

Programação Orientada a Aspectos (AOP)

AOP (*Aspect-Oriented Programming*) é uma técnica usada para separar **preocupações transversais** (ou **cross-cutting concerns**) em um sistema. São funcionalidades que, em vez de serem diretamente relacionadas ao propósito principal das classes, são aspectos auxiliares que afetam várias partes do código. Exemplos incluem:

- **Logs;**
- **Segurança;**
- **Controle de transações;**
- **Autenticação e autorização.**

Em vez de adicionar código de log ou segurança diretamente nas classes, esses comportamentos são isolados em **aspectos** e depois aplicados de forma automática onde necessário, usando um mecanismo de interceptação. Isso permite **manutenção mais fácil** e **separação de responsabilidades**.

Em Java, **AspectJ** é uma implementação poderosa de AOP. Ele permite a inserção de código antes, durante ou depois de métodos ou blocos de código sem modificar diretamente o código original.

Agora que entendemos a base, vamos analisar o projeto exemplo que implementa AOP:

1. Account.java

Contem a lógica básica de uma conta bancária. A classe Account possui um atributo balance (saldo) e um método withdraw(int amount) para realizar saques. A lógica do método de saque verifica se o saldo é suficiente para a retirada; se não for, retorna false. Caso contrário, atualiza o saldo e retorna true:

- **balance:** Define o saldo inicial da conta.
- **withdraw(int amount):** Verifica se há saldo suficiente para sacar o valor solicitado e, se houver, atualiza o saldo.

```
public class Account {  
    int balance = 20;  
    public boolean withdraw(int amount) {  
        if (balance < amount) {  
            return false;  
        }  
        balance = balance - amount;  
        return true;  
    }  
}
```

2. AccountAspect.aj

Esse arquivo contém o aspecto (aspect) em **AspectJ** para lidar com saques de uma conta. Ele faz parte da AOP e está interceptando a execução de saques na classe Account. O

aspecto define **pontos de corte** (*pointcuts*) e **conselhos** (*advices*) para adicionar comportamento extra ao método `withdraw()`.

Extensão .aj

A extensão `.aj` indica que esse arquivo é um aspecto de **AspectJ**, uma linguagem de extensão para Java projetada para programar aspectos. AspectJ usa a extensão `.aj` para diferenciar código regular Java de código de aspecto, que segue uma sintaxe específica para definir os pontos de interceptação:

- **`pointcut callWithdraw(int amount, Account account)`**: Define o ponto de corte, ou seja, onde o aspecto vai agir. Neste caso, ele intercepta qualquer chamada ao método `withdraw(int)` da classe `Account`.
- **`before(int amount, Account account)`**: Define um conselho que executa **antes** da execução do método `withdraw`. Ele registra o saldo atual e o valor do saque no log usando a biblioteca SLF4J.
- **`around(int amount, Account account)`**: Este é um conselho ao redor (*around advice*), que substitui a execução original do método. Ele verifica se o saldo é menor que o valor solicitado e, se for, bloqueia o saque e registra uma mensagem de erro no log.
- **`after(int amount, Account account)`**: Um conselho que executa **depois** da chamada do método `withdraw`. Ele registra o saldo atualizado no log.

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public aspect AccountAspect {
    private static final Logger logger = LoggerFactory.getLogger(AccountAspect.class);
    final int MIN_BALANCE = 10;

    pointcut callWithdraw(int amount, Account account):
        call(boolean posgrad.utfpr.edu.br.exemploaspectj.Account.withdraw(int)) &&
        args(amount) && target(account);

    before(int amount, Account account) : callWithdraw(amount, account) {
        logger.info(" [AspectJ] Balance before withdrawal: {}",
            Integer.valueOf(account.balance));
        logger.info(" [AspectJ] Withdraw amount: {}", Integer.valueOf(amount));
    }

    boolean around(int amount, Account account) : callWithdraw(amount, account) {
        if (account.balance < amount) {
            logger.info(" [AspectJ] Withdrawal Rejected!");
            return false;
        }
        return proceed(amount, account);
    }

