

Mini Meter Reading and Billing System



Mini Meter Reading and Billing System

قسم هندسة البرمجيات – كلية تقنية المعلومات جامعة صبراتة
مقرر أنماط تصميم البرمجيات (ITSE424)

إعداد:

أيوب احفيظ ابوالقاسم ابوسنينه
نجيد عبدالله عيسى عمارة

إشراف:

د. مي مفتاح البعباع

الفصل خريف 2025

1. المقدمة

□ آلية التطبيق

□ حساب الفواتير

□ نظام الدفع

□ تطبيق أنماط التصميم

□ محاكاة تعليمية

2. أهداف المشروع

❑ نظام فوترة مصغّر

❑ Java + MVC

❑ Design Patterns Implementation


❑ High Cohesion / Low Coupling


❑ تطبيق تعليمي عملي





3. نظرة عامة على النظام


تطبيق Java Console يحاكي نظام فوترة كهربائية مبسطة. 

يتيح للمستخدم إدخال قراءات العدادات وحساب فواتير الكهرباء. 

يدعم تنفيذ عمليات الدفع وتسجيل المعاملات المالية. 

يوفر إمكانية تقديم تقارير عن الأعطال أو الخدمات. 

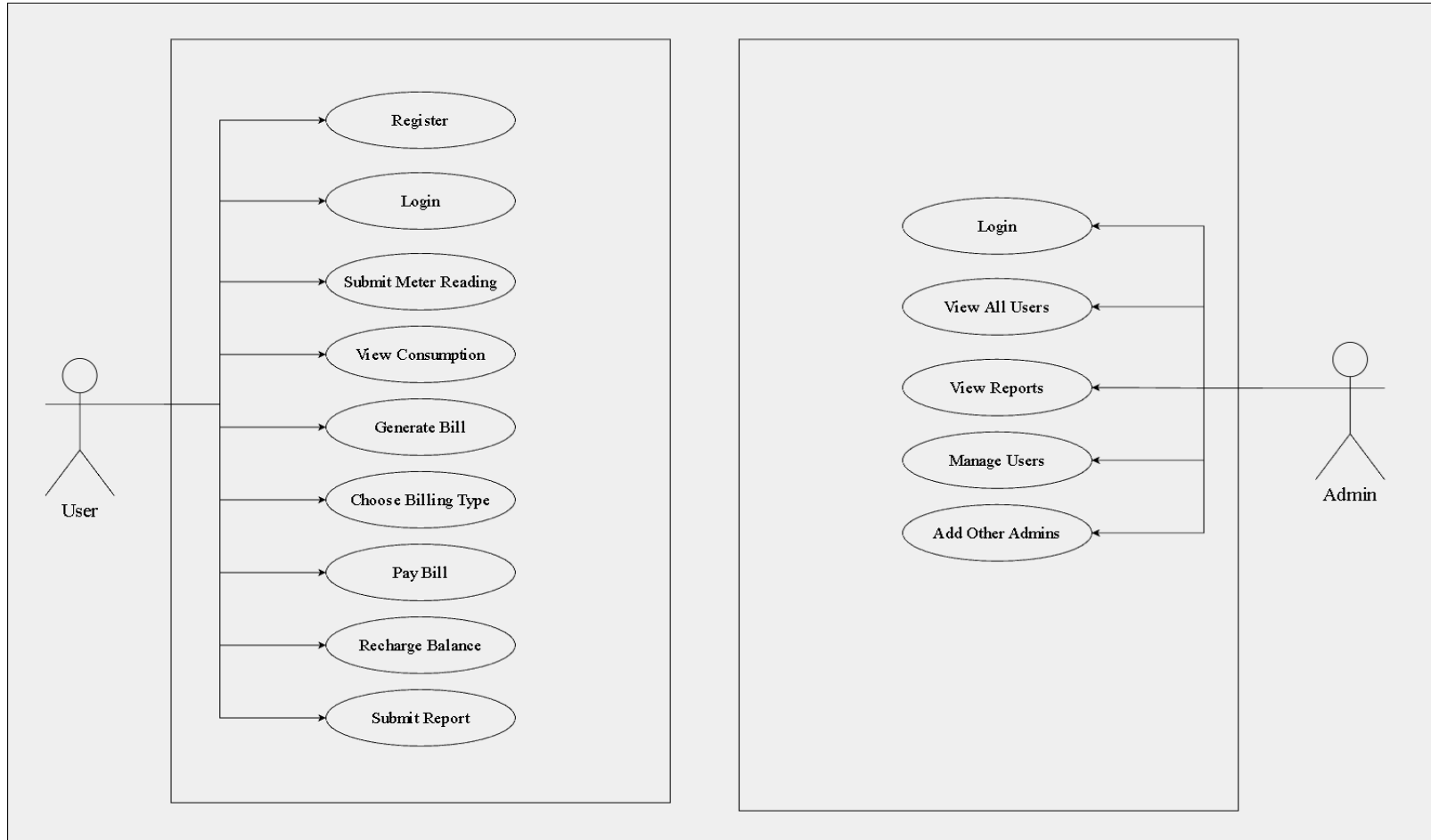
يدعم دورين رئيسيين في النظام: Admin & User. 

