FIAP - TDS - Prof. Tiago Ferrer

Domain Driven Design

Semana 2 - 04/03/24 : 08/03/2024

Objetivos

Nesta semana, embarcaremos em uma jornada de imersão na linguagem Java utilizando uma IDE, revisitando e aprofundando seus conhecimentos em suas principais características e sintaxe. Exploraremos em detalhes:

- Tipagens primitivas e não primitivas: Dominaremos os tipos de dados básicos (int, double, boolean etc.) e os tipos de referência (String, arrays etc.), compreendendo suas diferenças e aplicações.
- Operadores: Desvendaremos os operadores aritméticos, relacionais, lógicos, de atribuição, de incremento e decremento e o operador ternario condicional, aprimorando sua capacidade de escrever expressões complexas e precisas.
- Operações de controle: Aperfeiçoaremos o uso de estruturas condicionais e de repetição para criar programas mais dinâmicos e eficientes.

IDE - Integrated Development Environment

O que é uma IDE?

IDE significa **Ambiente de Desenvolvimento Integrado** (em inglês, Integrated Development Environment). É um software utilizado para criar aplicações que combina várias ferramentas de desenvolvimento em uma única interface gráfica, facilitando o trabalho dos programadores [1].

As principais ferramentas que uma IDE pode oferecer incluem [1]:

- Editor de código: para escrever e editar código-fonte.
- Compilador: para converter o código-fonte em linguagem de máquina.
- Debugger (Depurador): para encontrar e corrigir erros no código.
- Assistentes de código: para gerar automaticamente partes do código.

Quando e porque surgiram?

As IDEs surgiram na década de 1980 como uma resposta à necessidade de tornar o desenvolvimento de software mais eficiente e produtivo [2,3]. Naquela época, os programadores precisavam usar várias ferramentas diferentes para escrever, compilar, depurar e executar seus programas, o que tornava o processo demorado e propenso a erros.

As IDEs integram diversas ferramentas essenciais para o desenvolvimento de software em um único ambiente, permitindo com que os programadores se concentrem no que realmente é importante - *escrever código*. Essa característica fez com que as IDEs se tornassem populares rapidamente entre os programadores e, hoje em dia, são consideradas uma ferramenta essencial para o desenvolvimento de software.

Algumas das IDEs mais populares incluem:

- Visual Studio Code (Microsoft)
- IntelliJ IDEA (JetBrains)
- Eclipse (Fundação Eclipse)
- Xcode (Apple)
- NetBeans (Apache Software Foundation)

As principais vantagens de usar uma IDE incluem:

- Maior produtividade: as IDEs automatizam muitas tarefas repetitivas, liberando tempo para que os programadores se concentrem nas regras de negócio dos seus sistemas.
- Menor número de erros: as IDEs fornecem vários recursos para ajudar os programadores a encontrar e corrigir erros em seu código antes mesmo de compliar os códigos.
- Código mais legível: as IDEs podem formatar automaticamente o código, facilitando a difusão e aderência de padrões de formatação nos projetos.
- Melhor colaboração: as IDEs podem facilitar a colaboração entre programadores, fornecendo ferramentas para compartilhamento de código e controle de versão.
- Artificial Intelligence Assistant: as IDEs tem incorporado soluções integradas com IAs generativas permitindo geração automatica de código, documentação, commit messages e muito mais.

As desvantagens de usar uma IDE incluem:

- Curva de aprendizado: as IDEs podem ser complexas e com funcionalidades distintas, o que pode exigir algum tempo para aprender a usá-las de forma eficaz.
- Tamanho: as IDEs podem ser grandes e ocupar muito espaço em disco.
- Custo: algumas IDEs podem ser caras, especialmente para uso comercial.

Em resumo, as IDEs são ferramentas poderosas que podem ajudar os programadores a serem mais produtivos e eficientes. No entanto, é importante escolher uma IDE que seja adequada às suas necessidades e habilidades.

Curiosidades

A IDE mais utilizada no mundo para desenvolvimento Java é o **IntelliJ IDEA**. De acordo com a pesquisa anual do Stack Overflow Developer Survey, o IntelliJ IDEA é a IDE preferida por mais de 60% dos desenvolvedores Java [4].

