UTILITY-BILLING-APP

43	Design and implement a console-based Utility Billing application to register consumers, record smart-meter readings, compute bills with slabs, and track payments using OOP in Java. Requirements: 1. Create at least 4 classes: ○ Consumer – consumerId, name, address, tariffPlan, status. ○ Meter – meterId, consumer, lastReading, lastReadingDate, health. ○ Bill – billNo, consumer, units, amount, dueDate, state. ○ UtilityService – reading capture, bill generation, payment posting, dunning. 2. Each class must include: ○ ≥4 instance/static variables. ○ A constructor to initialize values. ○ ≥5 methods (getters/setters, recordReading(), generateBill(), applySurcharge(), recordPayment()). 3. Demonstrate OOPS Concepts: ○ Inheritance → DomesticTariff/CommercialTariff extend a base TariffPlan (you may add this class). ○ Method Overloading → generateBill() by units only or units + peakHours flag. ○ Method Overriding → tariff-specific slab calculation in subclasses. ○ Polymorphism → compute charges from List<TariffPlan> dynamically. ○ Encapsulation → guard bill state transitions and meter readings. 4. Write a Main class (UtilityAppMain) to test: ○ Register consumers/meters, record readings. ○ Generate bills, post payments, apply late fees. ○ Print aging reports and revenue by tariff type.

