deixa todos os caracteres em maiúsculo ou minúsculo.
- **contains()** - retorna um booleano dada uma string.
- **replace()** - trocar caracteres.
- **equals()** - compara a string original com a string passada como parâmetro. Retorna um valor booleano.
- **equalsIgnoreCase()** - o mesmo que equal ignorando diferença entre maiúsculo e minúsculo.
- **substring()** - selecionar uma pate da string.

```
var str = "Motorola da Xiaomi"
var str2 = "   aa   "

str.CharAt(2) // 't'
str.length() // 18
str2.trim() // "aa"

str.toLowerCase() // "motorola da xiaomi"
str.toUpperCase() // "MOTOROLA DA XIAOMI"

str.contains("M") // true
str.contains("y") // false

str.replace(charSequence: "o", charSequence: "c") // "Mctcrcla da Xiacmi"

str.equals("Motorola da Xiaomi") //true
str.equals("banana") // false
str.equals(str.toLowerCase()) // false

str.equalsIgnoreCase("Motorola da Xiaomi") // true
str.equalsIgnoreCase("banana") //false
str.equalsIgnoreCase(str.toLowerCase()) //true

str.substring(1, 4) // "oto"
```

Introdução a condicionais:

> Aula

- Operadores;
- If/Else;
- Definição de variavel condicionalmente;

Laços de repetição:

> Aula

- For;
- While.

Convenções de nomes:

> Aula

- **Classes:** Kamiel Case;
- **Métodos:** lower case ou, se composto: nomeComposto;
- **Variaveis:** lower case, geralmente sem números e caracteres especiais;

Code Style - Plugins:

- Check Style;
- PMD - Check conduct;

Debug de código

> Módulo/Aula

- Como usar o IteliJ para o debug

Orientação a Objetos com Java

> Módulo

Paradigma em orientação a objetos:

> Aula

Classe:

- Um modelo/molde usado para construir algo;

```
public class Pessoa {

    private String nome = "Marco";

    public String getNome() {
        return nome;
    }
}
```

Objeto:

- O algo de fato que foi construído a partir do molde.

```
public class ExemploPessoa {

    public static void main(String[] args) {
        Pessoa minhaPessoa = new Pessoa();

        minhaPessoa.getNome(); // "Marco"
    }
}
```

Métodos:

- Comportamento e ações do objeto.

```
public class Pessoa {

    private String nome = "Marco";

    public String getNome() {
        return nome;
    }

    public String falarProprioNome() {
        System.out.println("Meu nome é" + getNome())
    }
}
```

```
public class ExemploPessoa {

    public static void main(String[] args) {
        Pessoa minhaPessoa = new Pessoa();

        minhaPessoa.getNome();
        // retorna: "Marco"

        minhaPessoa.falarProprioNome()
        // printa: "Meu nome é Marco"
    }
}
```

Criando objetos com construtores:

> Aula

- Constroem objetos com base em uma classe.

Encapsulamento, herança e polimorfismo:

> Aula

Encapsulamento:

- Proteger informações de alguma forma.

Herança:

- A capacidade de uma Classe herdar o comportamento de outra.

```
public class Veiculo {
    // Serve para quase qualquer veículo
    private String modelo;
    private String marca;

    private int qtdPortas; // Serve só para carros
    private String cilindradas; // Serve só para motos
}
```

```
public class Veiculo {
    // Serve para quase qualquer veículo
    private String modelo;
    private String marca;
}

public class Carro extends Veiculo {
    private int qtdPortas;
}

public class Moto extends Veiculo {
    private String cilindradas;
}
```

Polimorfismo:

- Os métodos são diferenciados em diferentes herdeiros.

```
public class Veiculo {
    // Serve para quase qualquer veículo
    private String modelo;
    private String marca;
    private double valor;

    // O IPVA para veículos genéricos é no min. 0.01
    public double calcIPVA() {
        return valor * 0.01;
    }
}

public class Carro extends Veiculo {
    private int qtdPortas;

    // já para carros é no min. 0.07
    public double calcIPVA() {
        return valor * 0.07;
    }
}
```

```
public class Moto extends Veiculo {
    private String cilindradas;

    // e para motos 0.03
    public double calcIPVA() {
        return valor * 0.03;
    }
}
```

Características específicas em OO:

> Aula

- This, Super, Equals, HashCode

This:

- Auto referência ao escopo do objeto.

Super:

- Auto referencia à classe em que uma subclasse foi herdada.

Equals:

- Serve para fazer uma comparação entre objetos.

HashCode:

- Um código gerado que garante um caráter único a cada objeto.

Aprenda S.O.L.I.D. com Java

> Módulo

Introdução à aula:

> Aula

- SOLID é um acrônimo dos princípios da POO descritos por Robert C. Martin ("Uncle Bob")
- Feito para deixar códigos mais limpos, refatoraveis e reaproveitaveis.

