Jaypee Institute of Information Technology

Sector-62, Noida

A

Project Report

Of

<u>Object Oriented Analysis and Design Using Java</u>

on

Payment Gateway System

Submitted By:

S. No.	Enrolment No.	Name	Batch
1	23103278	Kush Kansal	B10
2	23103298	Aaditya Pratap Singh	B10
3	23103303	Prakhar Singhal	B10
4	23103374	Vasu Tayal	B10

Under Supervision Of:

Mr. Shivendra Vikram Singh

ABSTRACT

This project implements a simplified Payment Gateway System in Java using Object-Oriented Programming, SOLID design principles, and inheritance hierarchies.

The system simulates different payment methods such as Card, UPI, Bank Transfer, and Net Banking, along with features like refund processing, recurring payments, transaction storage, file-based persistence for payers and recurring data, transaction history tracking, audit logging, configuration management, and validation.

The design demonstrates clean architecture using interfaces, dependency injection, factory pattern, facade pattern, and separation of responsibilities. The project focuses on building a scalable, easily extendable payment system where new payment methods can be added without modifying existing code. An in-memory transaction store with file persistence, unified service layer, validators, and enhanced reporting ensure structured and reliable payment handling.

OBJECTIVES

1. Design a modular payment gateway using Java following SOLID principles and inheritance.
2. Simulate multiple payment methods (Card, UPI, Bank Transfer, Net Banking).
3. Implement transaction recording using an in-memory store with file-based persistence for payers and recurring payments
4. Implement refund support for methods that allow refunds.
5. Ensure code extensibility using loose coupling, interfaces, and design patterns (Factory, Dependency Injection, Facade).
6. Apply OOP principles, inheritance, and design patterns to demonstrate polymorphism and abstraction.
7. Demonstrate error handling, validation, logging, audit trails, and configuration management in a realistic payment scenario.
8. Showcase how new payment methods can be added easily without changes in existing classes, including transaction history and detailed reporting.
9. Develop a team-based project emphasizing collaboration, testing stubs, and educational documentation

TOPICS OF JAVA USED IN THE PROJECT

1. Object-Oriented Programming (OOP)

The payment processing system is structured using key Object-Oriented Programming concepts to ensure a maintainable and scalable design.

- **Classes & Objects:** The core entities modelled as classes include **Money**, **Payer**, **PayReq** (Payment Request), **PayResp** (Payment Response), **Transaction**, and **RecurringTransaction**. These classes serve as the building blocks of the system.
- **Encapsulation:** Data security and integrity are maintained by implementing **private fields** for all critical data, with controlled access provided via **getters and setters** where appropriate.
- **Abstraction:** Complexity is managed using **interfaces**, which define contracts for different system functionalities. Key interfaces include **IPayment**, **IRefund**, **ITransactionStore**, **IValidator**, and **IRecurringPayment**.
- **Polymorphism:** The system supports various payment methods through a single, unified interface. For example, **CardPayment**, **UpiPayment**, **BankTransfer**, and **NetBanking** all implement the same **IPayment** interface, allowing them to be treated interchangeably by the core service.
- **Inheritance:** Inheritance is used sparingly, primarily where a clear *is-a* relationship exists. **RecurringTransaction extends Transaction**. For most other relationships, **composition** is preferred to maintain flexibility.

2. Principles (SOLID)

The system design adheres to the SOLID principles for robust and flexible architecture.

- **S — Single Responsibility Principle (SRP):** Each class is designed to have **one purpose**, preventing tightly coupled code. Examples include **IdGen** (ID generation), **PayReqValidator** (request validation), **FileTxStore** (transaction storage), **PayerManager** (payer persistence), **RecurringManager** (recurring data), **PaymentService** (orchestration), **Scheduler** (recurring scheduling), **ReportGenerator** (reporting), and **PaymentFacade** (simplified access).
- **O — Open/Closed Principle (OCP):** The system is structured to be **open for extension but closed for modification**. New payment methods (e.g., NetBanking) can be added simply by creating a new class that implements the **IPayment** interface, without altering existing core payment classes. New storage types can also be implemented via **ITransactionStore**.
- **L — Leskov Substitution Principle (LSP):** Subtypes must be substitutable for their base types. Any specific payment method (e.g., CardPayment) can replace another through the **IPayment** interface without breaking the system. Similarly, **FileTxStore** can replace an in-memory store like **MemTxStore**.
- **I — Interface Segregation Principle (ISP):** Clients should not be forced to depend on methods they do not use. Functionality is broken down into specific interfaces, such as separating **IPayment** and **IRefund**. The **IRecurringPayment** interface is specific to recurring logic.
- **D — Dependency Inversion Principle (DIP):** High-level modules should not depend on low-level modules; both should depend on abstractions. **PaymentService** depends on interfaces

like **ITransactionStore** and **IValidator**, not on concrete implementation classes. The **PaymentFacade** also relies on interfaces for its underlying services.