Existem várias razões para o IntelliJ IDEA ter ganho tanto espaço no mercado:

- Recursos avançados: O IntelliJ IDEA oferece uma ampla gama de recursos avançados para desenvolvimento Java, como:
    - Autocompletar inteligente
    - Detecção de erros em tempo real
    - Refatoração eficiente de código
    - Suporte a frameworks populares como Spring Boot e Hibernate
    - Integração com ferramentas de teste
    - Integração com diversas soluções tecnologicas através de plugins
- Interface amigável: O IntelliJ IDEA possui uma interface amigável e personalizável, o que facilita o aprendizado e o uso da ferramenta.
- Disponibilidade em várias plataformas: O IntelliJ IDEA está disponível para Windows, macOS e Linux.
- Versão gratuita: O IntelliJ IDEA possui uma versão gratuita que oferece a maioria dos recursos básicos da ferramenta.

Algumas dicas para escolher a melhor IDE para você:

- Considere suas necessidades: Quais recursos você precisa em uma IDE?
- Experimente diferentes IDEs: A melhor maneira de encontrar a IDE ideal é experimentar diferentes opções.
- Leia avaliações e compare recursos: Pesquise online para ver o que outros desenvolvedores pensam sobre diferentes IDEs.

Introdução à linguagem Java

Java é uma linguagem de programação de uso geral, orientada a objetos, robusta, segura e portátil. Ela foi lançada em 1995 pela Sun Microsystems (atualmente parte da Oracle) e rapidamente se tornou uma das linguagens de programação mais populares do mundo [5].

Java é usada para desenvolver:

- Aplicações web (servlets, JSPs, frameworks como Spring Boot)
- Aplicações desktop (com interfaces gráficas - *GUI*)
- Aplicações móveis (Android)
- Aplicações em nuvem
- Jogos
- E muito mais!

Principais características da linguagem Java

- Orientada a objetos: Java é uma linguagem orientada a objetos, o que significa que os programas são escritos em termos de objetos que possuem propriedades e métodos. Ainda, oferece mecanismos de abstração, encapsulamento e hereditariedade [5]. Essas características tornam o código mais modular, reutilizável e de fácil manutenção.
- Robusta: Java foi projetada para ser robusta e confiável. O código Java é verificado em tempo de compilação e em tempo de execução para garantir que não haja erros.
- Segura: Java é uma linguagem de programação segura. Ela possui recursos que ajudam a prevenir ataques de segurança e proteger dados confidenciais.

- Portátil: Java é uma linguagem de programação portátil. O código Java pode ser executado em qualquer plataforma que tenha uma máquina virtual Java (JVM) instalada [5].
- Ampla comunidade: Java possui uma comunidade de usuários e desenvolvedores muito ativa. Isso significa que há uma grande quantidade de recursos disponíveis para aprender Java e obter ajuda com problemas de desenvolvimento [4].

Outras características importantes da linguagem Java incluem:

- Gerenciamento de memória automático / Sem ponteiros: Java possui um coletor de lixo que gerencia a memória automaticamente. Isso significa que os programadores não precisam se preocupar com a alocação e desalocação de memória [5].
- Suporte a multitarefa (*multithreaded*): Java permite que os programas executem multiplas tarefas concorrentes, ou seja, ao mesmo tempo.
- Performance: Java possui tecnologia *Hotspot* e o compilador *Just In Time* (JIT) que permitem, respectivamente, a identificação de trechos do código com alto uso e a sua compilação automática para linguagem nativa da máquina, promovendo aumento de performance significativo [5].
- Ampla biblioteca de classes: Java possui uma biblioteca de classes extensa que fornece funcionalidades para diversas tarefas de desenvolvimento.

Características gerais da sintaxe Java

A sintaxe Java é derivada da linguagem C e C++, mas com algumas diferenças importantes que a tornam mais simples e segura.

