PRÁTICA LABORATORIAL FACTORY METHOD & STRATEGY

Objetivos:

- Implementar um exemplo de Factory Method & Strategy

EXERCÍCIOS

Parte 1

Crie uma plataforma de pagamento em Paypal ou Cartão de Crédito.

Temos as interfaces PaymentStrategy e PaymentFactory que definem os contratos para as estratégias de pagamento e fábricas de estratégias de pagamento, respetivamente.

As classes CreditCardPaymentStrategy e PayPalPaymentStrategy são implementações concretas da interface PaymentStrategy. Elas implementam o método pay, que representa a ação de pagar um determinado valor.

As classes CreditCardPaymentFactory e PayPalPaymentFactory são implementações concretas da interface PaymentFactory. Elas implementam o método createPaymentStrategy, que cria uma instância da estratégia de pagamento correspondente.No método main, criamos instâncias das fábricas CreditCardPaymentFactory e PayPalPaymentFactory e, de seguida, usamos essas fábricas para criar as estratégias de pagamento correspondentes.Finalmente, utilizamos as estratégias de pagamento para fazer os pagamentos chamando o método pay com os valores apropriados.

```
// Interface Strategy
interface PaymentStrategy {
    void pay(double amount);
}

// Implementações concretas da interface Strategy
class CreditCardPaymentStrategy implements PaymentStrategy {
    private String cardNumber;
    private String expirationDate;
    private String cvv;

    public CreditCardPaymentStrategy(String cardNumber, String expirationDate, String cvv) {
        this.cardNumber = cardNumber;
        this.expirationDate = expirationDate;
        this.cvv = cvv;
    }

    @Override
    public void pay(double amount) {
        System.out.println("Paying " + amount + " using credit card: " + cardNumber);
    }
}

class PayPalPaymentStrategy implements PaymentStrategy {
    private String email;
    private String password;

    public PayPalPaymentStrategy(String email, String password) {
        this.email = email;
        this.password = password;
    }

    @Override
    public void pay(double amount) {
        System.out.println("Paying " + amount + " using PayPal");
    }
}

// Interface Factory
interface PaymentFactory {
    PaymentStrategy createPaymentStrategy();
}
```

```
// Implementações concretas da interface Factory
class CreditCardPaymentFactory implements PaymentFactory {
    private String cardNumber;
    private String expirationDate;
    private String cvv;

    public CreditCardPaymentFactory(String cardNumber, String expirationDate, String cvv) {
        this.cardNumber = cardNumber;
        this.expirationDate = expirationDate;
        this.cvv = cvv;
    }

    @Override
    public PaymentStrategy createPaymentStrategy() {
        return new CreditCardPaymentStrategy(cardNumber, expirationDate, cvv);
    }
}

class PayPalPaymentFactory implements PaymentFactory {
    private String email;
    private String password;

    public PayPalPaymentFactory(String email, String password) {
        this.email = email;
        this.password = password;
    }

    @Override
    public PaymentStrategy createPaymentStrategy() {
        return new PayPalPaymentStrategy(email, password);
    }
}

// Client
public class Main {
    public static void main(String[] args) {
        // A criar uma instância do Factory para criar a estratégia de pagamento com cartão de crédito
        PaymentFactory creditCardPaymentFactory = new CreditCardPaymentFactory("1234567890123456", "12/23",
"123");
        PaymentStrategy creditCardPaymentStrategy = creditCardPaymentFactory.createPaymentStrategy();

        // A criar uma instância do Factory para criar a estratégia de pagamento com PayPal
        PaymentFactory payPalPaymentFactory = new PayPalPaymentFactory("example@example.com", "password");
        PaymentStrategy payPalPaymentStrategy = payPalPaymentFactory.createPaymentStrategy();

        // A utilizar a estratégia de pagamento
        creditCardPaymentStrategy.pay(100.0);
        payPalPaymentStrategy.pay(50.0);
    }
}
```

2. Adicione o código para implementar um pagamento por transferência bancária tendo o número da conta como argumento. Recordar que transferência bancária terá uma fábrica e uma estratégia própria.

3. Adicione o código necessário para perguntar ao utilizador que tipo de pagamento quer usar e, de seguida, o montante que quer pagar.

4. Crie um programa de encriptação de texto.
Crie as interfaces EncryptionStrategy e EncryptionStrategyFactory que definem os contratos para as estratégias de criptografia e fábricas de estratégias de criptografia, respectivamente.
As classes AESEncryptionStrategy e DESEncryptionStrategy são implementações concretas da interface EncryptionStrategy. Devem implementar o método encrypt, que representa a ação de criptografar um texto.
As classes AESEncryptionStrategyFactory e DESEncryptionStrategyFactory são implementações concretas da interface EncryptionStrategyFactory. Elas implementam o método createEncryptionStrategy, que cria uma instância da estratégia de criptografia correspondente.
No método main, pergunte ao utilizador qual cifra que quer usar. Crie as instâncias das fábricas AESEncryptionStrategyFactory e DESEncryptionStrategyFactory e, em seguida, use essas fábricas para criar as estratégias de criptografia correspondentes.
Finalmente, utilize as estratégias de criptografia para criptografar um texto do utilizador chamando o método encrypt com o texto apropriado.

Exemplo de Cifra AES:

```
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
import java.nio.charset.StandardCharsets;
import java.util.Base64;

public class AESEncryptionExample {

    private static final String AES_ALGORITHM = "AES";
    private static final String SECRET_KEY = "mysecretkey12345"; // Chave secreta de 128 bits

    public static void main(String[] args) {
        String plainText = "Hello, World!";
        String encryptedText = encrypt(plainText);
        System.out.println("Encrypted Text: " + encryptedText);
    }

    public static String encrypt(String plainText) {
        try {
            SecretKeySpec secretKey = new SecretKeySpec(SECRET_KEY.getBytes(StandardCharsets.UTF_8),
AES_ALGORITHM);
            Cipher cipher = Cipher.getInstance(AES_ALGORITHM);
            cipher.init(Cipher.ENCRYPT_MODE, secretKey);
            byte[] encryptedBytes = cipher.doFinal(plainText.getBytes(StandardCharsets.UTF_8));
            return Base64.getEncoder().encodeToString(encryptedBytes);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

Cifra DES:

```
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.DESKeySpec;
import java.nio.charset.StandardCharsets;
import java.util.Base64;

public class DESEncryptionExample {

    private static final String DES_ALGORITHM = "DES";
    private static final String SECRET_KEY = "mysecret"; // Chave secreta de 8 caracteres

    public static void main(String[] args) {
        String plainText = "Hello, World!";
        String encryptedText = encrypt(plainText);
        System.out.println("Encrypted Text: " + encryptedText);
    }

    public static String encrypt(String plainText) {
        try {
            DESKeySpec desKeySpec = new DESKeySpec(SECRET_KEY.getBytes(StandardCharsets.UTF_8));
            SecretKeyFactory keyFactory = SecretKeyFactory.getInstance(DES_ALGORITHM);
            SecretKey secretKey = keyFactory.generateSecret(desKeySpec);

            Cipher cipher = Cipher.getInstance(DES_ALGORITHM);
            cipher.init(Cipher.ENCRYPT_MODE, secretKey);

            byte[] encryptedBytes = cipher.doFinal(plainText.getBytes(StandardCharsets.UTF_8));
            return Base64.getEncoder().encodeToString(encryptedBytes);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

Bom trabalho! ☺