3. Java Collections Framework and I/O

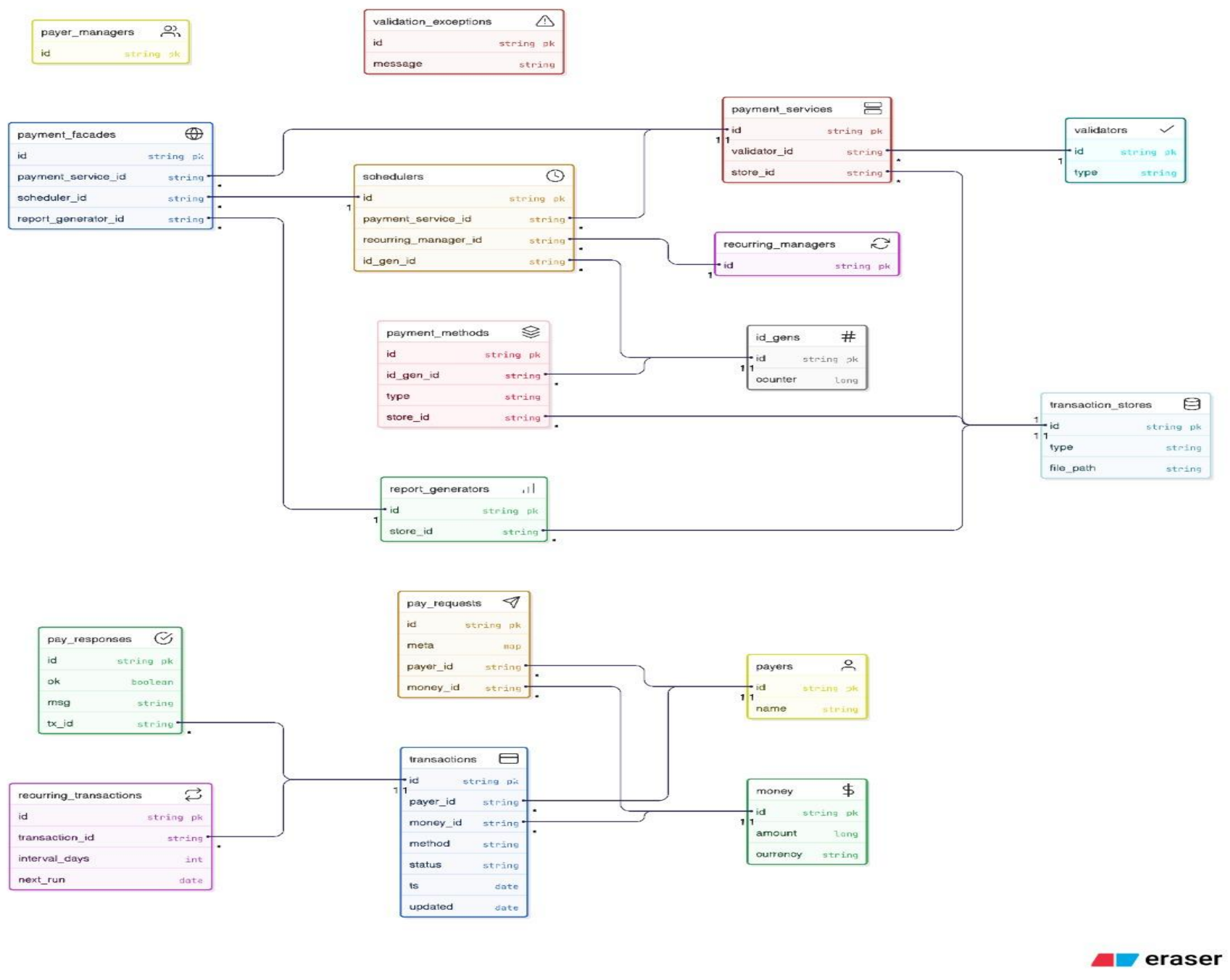
The system utilizes the Java Collections Framework for efficient data handling:

- **HashMap, Map, List, ArrayList, Collections.emptyMap()** are used for storing dynamic data such as in-memory transactions, lists of payers, and recurring payment schedules.

File Input/Output is used for persistence:

- **Print Writer, Buffered Reader, File Writer, File Reader** are employed for persisting critical data to files, including transactions (**transactions.txt**), payers (**payers.txt**), and recurring setup data (**recurring.txt**).