Algumas das principais características da sintaxe Java incluem:

- Orientação a objetos: A sintaxe Java reflete a natureza orientada a objetos da linguagem. Por exemplo, classes são definidas usando a palavra-chave `class` e objetos são instanciados usando a palavra-chave `new`.
- Fortemente tipada: Java é uma linguagem de programação fortemente tipada, o que significa que cada variável deve ter um tipo especificado. Isso ajuda a evitar erros de tipo em tempo de execução.
- *Case-sensitive*: A sintaxe Java é case-sensitive, o que significa que as palavras-chave e identificadores distinguem maiúsculas e minúsculas.
- Simples e concisa: A sintaxe Java é considerada simples e concisa em comparação com outras linguagens de programação, como C++.

Aqui estão alguns exemplos de código Java que ilustram algumas das características da sintaxe:

```java
// Classe
public class Pessoa {

    // Atributos
    private String nome;
    private int idade;

    // Construtor
    public Pessoa(String nome, int idade) {
        this.nome = nome;
        this.idade = idade;
    }

    // Métodos
    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public int getIdade() {
        return idade;
    }

    public void setIdade(int idade) {
        this.idade = idade;
    }

}

// Main
public class Main {

    public static void main(String[] args) {
        Pessoa pessoa = new Pessoa("João", 25);

        System.out.println("Nome: " + pessoa.getNome());
        System.out.println("Idade: " + pessoa.getIdade());

        pessoa.setNome("Maria");
        pessoa.setIdade(30);

        System.out.println("Nome: " + pessoa.getNome());
        System.out.println("Idade: " + pessoa.getIdade());
    }

}
```

Neste exemplo, podemos observar:

- A classe Pessoa é definida com dois atributos (`nome` e `idade`) e dois métodos (`getNome()` e `setIdade()`).
- A palavra-chave `public` é usada para indicar que a classe e os métodos são acessíveis de outras classes.

- A palavra-chave `private` é usada para indicar que os atributos são acessíveis apenas dentro da classe.
- A palavra-chave `new` é usada para instanciar um novo objeto da classe `Pessoa`.
- A palavra-chave `this` é usada para referenciar o objeto atual dentro da classe.

Tipagem

Tipos primitivos

Em Java, existem 8 tipos primitivos, que podem ser divididos em dois grupos:

Tipos numéricos:

- Inteiros:

Tipo	Bytes/bits	Faixa de valores
byte	1/8	de -128 a +127
short	2/16	de -32.768 a 32.767
int	4/32	de -2.147.483.648 a +2.147.483.647
long	8/64	de -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807

- Ponto flutuante:

Tipo	Bytes/bits	Faixa de valores
float	4/32	Valores positivos: de +140129846432481707e-45 a +3.40282346638528860e+38 / Valores negativos: de -3.4028234663852886e+38 a -140129846432481707e-45
double	8/64	Valores positivos: de +4940656458412465 44e-324 a +1.79769313486231570e+308 / Valores negativos: de -1.7976931348623157e+308 a -4.94065645841246544e-324

Tipo lógico:

Tipo	Bytes/bits	Faixa de valores
boolean	1/8	false ou true

Tipo caractere:

Tipo	Bytes/bits	Faixa de valores
char	2/16	de \u0000 a \uFFFFF

Características dos tipos primitivos:

- Armazenam valores simples: Os tipos primitivos armazenam valores simples como números, caracteres e valores booleanos.

- Imutáveis: Os valores dos tipos primitivos não podem ser alterados após a criação.
- Passados por valor: Quando um tipo primitivo é passado como argumento para um método, o valor é copiado.

Exemplos de uso dos tipos primitivos:

- `int idade = 25;` // Declara uma variável idade do tipo int e atribui o valor 25
- `long dias = 31;` ou `long dias = 31L;` // Declara uma variável dias do tipo long e atribui o valor 31
- `double preco = 12.50;` ou `double preco = 12.50D;` // Declara uma variável preco do tipo double e atribui o valor 12.50
- `boolean isAtivo = true;` // Declara uma variável isAtivo do tipo boolean e atribui o valor true

Tipos de referência (não primitivos)

Em Java, os tipos não primitivos, também conhecidos como tipos de referência, são usados para representar objetos. Eles são diferentes dos tipos primitivos, que armazenam valores simples como números, caracteres e valores booleanos.