ضمم باستخدام معمارية Model–View–Controller (MVC) مع تطبيق عدة أنماط تصميم برمجية. 



4. مخطط حالة الاستخدام

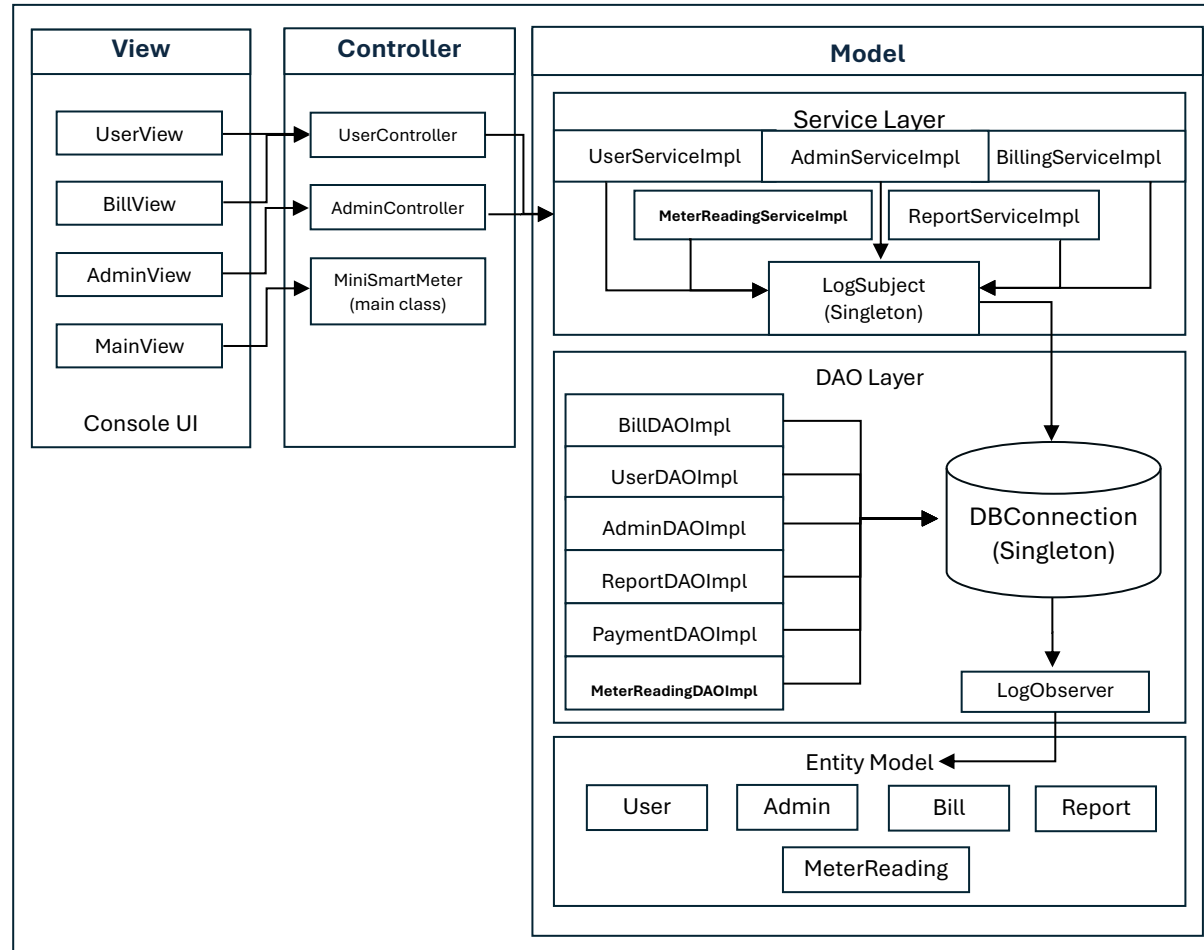
يوضح الشكل التالي مخطط حالات الاستخدام (Use Case Diagram) التفاعلات الرئيسية بين المستخدم والمشرف مع النظام ويبين الوظائف الأساسية التي يدعمها نظام قراءة العدادات والفوترة.