UML DIAGRAM



CODE IMPLEMENTATION OF MAIN CLASS

```java
import java.io.*;
import java.util.*;
import java.util.concurrent.atomic.AtomicLong;

class Money
{
    private final long amount; // in paise
    private final String currency;

    public Money(long amount, String currency)
    {
        this.amount = amount;
        this.currency = currency;
    }

    public long getAmount()
    {
        return amount;
    }

    public String getCurrency()
    {
        return currency;
    }

    public Money add(Money m) throws Exception
    {
        if (!m.currency.equals(this.currency))
        {
            throw new Exception("curr mismatch");
        }
        return new Money(this.amount + m.amount, currency);
    }

    public Money sub(Money m) throws Exception
    {
        if (!m.currency.equals(this.currency))
        {
            throw new Exception("curr mismatch");
        }
        return new Money(this.amount - m.amount, currency);
    }

    @Override
    public String toString()
    {
        return String.format("%s %.2f", currency, amount / 100.0);
    }
}

class Payer
{
    private final String id;
    private final String name;

    public Payer(String id, String name)
    {
        this.id = id;
        this.name = name;
    }

    public String getId()
    {
        return id;
    }

    public String getName()
    {
        return name;
    }
}

class PayReq
{
    private final Payer payer;
    private final Money amt;
    private final Map<String, String> meta;

    public PayReq(Payer p, Money amt, Map<String, String> meta)
    {
        this.payer = p;
        this.amt = amt;
        this.meta = meta == null ? Collections.emptyMap() : new
HashMap<>(meta);
    }

    public Payer getPayer()
    {
        return payer;
    }
    public Money getAmt()
    {
        return amt;
    }

    public Map<String, String> getMeta()
    {
        return meta;
    }
}

class PayResp
{
    private final String txId;
    private final boolean ok;
    private final String msg;

    public PayResp(String txId, boolean ok, String msg)
    {
        this.txId = txId;
        this.ok = ok;
        this.msg = msg;
    }

    public String getTxId()
    {
        return txId;
    }

    public boolean isOk()
    {
        return ok;
    }

    public String getMsg()
    {
        return msg;
    }

    @Override
    public String toString()
    {
        return "txId=" + txId + " ok=" + ok + " msg=" + msg;
    }
}
```

```
interface IPayment
{
    PayResp pay(PayReq req);
}

interface IRefund
{
    PayResp refund(String txId, Money amt);
}

interface ITransactionStore
{
    void save(Transaction t);

    Transaction find(String txId);

    List<Transaction> getAll();
}

// Transaction entity
class Transaction
{
    public enum Status
    {
        PENDING, SUCCESS, FAILED, REFUNDED
    }

    private final String id;
    private final String method;
    private final Payer payer;
    private final Money amt;
    private Status status;
    private final Date ts;
    private Date updated;

    public Transaction(String id, String method, Payer p, Money amt,
Status st)
    {
        this.id = id;
        this.method = method;
        this.payer = p;
        this.amt = amt;
        this.status = st;
        this.ts = new Date();
        this.updated = ts;
```

```java
}

public String getId()
{
    return id;
}

public String getMethod()
{
    return method;
}

public Payer getPayer()
{
    return payer;
}

public Money getAmt()
{
    return amt;
}

public Status getStatus()
{
    return status;
}

public void setStatus(Status s)
{
    this.status = s;
    this.updated = new Date();
}

@Override
public String toString()
{
    return "Transaction{"
            + "id='" + id + '\''
            + ", method='" + method + '\''
            + ", payer=" + payer.getName()
            + ", amt=" + amt
            + ", status=" + status
            + ", ts=" + ts
            + ", updated=" + updated
            + '}';
}
```

```java
    public String toCsv()
    {
        return id + "," + method + "," + payer.getId() + "," +
payer.getName() + "," + amt.getAmount() + "," + amt.getCurrency() + "," +
status;
    }

    public static Transaction fromCsv(String csv)
    {
        String[] parts = csv.split(",");
        if (parts.length != 7) throw new
IllegalArgumentException("Invalid CSV");
        return new Transaction(parts[0], parts[1], new Payer(parts[2],
parts[3]), new Money(Long.parseLong(parts[4]), parts[5]),
Transaction.Status.valueOf(parts[6]));
    }
}

class MemTxStore implements ITransactionStore
{
    private final Map<String, Transaction> map = new HashMap<>();

    @Override
    public void save(Transaction t)
    {
        map.put(t.getId(), t);
    }

    @Override
    public Transaction find(String txId)
    {
        return map.get(txId);
    }

    @Override
    public List<Transaction> getAll()
    {
        return new ArrayList<>(map.values());
    }
}

class FileTxStore implements ITransactionStore
