

IDE IntelliJ e Programação Estruturada vs Orientada a Objetos

Ferramentas de desenvolvimento e paradigmas de programação em Java

1. Introdução

Antes de escrever qualquer linha de código, um desenvolvedor Java precisa de um ambiente adequado para editar, compilar, executar e depurar seus programas. O **IntelliJ IDEA** é uma das IDEs (*Integrated Development Environment*) mais populares para esse fim. Além de apresentar a ferramenta, este texto compara dois paradigmas de programação que podem ser usados dentro dela: a **Programação Estruturada** e a **Programação Orientada a Objetos (POO)**, com exemplos equivalentes em Java para evidenciar as diferenças entre eles.

2. A IDE IntelliJ IDEA

O **IntelliJ IDEA**, desenvolvido pela JetBrains, é uma IDE voltada principalmente para Java, embora também suporte outras linguagens como Kotlin, Python e JavaScript por meio de plugins. Está disponível em uma versão gratuita (*Community*) e em uma versão paga (*Ultimate*), com recursos adicionais para desenvolvimento web e empresarial.

2.1. Principais Recursos

- **Autocompletar inteligente:** sugere métodos, variáveis e classes enquanto o código é digitado, reduzindo erros de digitação.
- **Depurador (debugger) integrado:** permite definir pontos de parada (*breakpoints*) e inspecionar variáveis linha a linha durante a execução.
- **Refatoração automática:** renomear variáveis, extrair métodos e reorganizar código com segurança em todo o projeto.
- **Integração com ferramentas de build:** suporte nativo a Maven e Gradle para gerenciar dependências e automatizar a compilação.
- **Controle de versão integrado:** suporte a Git diretamente na interface, sem precisar de linha de comando.
- **Geração automática de código:** criação rápida de construtores, getters, setters e sobrescrita de métodos como `equals()` e `toString()`.

2.2. Criando um Projeto Java no IntelliJ

De forma resumida, os passos básicos para iniciar um novo projeto são: (1) selecionar *New Project*; (2) escolher a linguagem Java e o JDK instalado; (3) definir o gerenciador de build (Maven, Gradle ou nenhum); (4) nomear o projeto e escolher o local de armazenamento; e (5) criar a primeira classe com um método `main`, ponto de entrada de todo programa Java.

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("Projeto criado no IntelliJ IDEA!");  
    }  
}
```

Ao clicar no botão de execução (ícone verde de "play") ao lado do método main, o IntelliJ compila e executa automaticamente o programa, exibindo a saída em um painel de console integrado.

3. Programação Estruturada

A **Programação Estruturada** é um paradigma baseado na ideia de que qualquer programa pode ser construído a partir de três estruturas básicas: **sequência** (instruções executadas em ordem), **decisão** (if/else, switch) e **repetição** (for, while). O código é organizado em **funções ou procedimentos** que operam sobre dados, mas, diferentemente da POO, os dados e as funções que os manipulam ficam **separados** — não existe a ideia de "objeto" combinando estado e comportamento.

Embora Java seja uma linguagem orientada a objetos, é possível escrever código em um **estilo estruturado**, concentrando toda a lógica em métodos static dentro de uma única classe, que funciona apenas como um agrupador de funções.

```
public class CalculadoraDeNotasEstruturada {  
  
    static double calcularMedia(double[] notas) {  
        double soma = 0;  
        for (double nota : notas) {  
            soma += nota;  
        }  
        return soma / notas.length;  
    }  
  
    static String situacao(double media) {  
        if (media >= 7.0) {  
            return "Aprovado";  
        } else if (media >= 5.0) {  
            return "Recuperação";  
        } else {  
            return "Reprovado";  
        }  
    }  
  
    public static void main(String[] args) {  
        double[] notasAna = {8.0, 7.5, 9.0};  
        double[] notasBruno = {4.0, 5.5, 6.0};  
  
        double mediaAna = calcularMedia(notasAna);  
        double mediaBruno = calcularMedia(notasBruno);  
  
        System.out.println("Ana: " + mediaAna + " - " + situacao(mediaAna));  
        System.out.println("Bruno: " + mediaBruno + " - " + situacao(mediaBruno));  
    }  
}
```

Note que os dados de cada aluno (as notas) são simples arrays de double, manipulados por funções externas (calcularMedia, situacao). Não há nenhuma classe representando o conceito de "Aluno" como uma entidade com estado e comportamento próprios.

4. Programação Orientada a Objetos

Na **Programação Orientada a Objetos**, dados (atributos) e comportamentos (métodos) que operam sobre esses dados são agrupados dentro de **classes**, e cada aluno passa a ser representado por um **objeto** com seu próprio estado. O mesmo problema do exemplo anterior pode ser reescrito de forma orientada a objetos.

```
public class Aluno {
    private String nome;
    private double[] notas;

    public Aluno(String nome, double[] notas) {
        this.nome = nome;
        this.notas = notas;
    }

    public double calcularMedia() {
        double soma = 0;
        for (double nota : notas) {
            soma += nota;
        }
        return soma / notas.length;
    }

    public String getSituacao() {
        double media = calcularMedia();
        if (media >= 7.0) {
            return "Aprovado";
        } else if (media >= 5.0) {
            return "Recuperação";
        } else {
            return "Reprovado";
        }
    }

    public String getNome() {
        return nome;
    }
}

public class Main {
    public static void main(String[] args) {
        Aluno ana = new Aluno("Ana", new double[]{8.0, 7.5, 9.0});
        Aluno bruno = new Aluno("Bruno", new double[]{4.0, 5.5, 6.0});

        System.out.println(ana.getNome() + ": " + ana.calcularMedia() + " - " + ana.getSituacao());
        System.out.println(bruno.getNome() + ": " + bruno.calcularMedia() + " - " + bruno.getSituacao());
    }
}
```

Agora, `calcularMedia()` e `getSituacao()` são operações que pertencem ao próprio objeto `Aluno`, que "sabe" calcular sua média e determinar sua situação a partir de seus próprios dados internos.

5. Comparação entre os Paradigmas

Aspecto	Programação Estruturada	Programação Orientada a Objetos
Organização do código	Funções/procedimentos separados dos dados	Classes que agrupam dados e comportamentos

Unidade principal	Função	Objeto
Reutilização de código	Por meio de chamadas de função	Por meio de herança e composição
Encapsulamento	Limitado ou inexistente	Suportado nativamente (private, get/set)
Modelagem do mundo real	Menos natural	Mais próxima de entidades reais
Indicado para	Scripts simples, algoritmos isolados	Sistemas grandes e complexos

6. Quando Usar Cada Abordagem

A programação estruturada continua sendo útil para scripts pequenos, algoritmos isolados ou tarefas de processamento de dados simples, em que criar classes traria complexidade desnecessária. Já a orientação a objetos se destaca em sistemas maiores, com múltiplas entidades que interagem entre si, pois facilita a organização, a manutenção e a evolução do código ao longo do tempo — características especialmente valorizadas em projetos profissionais desenvolvidos, por exemplo, no próprio IntelliJ IDEA.

7. Conclusão

O IntelliJ IDEA oferece um ambiente completo para escrever, executar e depurar código Java, seja ele estruturado ou orientado a objetos. Embora ambos os paradigmas possam resolver o mesmo problema — como visto no exemplo do cálculo de médias —, a orientação a objetos tende a produzir um código mais organizado, reutilizável e alinhado com a forma como pensamos sobre entidades do mundo real, sendo por isso o paradigma predominante em Java e em grande parte dos sistemas profissionais atuais.