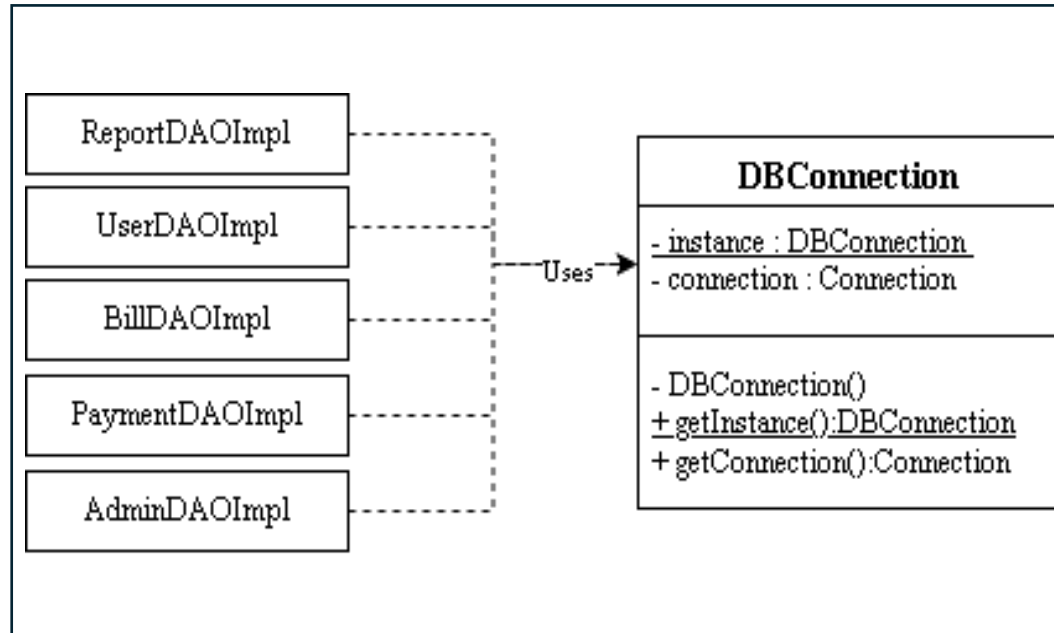
5. بنية النظام (MVC + Layers)

الشكل يوضح اعتماد النظام على معمارية MVC مع تصميم طبقي، حيث يتم فصل واجهة المستخدم عن منطق الأعمال وإدارة البيانات، مما يساهم في وضوح البنية البرمجية وسهولة الصيانة والتوسعة.