Características dos tipos não primitivos:

- Armazenam referências: Os tipos não primitivos armazenam referências para objetos na memória. Isso significa que eles não armazenam os valores dos objetos diretamente, mas sim um endereço de memória onde o objeto está armazenado.
- Mutáveis: Os valores dos tipos não primitivos podem ser alterados após a criação. Isso porque os objetos que eles referenciam podem ser modificados.
- Passados por referência: Quando um tipo não primitivo é passado como argumento para um método, a referência para o objeto é copiada. Isso significa que se o objeto for modificado dentro do método, a mudança será refletida na variável original.

Exemplos de tipos não primitivos:

- `String` : Representa uma sequência de caracteres.
- `Integer` : Representa um número inteiro.
- `Double` : Representa um número de ponto flutuante.
- `Long` : Representa um número inteiro.
- `Date` : Representa uma data.
- *Representam qualquer classe que a gente crie e/ou use!*

Qual a diferença entre int e Integer / long e Long?

Como podemos observar acima podemos tipar números como *tipos primitivos* e *tipos não primitivos*. A diferença está no fato de que os **tipos não primitivos** possuem métodos, como todas as classes, e ainda podem ter valor atribuído nulo. Essas características não são encontradas nos tipos primitivos, dessa maneira ao nao especificar um valor para um tipo `int` ou `long` o valor retornado será o valor padrão zero.

Operadores

Os operadores em Java são símbolos especiais que permitem realizar operações em valores, variáveis e expressões. Existem diversos tipos de operadores, cada um com uma função específica.

Operadores Aritméticos:

- `+` : Adição
- `-` : Subtração
- `*` : Multiplicação
- `/` : Divisão
- `%` : Módulo (resto da divisão)

Operadores Relacionais:

- `==` : Igualdade
- `!=` : Desigualdade
- `<` : Menor que
- `<=` : Menor ou igual que
- `>` : Maior que
- `>=` : Maior ou igual que

Operadores Lógicos:

- `&&` : E lógico
- `||` : Ou lógico
- `!` : Negação lógica

Operadores de Atribuição:

- `=` : Atribuição simples
- `+=` : Soma e atribui
- `-=` : Subtrai e atribui
- `*=` : Multiplica e atribui
- `/=` : Divide e atribui
- `%=` : Módulo e atribui

Operadores de Incremento e Decremento:

- `++` : Incrementa em 1
- `--` : Decrementa em 1

Operador Ternário:

```
condição ? valor_se_verdadeiro : valor_se_falso
```

Exemplos de uso de operadores:

```java
// Soma
int soma = 10 + 5;

// Comparação
boolean igual = 10 == 5;

// Negação lógica
boolean diferente = !(10 == 5);

// Atribuição
int numero = 10;

// Incremento
numero++;

// Operador ternário
String resultado = numero > 10 ? "Maior que 10" : "Menor ou igual a 10";
```

Operações de Controle em Java

As operações de controle em Java permitem que você defina o fluxo de execução de um programa. Elas determinam quais partes do código serão executadas e em qual ordem.

Existem diversos tipos de operações de controle em Java:

- Estruturas condicionais:
    - `if` : Executa um bloco de código se uma condição for verdadeira.
    - `else` : Executa um bloco de código se uma condição for falsa.
    - `else if` : Permite verificar várias condições em sequência.
    - `switch` : Executa um bloco de código específico com base no valor de uma variável.
- Estruturas de repetição:
    - `for` : Executa um bloco de código um número determinado de vezes.
    - `while` : Executa um bloco de código enquanto uma condição for verdadeira.
    - `do-while` : Executa um bloco de código pelo menos uma vez e depois repete enquanto uma condição for verdadeira.
- Instruções de desvio:
    - `break` : Sai de um loop ou bloco de código.
    - `continue` : Pula para a próxima iteração de um loop.