CODE:

```
package javaproject;

import java.util.*;


//---------- Tariff Plans (Inheritance + Polymorphism) ----------

abstract class TariffPlan {

 protected String planName;


 public TariffPlan(String planName) {

    this.planName = planName;

 }
```

```
 public String getPlanName() { return planName; }

 // Overridden in subclasses
 public abstract double calculateCharges(int units, boolean
peakHours);
}

class DomesticTariff extends TariffPlan {
 public DomesticTariff() { super("Domestic"); }

 @Override
 public double calculateCharges(int units, boolean peakHours) {
    double amount = 0;
    if (units <= 100) amount = units * 3;
    else if (units <= 300) amount = 100 * 3 + (units - 100) * 4.5;
    else amount = 100 * 3 + 200 * 4.5 + (units - 300) * 6;
    if (peakHours) amount *= 1.10; // surcharge 10%
    return amount;
 }
}

class CommercialTariff extends TariffPlan {
 public CommercialTariff() { super("Commercial"); }
```

```
 @Override
 public double calculateCharges(int units, boolean peakHours) {
    double amount = units * 7;
    if (units > 500) amount += (units - 500) * 2; // extra slab
    if (peakHours) amount *= 1.15; // surcharge 15%
    return amount;
 }
}

//---------- Consumer ----------
class Consumer {
 private int consumerId;
 private String name;
 private String address;
 private TariffPlan tariffPlan;
 private String status;

 public Consumer(int consumerId, String name, String address,
TariffPlan tariffPlan) {
    this.consumerId = consumerId;
    this.name = name;
    this.address = address;
    this.tariffPlan = tariffPlan;
    this.status = "ACTIVE";
```

```
 }

 public int getConsumerId() { return consumerId; }
 public TariffPlan getTariffPlan() { return tariffPlan; }
 public String getName() { return name; }
 public String getStatus() { return status; }
 public void setStatus(String status) { this.status = status; }

 @Override
 public String toString() {
    return "Consumer{" + consumerId + ", " + name + ", " +
tariffPlan.getPlanName() + "}";
 }
}

//---------- Meter ----------
class Meter {
 private int meterId;
 private Consumer consumer;
 private int lastReading;
 private Date lastReadingDate;
 private String health;

 public Meter(int meterId, Consumer consumer) {
```

```
        this.meterId = meterId;
        this.consumer = consumer;
        this.lastReading = 0;
        this.lastReadingDate = new Date();
        this.health = "OK";
    }

    public int getMeterId() { return meterId; }
    public Consumer getConsumer() { return consumer; }
    public int getLastReading() { return lastReading; }

    public void recordReading(int newReading) {
        if (newReading < lastReading) {
            System.out.println("Invalid reading. Cannot be less than last
reading.");
            return;
        }
        lastReading = newReading;
        lastReadingDate = new Date();
    }

    @Override
    public String toString() {
```

```
    return "Meter{" + meterId + ", Last=" + lastReading + ",
Consumer=" + consumer.getName() + "}";
 }
}

//---------- Bill ----------
class Bill {
 private static int counter = 1;
 private int billNo;
 private Consumer consumer;
 private int units;
 private double amount;
 private Date dueDate;
 private String state; // GENERATED, PAID, OVERDUE

 public Bill(Consumer consumer, int units, double amount) {
    this.billNo = counter++;
    this.consumer = consumer;
    this.units = units;
    this.amount = amount;
    this.dueDate = new Date(System.currentTimeMillis() +
7L*24*60*60*1000); // +7 days
    this.state = "GENERATED";
 }
```

```
public int getBillNo() { return billNo; }
public Consumer getConsumer() { return consumer; }
public double getAmount() { return amount; }
public String getState() { return state; }

public void recordPayment(double payment) {
    if (state.equals("PAID")) {
        System.out.println("Bill already paid.");
        return;
    }
    if (payment >= amount) {
        state = "PAID";
        System.out.println("Payment successful for Bill#" + billNo);
    } else {
        System.out.println("Partial payment not allowed. Please pay
full.");
    }
}

public void applySurcharge() {
    if (state.equals("GENERATED") && new Date().after(dueDate)) {
        amount *= 1.05; // 5% late fee
        state = "OVERDUE";
```

```
    }
 }

 @Override
 public String toString() {
    return "Bill{" + billNo + ", Consumer=" + consumer.getName() +
          ", Units=" + units + ", Amount=" + amount +
          ", State=" + state + "}";
 }
}

//---------- Utility Service ----------
class UtilityService {
 private List<Consumer> consumers = new ArrayList<>();
 private List<Meter> meters = new ArrayList<>();
 private List<Bill> bills = new ArrayList<>();

 public Consumer registerConsumer(int id, String name, String addr,
TariffPlan plan) {
    Consumer c = new Consumer(id, name, addr, plan);
    consumers.add(c);
    Meter m = new Meter(id, c);
    meters.add(m);
    return c;
```

```
}

public void recordReading(int meterId, int newReading) {
    for (Meter m : meters) {
        if (m.getMeterId() == meterId) {
            m.recordReading(newReading);
            return;
        }
    }
}

// Overloaded generateBill
public void generateBill(int meterId, int newReading) {
    generateBill(meterId, newReading, false);
}
public void generateBill(int meterId, int newReading, boolean
peakHours) {
    for (Meter m : meters) {
        if (m.getMeterId() == meterId) {
            int consumed = newReading - m.getLastReading();
            m.recordReading(newReading);
            double amount =
m.getConsumer().getTariffPlan().calculateCharges(consumed,
peakHours);
            Bill b = new Bill(m.getConsumer(), consumed, amount);
```

```
            bills.add(b);
            System.out.println("Generated: " + b);
            return;
        }
    }
}

public void recordPayment(int billNo, double amount) {
    for (Bill b : bills) {
        if (b.getBillNo() == billNo) {
            b.recordPayment(amount);
            return;
        }
    }
}

public void applyDunning() {
    for (Bill b : bills) b.applySurcharge();
}

public void reportRevenue() {
    double domestic = 0, commercial = 0;
    for (Bill b : bills) {
```

```
        if (b.getConsumer().getTariffPlan() instanceof DomesticTariff)
domestic += b.getAmount();
        else commercial += b.getAmount();
    }
    System.out.println("Revenue Report: Domestic=" + domestic + ",
Commercial=" + commercial);
 }

 public void listBills() {
    for (Bill b : bills) System.out.println(b);
 }
}

//---------- Main ----------
public class UtilityAppMain {
 public static void main(String[] args) {
    UtilityService service = new UtilityService();

    // Register consumers
    Consumer c1 = service.registerConsumer(1, "Alice", "Street 1",
new DomesticTariff());
    Consumer c2 = service.registerConsumer(2, "BobCorp", "Market
Rd", new CommercialTariff());

    // Record readings & generate bills
```

```
    service.generateBill(1, 150); // Domestic
    service.generateBill(2, 600, true); // Commercial + peakHours

    // List bills
    service.listBills();

    // Record payments
    service.recordPayment(1, 525);  // Full payment
    service.recordPayment(2, 4200); // Full payment

    // Apply dunning (surcharge if overdue)
    service.applyDunning();

    // Show revenue report
    service.reportRevenue();
  }
}
```

OUTPUT:

```
Console
<terminated> UtilityAppMain [Java Application] C:\Users\mvign\Downloads\eclipse-jee-2025-06-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_2
Generated: Bill{1, Consumer=Alice, Units=150, Amount=525.0, State=GENERATED}
Generated: Bill{2, Consumer=BobCorp, Units=600, Amount=5060.0, State=GENERATED}
Bill{1, Consumer=Alice, Units=150, Amount=525.0, State=GENERATED}
Bill{2, Consumer=BobCorp, Units=600, Amount=5060.0, State=GENERATED}
Payment successful for Bill#1
Partial payment not allowed. Please pay full.
Revenue Report: Domestic=525.0, Commercial=5060.0
```

GitHub repository link:

https://github.com/RAGAVENTHIRAN243