6. أنماط التصميم المستخدمة



Singleton Pattern 6.1

❑ الفكرة الأساسية للنمط

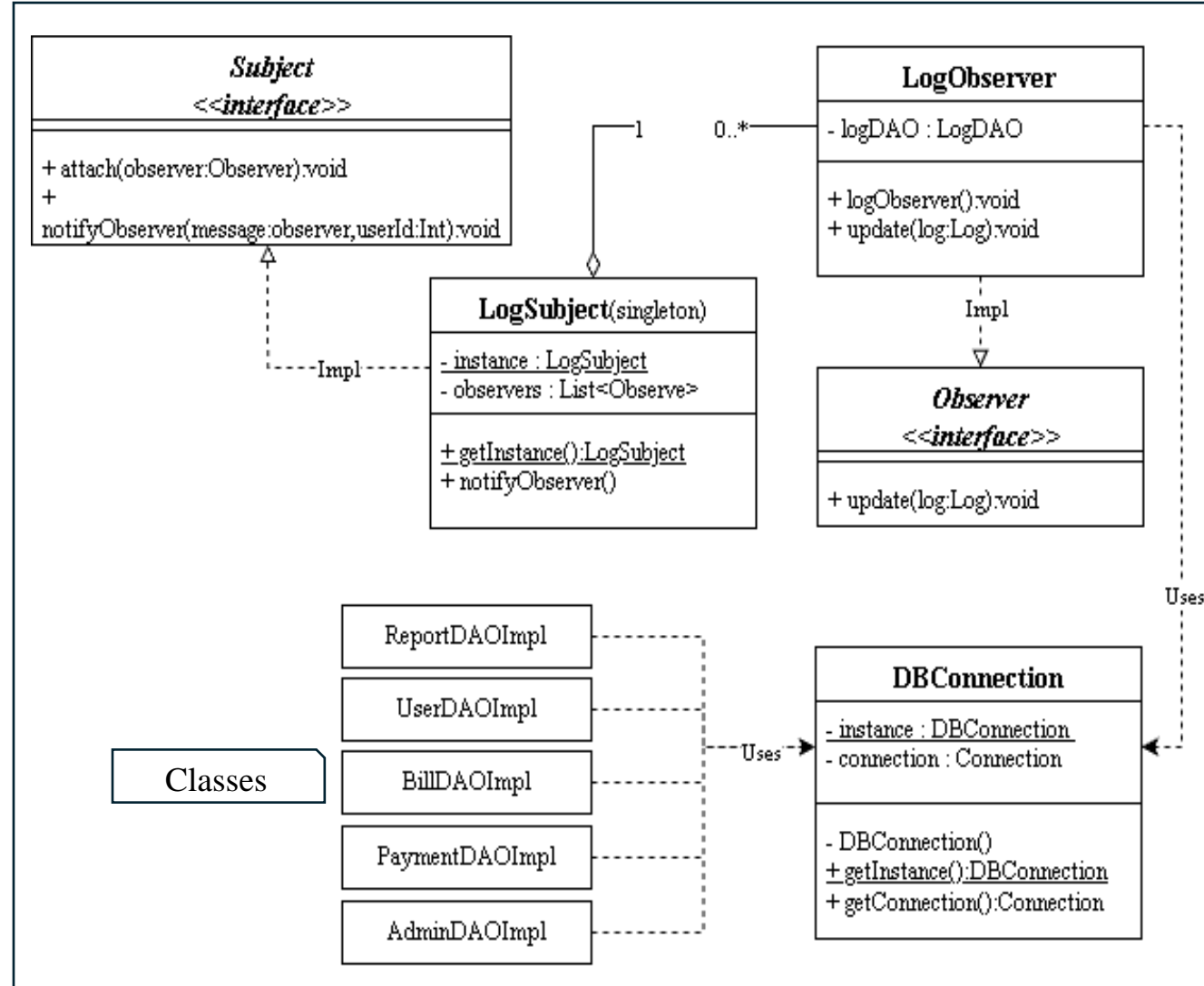
❑ التطبيق في المشروع

❑ المزايا

❑ العيوب



6. أنماط التصميم المستخدمة



Observer Pattern 6.2

❑ الفكرة الأساسية للنمط

❑ التطبيق في المشروع

❑ المزايا

❑ العيوب



6. أنماط التصميم المستخدمة

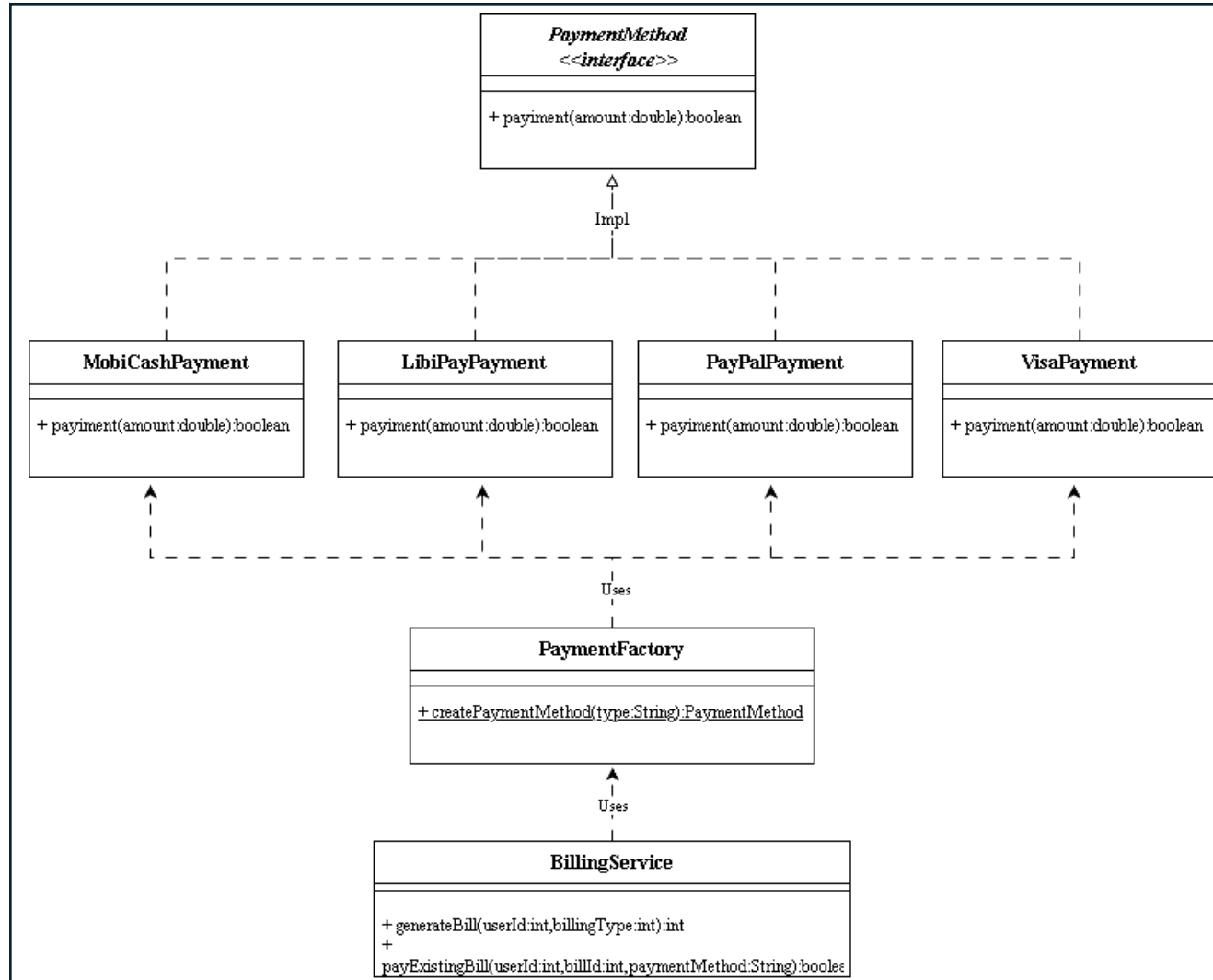
Factory Pattern 6.3

❑ الفكرة الأساسية للنمط

❑ التطبيق في المشروع

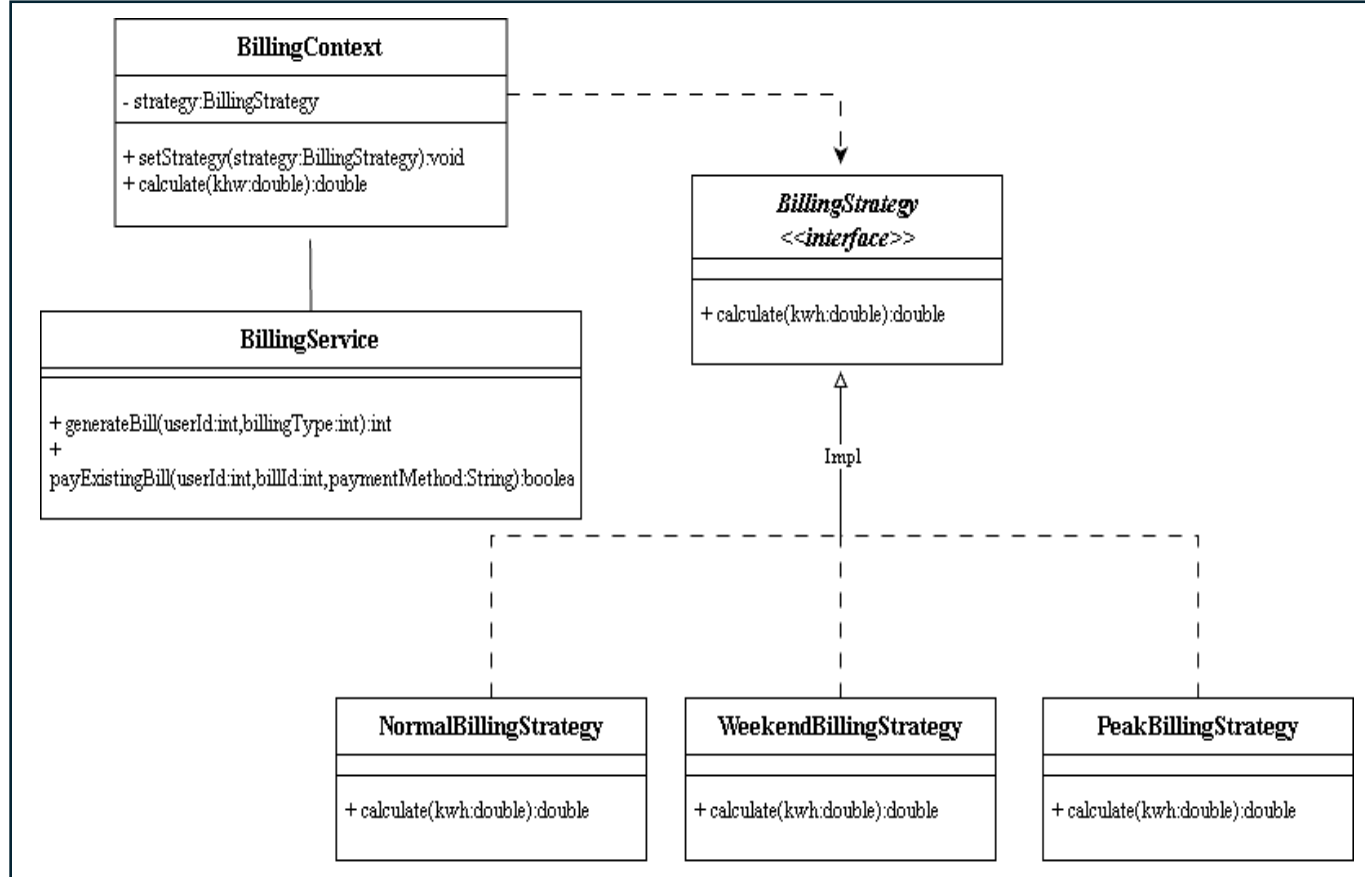
❑ المزايا

❑ العيوب





6. أنماط التصميم المستخدمة



Strategy Pattern 6.4

❑ الفكرة الأساسية للنمط

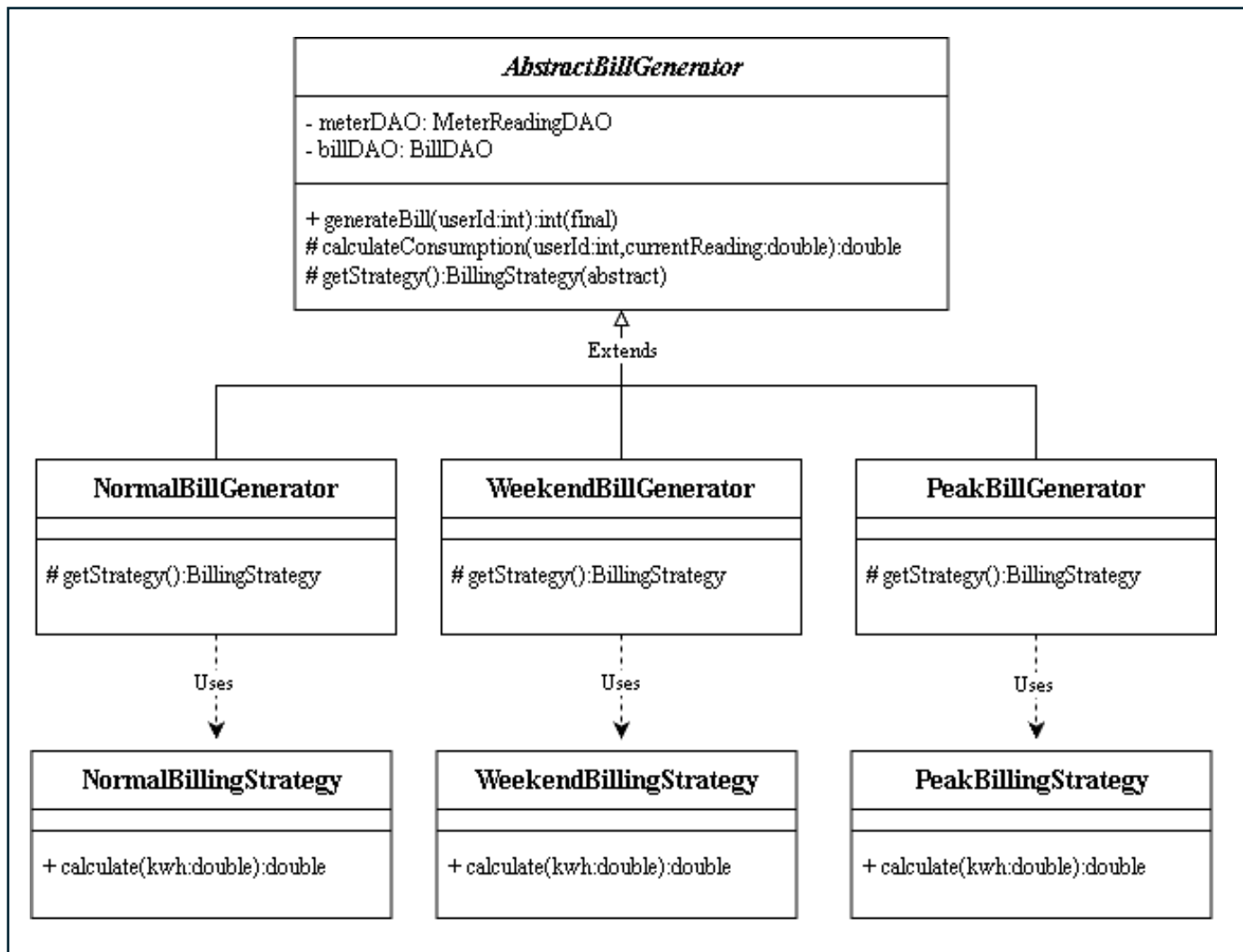
❑ التطبيق في المشروع

❑ المزايا

❑ العيوب



6. أنماط التصميم المستخدمة



Template Pattern 6.5

❑ الفكرة الأساسية للنمط

❑ التطبيق في المشروع

❑ المزايا

❑ العيوب



6. أنماط التصميم المستخدمة

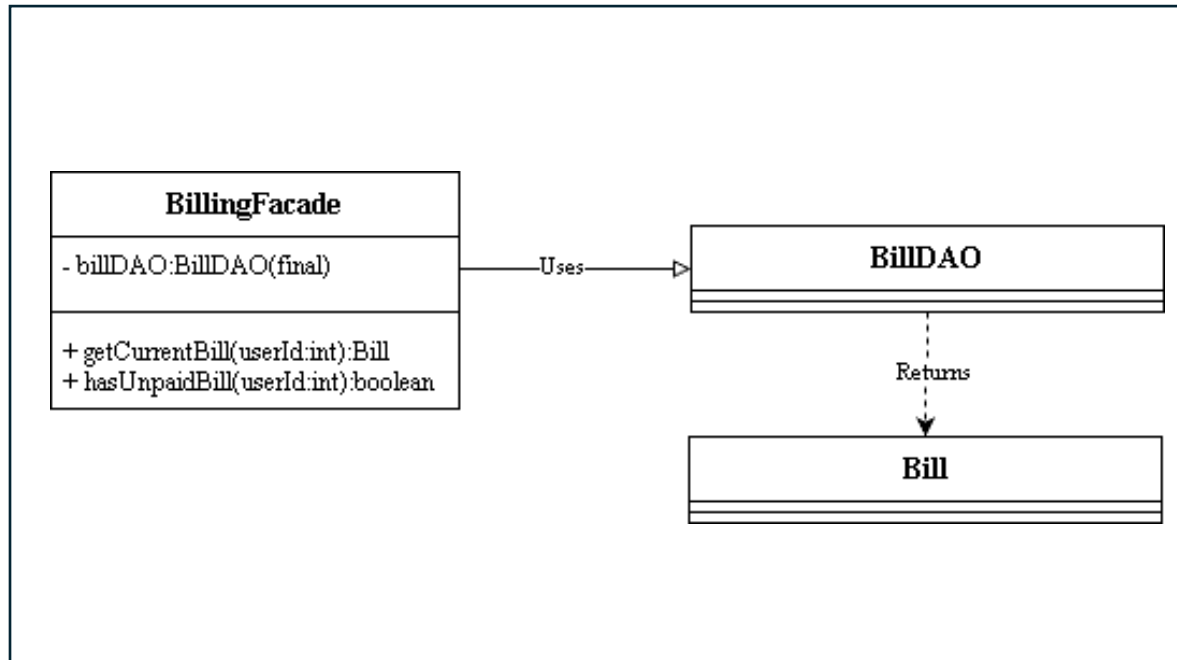
Facade Pattern 6.6

❑ الفكرة الأساسية للنمط

❑ التطبيق في المشروع

❑ المزايا

❑ العيوب





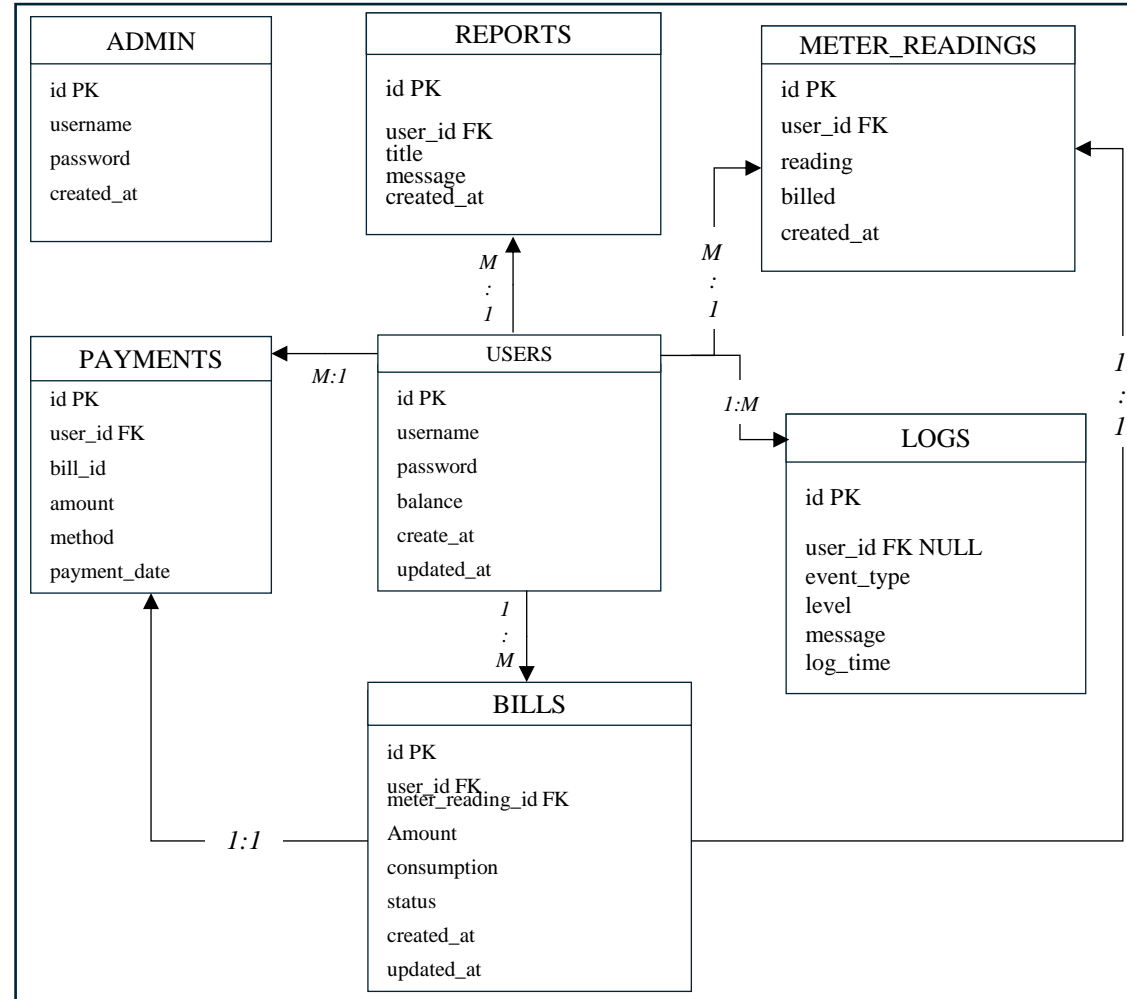
```
classDiagram
    class PaymentMethod {
        <<interface>>
        +payment(amount:double):boolean
    }
    class MobICashPayment {