S.O.L.I.D.:

- Single Responsability Principle
- Open Closed Principle
- Liskov Substitution Principle
- Interface Segregation Principle
- Dependency Inversion Principle

Conceito: Single Responsability Principle:

> Aula

- > "A class should have one, and only one, reason to change"
- Não fazer classes com várias responsabilidades
- Cada classe deve ter uma única responsabilidade

Conceito: Open Closed Principle:

> Aula

- > "You should be able to extend a classes behavior, without modifying it"
- Objetos devem estar **abertas para extensão**, mas **fechadas para modificação**.
- Quando novos comportamentos precisam ser adicionados no software, **devemos estender e não alterar o código fonte original**.

Conceito: Liskov Substitution Principle

> Aula

- > "Derrived classes must be substitutable for their base classes"
- Esse princípio foi introduzido por Barbara Liskov em 1987
- Por exemplo, apesar de um Quadrado() derivar da classe Retangulo(), métodos como setAltura()/setLargura() separadamente não se aplicam à um quadrado, uma vez que seus lados têm sempre a mesma medida. Esse tipo de abstração pode causar erros ou comportamentos inesperados.

Conceito: Inface Segregation Principle

> Aula

- > "Make fine grained interfaces that are client specific."
- Uma classe **não deve** ser forcada a implementar interfaces e **métodos** que **não serão utilizados**.
- É melhor criar **interfaces** mais **específicas** ao invés de termos uma única **interface genérica**.

Conceito: Dependency Inversion Principle

- > "Depend on abstractions,not on concretions"
  >
  > "Dependa de abstrações e não de implementações"
- Um módulo de alto nível não deve depender de módulos de baixo nível, ambos devem depender da abstração.

Trabalhando com Datas

> Módulo

Class Date:

> Aula

- A implementação do java.etil.Date foi desde a versão 1.0 da linguagem.

Construtores:

- **Date()** - momento atual (na verdade o milisegundo mais próximo do momento de execução)
- **Date(long date)** - espera que vc diga qual é a data que está sendo instanciada. Necessita como parâmetro **milissegundos com base no padrão epoch**, que usa como referência **1 de janeiro de 1970 00:00:00**

Métodos úteis:

- **after/before()** - compara duas datas e retorna se é anterior ou posterior;;
- **compareToDate()** - retorna um int com a -1, 0 ou 1;
- **equals()** - compara duas datas e retorna boolean;
- **getTime()** - rebece um objeto Date e retorna um Long com milisegundos;
- **setTime()** - oposto;
- **from()** - recebe um instante e retorna uma data estática;
- **toInstant()** - oposto;

Classe Calendar:

> Aula

- Implementada na JDK 1.1 para facilitar algusn recursos que a class Date não fornecia.
- Alguns construtores do Date foram depreciados;

Classe DateFormat:

> Aula

- Existem duas classe DateFormat e SimpleDateFormat;

Conhecendo date a partir do Java 8:

> Aula

- A grande melhoria é no pacote java.time, herdada do projeto Joda Time.
- As classes destaques é **LocalDate**, **LocalTime** e **LocalDateTime**.

LocalDate:

- É uma classe imutavel, só trabalha com datas.
- Padrão é yyyy-MM-dd.

LocalTime:

- Imutavel.
- Só trabalha com horas;
- Padrão é 12:22:10:12323712.

LocalDateTime:

- Funciona como uma junção dos outros dois;
- Imutavel;

Trabalhando com Arrays

> Módulo/Aula

- Estrutura de dados que nos permite organizar valores na memória;
- Fazem parte da biblioteca java.util que é nativa do Java;
- Armazena elementos do mesmo tipo;
- Podem ser uni ou multidimensionais;

Tratamento do exceções:

> Módulo

Aprenda o funcionamento de Exeções em Java:

> Aula

Exceptions:

- São todos os erros que ocorrem durante o processamento de um fluxo ou método.
- Podem ser esperadas ou não.
- Essas falhas não devem ocorrer rotineiramento no fluxo ne um sistema.

Exemplo de tratamento:

```
try {
    new java.io.FileInputStream("arquivo.txt);
} catch (java.io.FileNotFoundExeption e) {
    System.out.print("Não deu :/")
}
```

Conheça o finally e throw:

> Aula

Finally:

- É colocado abaixo do catch.
- Sempre é executado antes do fim do método.

- Normalmente usado para liberar memória ou finalizar conexões.

Throw e Throws:

- Throw: Lançar manualmente um tipo de exceção, juntamente com a mensagem de erro, para o método que fez a chamada;
- Thows: assinatura do método que será retornado caso ocorra erro para o método que fez a chamada, dentro de um fluxo encadeado.

Certificado \P/

---

https://certificates.digitalinnovation.one/EF8FDFEB