{
    private final Map<String, Transaction> map = new HashMap<>();
    private final String filePath;
    public FileTxStore(String filePath)
    {
        this.filePath = filePath;
        loadFromFile();
    }

    private void loadFromFile()
    {
        try (BufferedReader br = new BufferedReader(new
FileReader(filePath)))
        {
            String line;
            while ((line = br.readLine()) != null)
            {
                if (!line.trim().isEmpty())
                {
                    Transaction t = Transaction.fromCsv(line);
                    map.put(t.getId(), t);
                }
            }
        }
        catch (IOException e)
        {
        }
    }

    @Override
    public void save(Transaction t)
    {
        map.put(t.getId(), t);
        appendToFile(t);
    }

    private void appendToFile(Transaction t)
    {
        try (FileWriter fw = new FileWriter(filePath, true))
        {
            fw.write(t.toCsv() + "\n");
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
```

```java
    @Override
    public Transaction find(String txId)
    {
        return map.get(txId);
    }

    @Override
    public List<Transaction> getAll()
    {
        return new ArrayList<>(map.values());
    }
}

// Unique id generator
class IdGen
{
    private final AtomicLong c = new AtomicLong(1000);

    public String next(String prefix)
    {
        return prefix + c.getAndIncrement();
    }
}

interface IValidator<T>
{
    void validate(T t) throws ValidationException;
}

class PayReqValidator implements IValidator<PayReq>
{
    @Override
    public void validate(PayReq r) throws ValidationException
    {
        if (r.getAmt() == null)
        {
            throw new ValidationException("Amount is null");
        }
        if (r.getAmt().getAmount() <= 0)
        {
            throw new ValidationException("Amount must be positive");
        }
        if (r.getPayer() == null)
        {
            throw new ValidationException("Payer is null");
        }
    }
}

class CardPayment implements IPayment, IRefund
{
    private final ITransactionStore store;
    private final IdGen idg;

    public CardPayment(ITransactionStore s, IdGen idg)
    {
        this.store = s;
        this.idg = idg;
    }

    @Override
    public PayResp pay(PayReq req)
    {
        String tx = idg.next("CARD-");
        Transaction t = new Transaction(tx, "CARD", req.getPayer(),
req.getAmt(), Transaction.Status.PENDING);
        store.save(t);
        boolean ok = true;
        if (ok)
        {
            t.setStatus(Transaction.Status.SUCCESS);
            store.save(t);
            return new PayResp(tx, true, "card success");
        }
        else
        {
            t.setStatus(Transaction.Status.FAILED);
            store.save(t);
            return new PayResp(tx, false, "card failed");
        }
    }

    @Override
    public PayResp refund(String txId, Money amt)
    {
        Transaction t = store.find(txId);
        if (t == null)
        {
            return new PayResp("", false, "tx not found");
        }
        if (!t.getMethod().equals("CARD"))
        {
            return new PayResp(txId, false, "method mismatch");
        }
        Transaction r = new Transaction(idg.next("R-"), "CARD-REFUND",
t.getPayer(), amt,
                Transaction.Status.REFUNDED);
        store.save(r);
        t.setStatus(Transaction.Status.REFUNDED);
        store.save(t);
        return new PayResp(r.getId(), true, "refund success");
    }
}

class UpiPayment implements IPayment, IRefund
{
    private final ITransactionStore s;
    private final IdGen g;

    public UpiPayment(ITransactionStore s, IdGen g)
    {
        this.s = s;
        this.g = g;
    }

    @Override
    public PayResp pay(PayReq r)
    {
        String tx = g.next("UPI-");
        Transaction t = new Transaction(tx, "UPI", r.getPayer(),
r.getAmt(), Transaction.Status.PENDING);
        s.save(t);
        boolean ok = sendToBank();
        if (ok)
        {
            t.setStatus(Transaction.Status.SUCCESS);
            s.save(t);
            return new PayResp(tx, true, "upi success");
        }
        else
        {
            t.setStatus(Transaction.Status.FAILED);
            s.save(t);
            return new PayResp(tx, false, "upi failed");
        }
    }

    @Override
    public PayResp refund(String txId, Money amt)
    {
        Transaction t = s.find(txId);
        if (t == null)
        {
            return new PayResp("", false, "tx not found");
        }
        if (!t.getMethod().equals("UPI"))
        {
            return new PayResp(txId, false, "method mismatch");
        }
        boolean ok = reverse();
        if (ok)
        {
            Transaction r = new Transaction(g.next("R-"), "UPI-REFUND",
t.getPayer(), amt, Transaction.Status.REFUNDED);
            s.save(r);
            t.setStatus(Transaction.Status.REFUNDED);
            s.save(t);
            return new PayResp(r.getId(), true, "refund success");