        +payment(amount:double):boolean
    }
    class LibiPayPayment {
        +payment(amount:double):boolean
    }
    class PayPalPayment {
        +payment(amount:double):boolean
    }
    class VisaPayment {
        +payment(amount:double):boolean
    }
    class Subject {
        <<interface>>
        +attach(observer:Observer):void
        +notifyObserver(message:observer,userid:int):void
    }
    class LogSubject {
        singleton
        -instance:LogSubject
        -observers:List<Observer>
        +getInstance():LogSubject
        +notifyObserver()
    }
    class LogObserver {
        -logDAO:LogDAO
        +logObserver():void
        +update(log:Log):void
    }
    class Observer {
        <<interface>>
        +update(log:Log):void
    }
    class PaymentFactory {
        +createPaymentMethod(type:String):PaymentMethod
    }
    class BillingService {
        +generateBill(userid:int,billingType:int):int
        +payExistingBill(userid:int,billid:int,paymentMethod:String):boolean
    }
    class BillingContext {
        -strategy:BillingStrategy
        +setStrategy(strategy:BillingStrategy):void
        +calculate(kwh:double):double
    }
    class BillingStrategy {
        <<interface>>
        +calculate(kwh:double):double
    }
    class NormalBillingStrategy {
        +calculate(kwh:double):double
    }
    class WeekendBillingStrategy {
        +calculate(kwh:double):double
    }
    class PeakBillingStrategy {
        +calculate(kwh:double):double
    }
    class PeakBillGenerator {
        #getStrategy():BillingStrategy
    }
    class NormalBillGenerator {
        #getStrategy():BillingStrategy
    }
    class WeekendBillGenerator {
        #getStrategy():BillingStrategy
    }
    class ReportDAOImpl
    class UserDAOImpl
    class BillDAOImpl
    class PaymentDAOImpl
    class AdminDAOImpl
    class DBConnection {
        -instance:DBConnection
        -connection:Connection
        -DBConnection()
        +getInstance():DBConnection
        +getConnection():Connection
    }
    class BillingFacade {
        -billDAO:BillDAO(final)
        +getCurrentBill(userid:int):Bill
        +hasUnpaidBill(userid:int):boolean
    }
    class BillDAO {
    }
    class Bill {
    }
    class AbstractBillGenerator {