    after(int amount, Account account) : callWithdraw(amount, account) {
        logger.info(" [AspectJ] Balance after withdrawal : {}",
            Integer.valueOf(account.balance));
    }
}
```

3. ExemploAspect.java

Esse arquivo contém o **método principal** da aplicação, ou seja, o ponto de entrada onde a execução começa. Ele instancia a classe `Account` e realiza algumas operações de saque para testar a lógica implementada no `Account.java` e o comportamento do aspecto `AccountAspect.aj`.

O que faz:

- **Instancia a classe `Account`:** Cria uma nova conta com o saldo inicial definido na classe `Account` (que é 20).
- **Executa saques:**
 - Primeiro tenta sacar **5 unidades** da conta, o que deve ser bem-sucedido (saldo após o saque: 15).
 - Depois tenta sacar **10 unidades**, que também deve funcionar (saldo após o saque: 5).
 - Por fim, tenta sacar **39 unidades**, o que vai falhar porque o saldo é insuficiente.

Essas operações geram **logs** no console, controlados pelo aspecto `AccountAspect.aj`, que intercepta as chamadas para `withdraw()`. Assim, a cada operação, o saldo anterior, o valor a ser sacado e o saldo final são registrados no log.

```
public class ExemploAspectJ {  
    public static void main(String[] args) {  
        Account account = new Account();  
        account.withdraw(5);  
        account.withdraw(10);  
        account.withdraw(39);  
    }  
}
```

4. logback.xml

Esse arquivo é a configuração do **Logback**, uma popular biblioteca de logging usada em aplicações Java. Ele define como e onde os logs serão exibidos. Aqui, está configurado para enviar logs para o console.

O que faz:

- **<appender name="CONSOLE">**: Define que os logs serão exibidos no console.
- **<pattern>**: Define o formato da mensagem de log. No caso, os logs exibirão a hora, a thread, o nível de log (INFO, WARN, ERROR) e a mensagem de log.
- **<root level="info">**: Define o nível mínimo de log como `INFO`, ou seja, apenas mensagens com nível de `INFO` ou superior serão registradas.

Esse arquivo é importante no contexto da AOP porque os *advices* no aspecto `AccountAspect.aj` usam o logger para registrar informações, e esse arquivo define como essas mensagens serão gerenciadas e exibidas.

Fluxo de execução:

1. O programa começa no `ExemploAspectJ.main()` e cria uma instância de `Account`.
2. Ele tenta realizar alguns saques, chamando o método `withdraw()`.
3. O aspecto definido em `AccountAspect.aj` intercepta essas chamadas, aplicando o código adicional de log antes, durante e depois dos saques.
4. Dependendo do saldo e do valor do saque, o programa pode permitir ou bloquear a operação.
5. Os logs das operações são exibidos no console graças à configuração do `logback.xml`.

DEPENDÊNCIAS (pom.xml)

I. Dependências

Essas são as bibliotecas que projeto precisa para funcionar. São baixadas automaticamente pelo Maven a partir de repositórios, como o **Maven Central**.