        }
        else
        {
            return new PayResp("", false, "refund failed");
        }
    }

    private boolean sendToBank()
    {
        return true;
    }

    private boolean reverse()
    {
        return true;
    }
}

class PaymentService
{
    private final Map<String, IPayment> payments = new HashMap<>();
    private final IValidator<PayReq> v;
    private final ITransactionStore s;

    public PaymentService(IValidator<PayReq> v, ITransactionStore s)
    {
```

```java
        this.v = v;
        this.s = s;
    }

    public void register(String key, IPayment p)
    {
        payments.put(key, p);
    }

    public PayResp execute(String method, PayReq req)
    {
        try
        {
            v.validate(req);
        }
        catch (ValidationException e)
        {
            return new PayResp("", false, e.getMessage());
        }
        IPayment p = payments.get(method);
        if (p == null)
        {
            return new PayResp("", false, "method unsupported");
        }
        return p.pay(req);
    }

    public PayResp refund(String method, String txId, Money amt)
    {
        IPayment p = payments.get(method);
        if (p instanceof IRefund ref)
        {
            return ref.refund(txId, amt);
        }
        return new PayResp("", false, "refund not supported");
    }

    public Transaction find(String txId)
    {
        return s.find(txId);
    }
}

class ValidationException extends Exception
{
    public ValidationException(String message)
    {
        super(message);
    }
}

class BankTransfer implements IPayment, IRefund
{
    private final ITransactionStore store;
    private final IdGen idg;

    public BankTransfer(ITransactionStore s, IdGen idg)
    {
        this.store = s;
        this.idg = idg;
    }

    @Override
    public PayResp pay(PayReq req)
    {
        String tx = idg.next("BT-");
        Transaction t = new Transaction(tx, "BANK_TRANSFER",
req.getPayer(), req.getAmt(), Transaction.Status.PENDING);
        store.save(t);
        boolean ok = simulateTransfer();
        if (ok)
        {
            t.setStatus(Transaction.Status.SUCCESS);
            store.save(t);
            return new PayResp(tx, true, "bank transfer success");
        }
        else
        {
            t.setStatus(Transaction.Status.FAILED);
            store.save(t);
            return new PayResp(tx, false, "bank transfer failed");
        }
    }

    @Override
    public PayResp refund(String txId, Money amt)
    {
        Transaction t = store.find(txId);
        if (t == null)
        {
            return new PayResp("", false, "tx not found");
        }
        if (!t.getMethod().equals("BANK_TRANSFER"))
        {
            return new PayResp(txId, false, "method mismatch");
        }
        Transaction r = new Transaction(idg.next("R-"), "BT-REFUND",
t.getPayer(), amt,
                Transaction.Status.REFUNDED);
        store.save(r);
        t.setStatus(Transaction.Status.REFUNDED);
        store.save(t);
        return new PayResp(r.getId(), true, "refund success");
    }

    private boolean simulateTransfer()
    {
        return true;
    }
}

interface IRecurringPayment
{
    void schedule(PayReq req, int intervalDays);

    void processRecurring();
}

class RecurringTransaction extends Transaction
{
    private final int intervalDays;
    private Date nextRun;

    public RecurringTransaction(String id, String method, Payer p, Money
amt, Status st, int interval)
    {
        super(id, method, p, amt, st);
        this.intervalDays = interval;
        this.nextRun = new Date(System.currentTimeMillis() + interval *
24 * 60 * 60 * 1000L);
    }

    public int getIntervalDays()
    {
        return intervalDays;
    }

    public Date getNextRun()
    {
        return nextRun;
    }

    public void updateNextRun()
    {
        this.nextRun = new Date(nextRun.getTime() + intervalDays * 24 *
60 * 60 * 1000L);
    }

    public void setNextRun(Date nextRun)
    {
        this.nextRun = nextRun;
    }
}

class RecurringManager
{
    public void saveRecurring(List<RecurringTransaction> recurring,
String filename)
    {
        try (PrintWriter pw = new PrintWriter(new FileWriter(filename)))
        {
            for (RecurringTransaction rt : recurring)
            {
                pw.println(rt.getId() + "," + rt.getPayer().getId() + ","
+ rt.getPayer().getName() + "," +
                        rt.getAmt().getAmount() + "," +
rt.getAmt().getCurrency() + "," + rt.getIntervalDays() + "," +
rt.getNextRun().getTime());
            }
        }
        catch (IOException e)
        {
            System.out.println("Error saving recurring: " +
e.getMessage());
        }
    }

    public List<RecurringTransaction> loadRecurring(String filename)
    {
        List<RecurringTransaction> recurring = new ArrayList<>();
        try (BufferedReader br = new BufferedReader(new
FileReader(filename)))
        {
            String line;
```

