INF008 – Programação Orientada a Objetos

3ª. Lista de Exercícios

Questão 1: Sistema de Análise de Dados com Múltiplos Algoritmos e Fontes

Implemente um framework de análise de dados que suporte diferentes algoritmos e fontes de dados através de hierarquias independentes. O sistema deve permitir combinar algoritmos e fontes dinamicamente. Novas implementações devem ser suportadas sem alterações nos métodos `executeAnalysis()`, `executeBatchAnalysis()` e `generateComparativeReport()`.

Código do cliente:

```
public class DataAnalyticsApp {
    public static void main(String[] args) {
        AnalyticsPipeline pipeline = new AnalyticsPipeline();

        // Diferentes combinações de algoritmos e fontes
        AnalysisResult result1 = pipeline.executeAnalysis(
            new MLRegressionAnalyzer(),
            new DatabaseDataSource("sales_db")
        );

        AnalysisResult result2 = pipeline.executeAnalysis(
            new StatisticalAnalyzer(),
            new FileDataSource("data.csv")
        );

        AnalysisResult result3 = pipeline.executeAnalysis(
            new NeuralNetworkAnalyzer(),
            new APIDataSource("https://api.data.com")
        );

        // Processamento em lote com múltiplas combinações
```

```
        Map<IDataAnalyzer, IDataSource> analysisJobs = new LinkedHashMap<>();
        analysisJobs.put(new MLRegressionAnalyzer(),
                         new DatabaseDataSource("db1"));
        analysisJobs.put(new StatisticalAnalyzer(),
                         new FileDataSource("file1.csv"));

        List<AnalysisResult> batchResults =
                                    pipeline.executeBatchAnalysis(analysisJobs);

        // Análise comparativa
        ComparativeReport report = pipeline.generateComparativeReport(
            Arrays.asList(new MLRegressionAnalyzer(),
                          new StatisticalAnalyzer()),
            new DatabaseDataSource("comparison_db")
        );
    }
}
```

Questão 2: Sistema de Processamento de Pedidos para E-commerce com Múltiplas Estratégias

Desenvolva um sistema de processamento de pedidos que utilize hierarquias paralelas para estratégias de processamento e cálculo de frete. O sistema deve ser extensível para novos tipos de pedidos e regras de negócio sem requerer mudanças nos métodos `processOrder()` e `processOrderBatch()`.

Código do cliente:

```
public class OrderManagementApp {
    public static void main(String[] args) {
        OrderProcessor processor = new OrderProcessor();

        // Diferentes tipos de pedidos com estratégias específicas
        Order standardOrder = new StandardOrder(items, customer);
```

```
Order expressOrder = new ExpressOrder(items, customer);
Order internationalOrder = new InternationalOrder(items, customer);

// Processamento polimórfico através de múltiplas hierarquias
ProcessingResult result1 = processor.processOrder(
    standardOrder,
    new StandardProcessingStrategy(),
    new DomesticShippingCalculator()
);

ProcessingResult result2 = processor.processOrder(
    expressOrder,
    new ExpressProcessingStrategy(),
    new PriorityShippingCalculator()
);

ProcessingResult result3 = processor.processOrder(
    internationalOrder,
    new InternationalProcessingStrategy(),
    new InternationalShippingCalculator()
);

// Processamento em lote com diferentes combinações
List<Order> orderBatch = Arrays.asList(standardOrder, expressOrder);
Map<Order, IProcessingStrategy> strategyMap = new HashMap<>();
strategyMap.put(standardOrder, new StandardProcessingStrategy());
strategyMap.put(expressOrder, new ExpressProcessingStrategy());

List<ProcessingResult> batchResults = processor.processOrderBatch(
    orderBatch,
    strategyMap,
    new DomesticShippingCalculator()
);
```

```
    }
}
```

Questão 3: Sistema de Processamento de Pagamentos para E-commerce

Implemente um sistema de pagamentos que suporte múltiplas formas de pagamento através de hierarquias paralelas. O sistema deve utilizar um registro dinâmico de processadores de pagamento e analisadores de risco. O código cliente deve demonstrar o uso de classes concretas específicas.

Código do cliente:

```
public class ECommerceApp {
    public static void main(String[] args) {
        PaymentOrchestrator orchestrator = new PaymentOrchestrator();

        // Registro de processadores
        // cada um sabe quais tipos de pagamento suporta
        orchestrator.registerPaymentProcessor(new CreditCardProcessor());
        orchestrator.registerPaymentProcessor(new PIXProcessor());
        orchestrator.registerPaymentProcessor(new CryptoProcessor());

        // Registro de analisadores de risco
        orchestrator.registerRiskAnalyzer("high", new HighRiskAnalyzer());
        orchestrator.registerRiskAnalyzer("low", new LowRiskAnalyzer());
        orchestrator.registerRiskAnalyzer("medium", new MediumRiskAnalyzer());

        // Processamento - o próprio PAGAMENTO determina qual processador usar
        // IMPOSSÍVEL usar CreditCardPayment com PIXProcessor
        PaymentResult result1 = orchestrator.processPayment(
            new CreditCardPayment(150.0, "4111111111111111", "12/25"),
            "high"  // Apenas chave do analisador de risco
        );
```

```
        PaymentResult result2 = orchestrator.processPayment(
            new PIXPayment(200.0, "123e4567-e89b-12d3-a456-426614174000"),
            "low"   // Apenas chave do analisador de risco
        );

        PaymentResult result3 = orchestrator.processPayment(
            new CryptoPayment(300.0, "0x742d35Cc6634C0532925a3b8D", "ETH"),
            "medium" // Apenas chave do analisador de risco
        );

        // Processamento em lote - type safe
        List<IPayment> batchPayments = Arrays.asList(
            new CreditCardPayment(100.0, "4222222222222222", "06/24"),
            new PIXPayment(50.0, "123e4567-e89b-12d3-a456-426614174001"),
            new CryptoPayment(75.0, "0x842d35Cc6634C0532925a3b8E", "BTC")
        );

        List<PaymentResult> batchResults = orchestrator.processBatch(
            batchPayments,
            "low"  // Analisador de risco para todo o lote
        );

        // Adição de novo tipo de pagamento SEM risco de combinação inválida
        // orchestrator.registerPaymentProcessor(new DigitalWalletProcessor());
        // new DigitalWalletPayment(...)
        //           só funcionará com DigitalWalletProcessor
    }
}
```