```
<dependencies>
  <dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjrt</artifactId>
    <version>${aspectj.version}</version>
  </dependency>
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>2.0.16</version>
  </dependency>
  <dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>1.5.8</version>
  </dependency>
</dependencies>
```

Detalhe de cada dependência:

- **org.aspectj:aspectjrt:**
 - **AspectJ Runtime.** Esta é a biblioteca de **runtime** necessária para executar os aspectos em **AspectJ**. Ela contém as classes e métodos que permitem que a instrumentação dos aspectos funcione corretamente durante a execução do código.
 - A versão `${aspectj.version}` está definida em outra parte do `pom.xml` (geralmente em `<properties>`), permitindo que você altere a versão do AspectJ facilmente sem mexer em vários lugares.
- **org.slf4j:slf4j-api:**
 - **Simple Logging Facade for Java (SLF4J)** é uma API de abstração para sistemas de log. Ela permite que você troque a implementação de log (por exemplo, **Logback**, **Log4j**) sem alterar o código do projeto. No seu projeto, o AspectJ está usando o SLF4J para emitir logs das operações de saque.

- `ch.qos.logback:logback-classic`:
 - **Logback** é uma implementação concreta do SLF4J, que realiza a gravação dos logs. No projeto, ele é usado para configurar e exibir logs no console ou em arquivos, conforme definido no `logback.xml`.
-

II. Configurações de Build

Aqui estão os **plugins** que o Maven usa para construir e processar o projeto.

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.10.1</version>
      <configuration>
        <source>${maven.compiler.source}</source>
        <target>${maven.compiler.target}</target>
      </configuration>
    </plugin>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>aspectj-maven-plugin</artifactId>
      <version>1.14.0</version>
      <configuration>
        <complianceLevel>20</complianceLevel>
        <source>${maven.compiler.source}</source>
        <target>${maven.compiler.target}</target>
        <showWeaveInfo>true</showWeaveInfo>
        <verbose>true</verbose>
        <Xlint>ignore</Xlint>
        <encoding>UTF-8</encoding>
      </configuration>
      <executions>
        <execution>
          <goals>
            <goal>compile</goal>
            <goal>test-compile</goal>
          </goals>
        </execution>
      </executions>
      <dependencies>
        <dependency>
          <groupId>org.aspectj</groupId>
          <artifactId>aspectjtools</artifactId>
          <version>${aspectj.version}</version>
        </dependency>
      </dependencies>
    </plugin>
  </plugins>
</build>
```

Maven Compiler Plugin:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
```

```

<artifactId>maven-compiler-plugin</artifactId>
<version>3.10.1</version>
<configuration>
  <source>${maven.compiler.source}</source>
  <target>${maven.compiler.target}</target>
</configuration>
</plugin>

```

Este plugin configura o **compilador Java**:

- **source**: Define a versão do código-fonte Java (por exemplo, Java 20).
- **target**: Define a versão do bytecode gerado (também Java 20 neste caso).

Isso garante que o código seja compilado na versão correta de acordo com a compatibilidade do projeto.

AspectJ Maven Plugin:

```

<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>aspectj-maven-plugin</artifactId>
  <version>1.14.0</version>
  <configuration>
    <complianceLevel>20</complianceLevel>
    <source>${maven.compiler.source}</source>
    <target>${maven.compiler.target}</target>
    <showWeaveInfo>true</showWeaveInfo>
    <verbose>true</verbose>
    <Xlint>ignore</Xlint>
    <encoding>UTF-8</encoding>
  </configuration>
  <executions>
    <execution>
      <goals>
        <goal>compile</goal>
        <goal>test-compile</goal>
      </goals>
    </execution>
  </executions>
  <dependencies>
    <dependency>
      <groupId>org.aspectj</groupId>
      <artifactId>aspectjtools</artifactId>
      <version>${aspectj.version}</version>
    </dependency>
  </dependencies>
</plugin>

```

Esse é o plugin que realiza a **compilação do AspectJ** no projeto. Ele garante que os aspectos (.aj) sejam processados junto com o código Java e aplicados corretamente.

Configurações importantes:

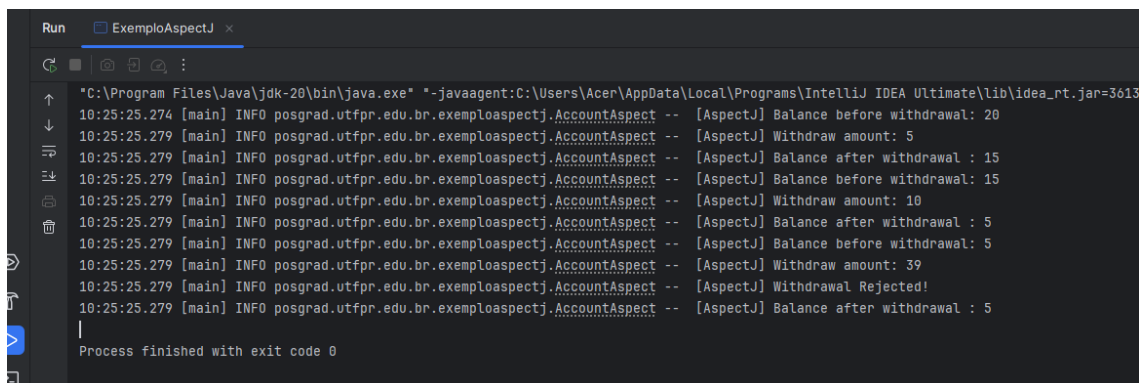
- **complianceLevel**: Especifica o nível de conformidade com a versão Java. Aqui está configurado para o **Java 20**, o que significa que o código será compilado de acordo com as regras e padrões dessa versão.

- **showWeaveInfo** e **verbose**: Essas opções estão ativadas (`true`) para exibir informações detalhadas sobre o processo de instrumentação do AspectJ, como quais métodos estão sendo interceptados e quais aspectos estão sendo aplicados.
- **xlint**: Aqui está definido como `ignore`, o que significa que os avisos de lint (análises estáticas) relacionados ao AspectJ serão ignorados. Isso pode ser útil para evitar avisos que não são críticos.
- **executions**: Define que o plugin será executado durante a fase de **compilação e testes** (`compile` e `test-compile`), garantindo que os aspectos sejam aplicados tanto no código de produção quanto no código de teste.
- **dependencies**:
 - **org.aspectj:aspectjtools**: Esta dependência fornece as ferramentas necessárias para o **compilador do AspectJ**. É ela que realiza a instrumentação dos aspectos no código.

Dessa forma o `pom.xml` do projeto está configurado para:

1. **Compilar** o código Java usando a versão especificada (neste caso, Java 20).
2. **Instrumentar** os aspectos com o plugin do **AspectJ Maven**.
3. **Gerenciar logs** usando **SLF4J** e **Logback** para exibir as informações sobre o comportamento dos aspectos e do código Java.

E o resultado final:



```

Run ExemploAspectJ x
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:C:\Users\Acer\AppData\Local\Programs\IntelliJ IDEA Ultimate\lib\idea_rt.jar=3613
10:25:25.274 [main] INFO posgrad.utfpr.edu.br.exemploaspectj.AccountAspect -- [AspectJ] Balance before withdrawal: 20
10:25:25.279 [main] INFO posgrad.utfpr.edu.br.exemploaspectj.AccountAspect -- [AspectJ] Withdraw amount: 5
10:25:25.279 [main] INFO posgrad.utfpr.edu.br.exemploaspectj.AccountAspect -- [AspectJ] Balance after withdrawal : 15
10:25:25.279 [main] INFO posgrad.utfpr.edu.br.exemploaspectj.AccountAspect -- [AspectJ] Balance before withdrawal: 15
10:25:25.279 [main] INFO posgrad.utfpr.edu.br.exemploaspectj.AccountAspect -- [AspectJ] Withdraw amount: 10
10:25:25.279 [main] INFO posgrad.utfpr.edu.br.exemploaspectj.AccountAspect -- [AspectJ] Balance after withdrawal : 5
10:25:25.279 [main] INFO posgrad.utfpr.edu.br.exemploaspectj.AccountAspect -- [AspectJ] Balance before withdrawal: 5
10:25:25.279 [main] INFO posgrad.utfpr.edu.br.exemploaspectj.AccountAspect -- [AspectJ] Withdraw amount: 39
10:25:25.279 [main] INFO posgrad.utfpr.edu.br.exemploaspectj.AccountAspect -- [AspectJ] Withdrawal Rejected!
10:25:25.279 [main] INFO posgrad.utfpr.edu.br.exemploaspectj.AccountAspect -- [AspectJ] Balance after withdrawal : 5
Process finished with exit code 0

```

O git deste projeto está em

<https://github.com/rgiovann/AspecjJExemplo/>