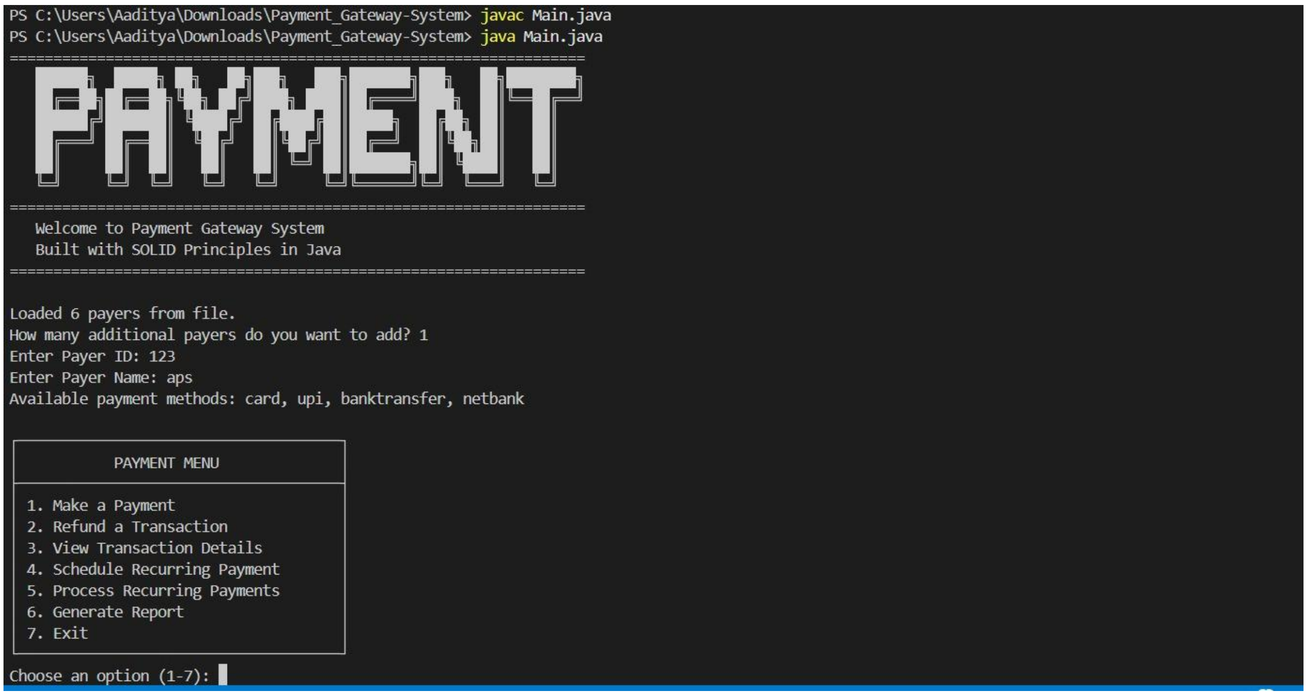
```java
            while ((line = br.readLine()) != null)
            {
                String[] parts = line.split(",");
                if (parts.length == 7)
                {
                    String id = parts[0];
                    String payerId = parts[1];
                    String payerName = parts[2];
                    long amount = Long.parseLong(parts[3]);
                    String currency = parts[4];
                    int interval = Integer.parseInt(parts[5]);
                    long nextRunTime = Long.parseLong(parts[6]);
                    Payer payer = new Payer(payerId, payerName);
                    Money amt = new Money(amount, currency);
                    RecurringTransaction rt = new
RecurringTransaction(id, "RECURRING", payer, amt,
Transaction.Status.PENDING, interval);
                    rt.setNextRun(new Date(nextRunTime));
                    recurring.add(rt);
                }
            }
        }
        catch (IOException e)
        {
            // File not found, start empty
        }
        return recurring;
    }
}

class Scheduler implements IRecurringPayment
{
    private final List<RecurringTransaction> recurring = new
ArrayList<>();
    private final PaymentService svc;
    private final IdGen idg;
    private final RecurringManager rm;

    public Scheduler(PaymentService svc, IdGen idg)
    {
        this.svc = svc;
        this.idg = idg;
        this.rm = new RecurringManager();
        recurring.addAll(rm.loadRecurring("recurring.txt"));
    }
    @Override
    public void schedule(PayReq req, int intervalDays)
    {
        String tx = idg.next("REC-");
        RecurringTransaction rt = new RecurringTransaction(tx,
"RECURRING", req.getPayer(), req.getAmt(),
                Transaction.Status.PENDING, intervalDays);
        recurring.add(rt);
        rm.saveRecurring(recurring, "recurring.txt");
    }

    @Override
    public void processRecurring()
    {
        Date now = new Date();
        for (RecurringTransaction rt : new ArrayList<>(recurring))
        {
            if (rt.getNextRun().before(now) ||
rt.getNextRun().equals(now))
            {
                PayReq pr = new PayReq(rt.getPayer(), rt.getAmt(),
Collections.emptyMap());
                PayResp resp = svc.execute("card", pr); // assume card
for recurring
                if (resp.isOk())
                {
                    rt.updateNextRun();
                    rm.saveRecurring(recurring, "recurring.txt");
                }
                else
                {
                    // remove if failed
                    recurring.remove(rt);
                    rm.saveRecurring(recurring, "recurring.txt");
                }
            }
        }
    }
}

class ReportGenerator
{
    private final ITransactionStore store;

    public ReportGenerator(ITransactionStore store)
    {
        this.store = store;
    }

    public void generateReport()
    {
        List<Transaction> all = store.getAll();
        System.out.println("========================================");
        System.out.println("         Transaction Report");
        System.out.println("========================================");
        System.out.println();
        if (all.isEmpty())
        {
            System.out.println("No transactions found.");
            return;
        }
        System.out.println("+------------+--------------+-------+--------
+----------+");
        System.out.println("| TxID       | Method       | Payer | Amount 
| Status   |");
        System.out.println("+------------+--------------+-------+--------
+----------+");
        int success = 0, failed = 0, pending = 0, refunded = 0;
        for (Transaction t : all)
        {
            String status = t.getStatus().toString();
            switch (t.getStatus())
            {
                case SUCCESS -> success++;
                case FAILED -> failed++;
                case PENDING -> pending++;
                case REFUNDED -> refunded++;
            }
            System.out.printf("| %-10s | %-12s | %-5s | %-6s | %-8s |\n",
                    t.getId(), t.getMethod(), t.getPayer().getName(),
t.getAmt().toString(), status);
        }
        System.out.println("+------------+--------------+-------+--------
+----------+");
        System.out.println();
        System.out.println("Summary:");
        System.out.println("Total Transactions: " + all.size());
        System.out.println("Successful: " + success);
        System.out.println("Failed: " + failed);
        System.out.println("Pending: " + pending);
        System.out.println("Refunded: " + refunded);
        System.out.println();
        System.out.println("Report generated at " + new Date());
    }
}

class PaymentFacade
{
    private final PaymentService svc;