Exemplos de uso de operações de controle:

```java
// Estruturas condicionais (if)

int idade = 17;

if (idade >= 18) {
    System.out.println("Maior de idade");
} else {
    System.out.println("Menor de idade");
}
```

```java
// Estruturas de repetição (for)

for (int i = 0; i < 10; i++) {
    System.out.println(i);
}
```

```java
// Estruturas de repetição (while)

int numero = 1;

while (numero <= 10) {
    if (numero % 2 == 0) {
        // Pula para a próxima iteração do loop se o número for par
        continue;
    }

    System.out.println("Número ímpar: " + numero);

    // Sai do loop se o número for 5
    if (numero == 5) {
        break;
    }

    numero++;
}
```

```java
// Estruturas de repetição (do-while)

int idade = 0;

do {
     System.out.println("Menor de idade com " + idade + " anos");
     idade ++;
     } while (numero < 18);
```

```java
// Instrução de desvio (switch)

int dia = 5;

switch (dia) {
            case 5:
                System.out.println("Dia de receber salário");
                break;
            case 10:
                System.out.println("Dia de pagar a luz");
                break;
            case 15:
                System.out.println("Dia de pagar a internet");
                break;
            default:
                System.out.println("Dia sem obrigação financeiras");
        }
```

Lendo inputs do console

A biblioteca `java.util.Scanner` em Java fornece uma maneira simples e eficiente de ler dados do console. Ela oferece métodos para ler diferentes tipos de dados, como strings, números inteiros e números de ponto flutuante.

Como usar a biblioteca Scanner:

1. Importar a biblioteca:

```java
import java.util.Scanner;
```

2. Criar um objeto Scanner:

```java
Scanner scanner = new Scanner(System.in);
```

3. Ler dados usando métodos específicos:

- `scanner.next()` : Lê a próxima string
- `scanner.nextInt()` : Lê o próximo número inteiro
- `scanner.nextDouble()` : Lê o próximo número de ponto flutuante

4. Fechar o objeto Scanner:

```java
scanner.close();
```

Exemplo

```
Scanner scanner = new Scanner(System.in);

System.out.println("Digite seu nome: ");
String nome = scanner.next();

System.out.println("Digite sua idade: ");
int idade = scanner.nextInt();

System.out.println("Olá, " + nome + "! Você tem " + idade + " anos de
idade.");

scanner.close();
```

Exercícios

1. **Calculadora básica:** Crie um programa que realiza as operações básicas de adição, subtração, multiplicação e divisão. O programa deve solicitar ao usuário dois números e a operação desejada, e imprimir o resultado da operação.

2. **Conversão de unidades:** Crie um programa que converte unidades de comprimento. O programa deve solicitar ao usuário o valor e a unidade de origem, e imprimir o valor convertido na unidade de destino. Unidades que devem estar disponíveis: kilometro (km), metro (m) e centimetro (cm).

Referências

1. RED HAT. O que é um IDE?

   [S.l.], [s.d.]. Disponível em: https://www.redhat.com/pt-br/topics/middleware/what-is-ide (https://www.redhat.com/pt-br/topics/middleware/what-is-ide). Acesso em: 02 mar. 2024.

2. ALURA. O que é uma IDE?

   [S.l.], [s.d.]. Disponível em: https://www.alura.com.br/artigos/o-que-e-uma-ide (https://www.alura.com.br/artigos/o-que-e-uma-ide). Acesso em: 02 mar. 2024.

3. EMERY, Gabriel D. Drops Java: Escolha de uma IDE.

   LinkedIn, 2023. Disponível em: https://www.linkedin.com/pulse/drops-java-escolha-de-uma-ide-gabriel-d-emery/?originalSubdomain=pt (https://www.linkedin.com/pulse/drops-java-escolha-de-uma-ide-gabriel-d-emery/?originalSubdomain=pt). Acesso em: 02 mar. 2024.

4. STACK OVERFLOW. Stack Overflow Developer Survey 2023.

   [S.l.], 2023. Disponível em: https://survey.stackoverflow.co/2023/ (https://survey.stackoverflow.co/2023/). Acesso em: 02 mar. 2024.

5. JANDLR JR., Peter. Java Guia do Programador, 4. ed. São Paulo: Novatec Editora, 2021.