        -meterDAO:MeterReadingDao
        -billDAO:BillDAO
        +generateBill(userid:int):int(final)
        #calculateConsumption(userid:int,currentReading:double):double
        #getStrategy():BillingStrategy(abstract)
    }

    PaymentMethod <|-- MobICashPayment
    PaymentMethod <|-- LibiPayPayment
    PaymentMethod <|-- PayPalPayment
    PaymentMethod <|-- VisaPayment
    Subject <|-- LogSubject
    LogSubject o-- "1" LogObserver
    LogObserver <|-- Observer
    PaymentFactory <|-- BillingService
    BillingService <|-- BillingContext
    BillingContext <|-- BillingStrategy
    BillingStrategy <|-- NormalBillingStrategy
    BillingStrategy <|-- WeekendBillingStrategy
    BillingStrategy <|-- PeakBillingStrategy
    PeakBillGenerator <|-- NormalBillGenerator
    PeakBillGenerator <|-- WeekendBillGenerator
    ReportDAOImpl ..> DBConnection
    UserDAOImpl ..> DBConnection
    BillDAOImpl ..> DBConnection
    PaymentDAOImpl ..> DBConnection
    AdminDAOImpl ..> DBConnection
    DBConnection <|-- BillingFacade
    BillingFacade <|-- BillingContext
    BillingContext <|-- BillingStrategy