    private final Scheduler sch;
    private final ReportGenerator rg;

    public PaymentFacade(PaymentService svc, Scheduler sch,
ReportGenerator rg)
    {
        this.svc = svc;
        this.sch = sch;
        this.rg = rg;
    }

    public PayResp pay(String method, Payer payer, long amount, String
currency)
    {
        Money m = new Money(amount, currency);
        PayReq req = new PayReq(payer, m, Collections.emptyMap());
        return svc.execute(method, req);
    }

    public void scheduleRecurring(Payer payer, long amount, String
currency, int interval)
    {
        Money m = new Money(amount, currency);
        PayReq req = new PayReq(payer, m, Collections.emptyMap());
        sch.schedule(req, interval);
    }

    public void processRecurring()
    {
        sch.processRecurring();
    }

    public void generateReport()
    {
        rg.generateReport();
    }

    public PayResp refund(String method, String txId, Money amt)
    {
```

```java
        return svc.refund(method, txId, amt);
    }

    public Transaction find(String txId)
    {
        return svc.find(txId);
    }
}

class PayerManager
{
    public void savePayers(List<Payer> payers, String filename)
    {
        try (PrintWriter pw = new PrintWriter(new FileWriter(filename)))
        {
            for (Payer p : payers)
            {
                pw.println(p.getId() + "," + p.getName());
            }
        }
        catch (IOException e)
        {
            System.out.println("Error saving payers: " + e.getMessage());
        }
    }

    public List<Payer> loadPayers(String filename)
    {
        List<Payer> payers = new ArrayList<>();
        try (BufferedReader br = new BufferedReader(new
FileReader(filename)))
        {
            String line;
            while ((line = br.readLine()) != null)
            {
                String[] parts = line.split(",");
                if (parts.length == 2)
                {
                    payers.add(new Payer(parts[0], parts[1]));
                }
            }
        }
        catch (IOException e)
        {
            // File not found or error, start with empty list
        }
        return payers;
    }
}

// Demo main
public class Main
{
    private static PaymentFacade setupPayments()
    {
        ITransactionStore store = new FileTxStore("transactions.txt");
        IdGen idg = new IdGen();

        CardPayment card = new CardPayment(store, idg);
        UpiPayment upi = new UpiPayment(store, idg);
        BankTransfer bt = new BankTransfer(store, idg);

        PayReqValidator val = new PayReqValidator();
        PaymentService svc = new PaymentService(val, store);

        svc.register("card", card);
        svc.register("upi", upi);
        svc.register("banktransfer", bt);

        Scheduler sch = new Scheduler(svc, idg);
        ReportGenerator rg = new ReportGenerator(store);
        PaymentFacade facade = new PaymentFacade(svc, sch, rg);

        // Add NetBanking
        IPayment nb = new IPayment()
        {
            private final ITransactionStore s = store;
            private final IdGen g2 = idg;

            @Override
            public PayResp pay(PayReq req)
            {
                String tx = g2.next("NB-");
                Transaction t = new Transaction(tx, "NETBANK",
req.getPayer(), req.getAmt(),
                        Transaction.Status.PENDING);
                s.save(t);
                t.setStatus(Transaction.Status.SUCCESS);
                s.save(t);
                return new PayResp(tx, true, "netbank success");
            }
        };
        svc.register("netbank", nb);

        return facade;
    }

    private static List<Payer> managePayers(Scanner sc)
    {
        PayerManager pm = new PayerManager();
        List<Payer> payers = pm.loadPayers("payers.txt");

        System.out.println("Loaded " + payers.size() + " payers from
file.");
        System.out.print("How many additional payers do you want to add?
");
        int numPayers = sc.nextInt();
        sc.nextLine();

        for (int i = 0; i < numPayers; i++)
        {
            System.out.print("Enter Payer ID: ");
            String id = sc.nextLine();
            System.out.print("Enter Payer Name: ");
            String name = sc.nextLine();
            Payer p = new Payer(id, name);
            payers.add(p);
        }

        pm.savePayers(payers, "payers.txt");

        return payers;
    }

    private static void runDemo(PaymentFacade facade, List<Payer> payers,
Scanner sc)
    {
        System.out.println("Available payment methods: card, upi,
banktransfer, netbank");
        System.out.println();

        while (true)
        {

System.out.println("┌──────────────────────────────────────┐");
            System.out.println("│           PAYMENT MENU
│");
```