    BillingStrategy <|-- NormalBillingStrategy
    BillingStrategy <|-- WeekendBillingStrategy
    BillingStrategy <|-- PeakBillingStrategy
    PeakBillGenerator <|-- NormalBillGenerator
    PeakBillGenerator <|-- WeekendBillGenerator
    ReportDAOImpl ..> DBConnection
    UserDAOImpl ..> DBConnection
    BillDAOImpl ..> DBConnection
    PaymentDAOImpl ..> DBConnection
    AdminDAOImpl ..> DBConnection
    DBConnection <|-- BillingFacade
    BillingFacade <|-- BillingContext
    BillingContext <|-- BillingStrategy
    BillingStrategy <|-- NormalBillingStrategy
    BillingStrategy <|-- WeekendBillingStrategy
    BillingStrategy <|-- PeakBillingStrategy
    PeakBillGenerator <|-- NormalBillGenerator
    PeakBillGenerator <|-- WeekendBillGenerator
```



7. تصميم قاعدة البيانات (ERD)

يوضح هذا الشكل مخطط قاعدة البيانات ERD للنظام، مبيّنًا الكيانات الرئيسية والعلاقات بينها لضمان تنظيم البيانات ودعم عمليات الفوترة والتسجيل بكفاءة.





8. الخلاصة

من خلال تطبيق أنماط مثل Singleton وObserver وFactory وStrategy وTemplate Method وFacade وDAO، تم تحقيق فصل واضح للمسؤوليات، وزيادة مرونة النظام، وتحسين تنظيم الكود البرمجي وقابليته للتوسعة.