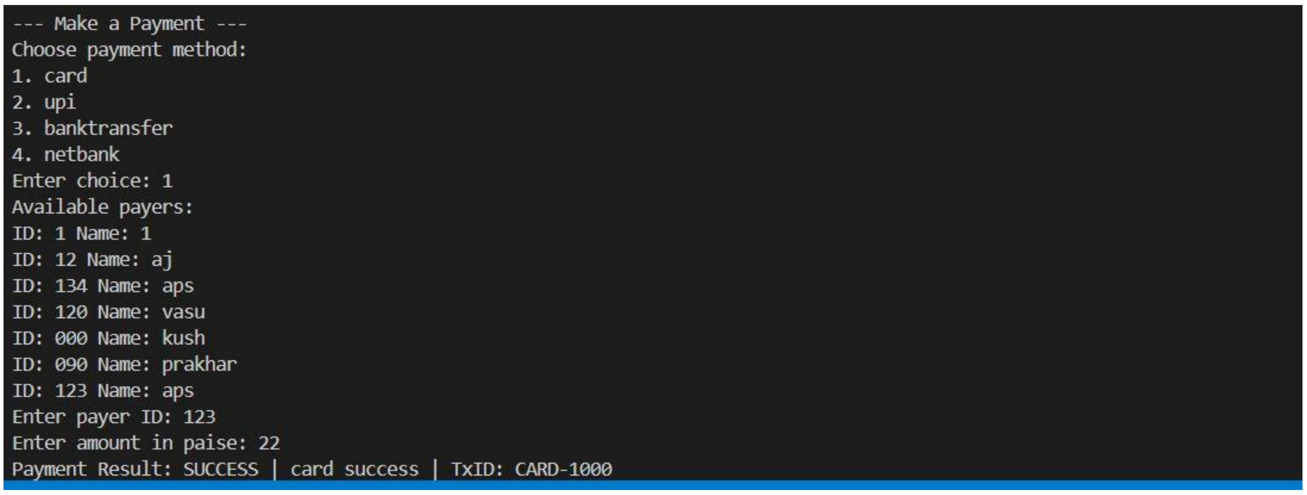
```
System.out.println("├────────────────────────────────────────┤");
            System.out.println("│ 1. Make a Payment
│");
            System.out.println("│ 2. Refund a Transaction
│");
            System.out.println("│ 3. View Transaction Details
│");
            System.out.println("│ 4. Schedule Recurring Payment
│");
            System.out.println("│ 5. Process Recurring Payments
│");
            System.out.println("│ 6. Generate Report
│");
            System.out.println("│ 7. Exit
│");

System.out.println("└────────────────────────────────────────┘");
            System.out.print("Choose an option (1-7): ");
            int choice = sc.nextInt();
            sc.nextLine();

            switch (choice)
            {
                case 1 ->
                {
                    System.out.println("\n--- Make a Payment ---");
                    System.out.println("Choose payment method:");
                    System.out.println("1. card");
                    System.out.println("2. upi");
                    System.out.println("3. banktransfer");
                    System.out.println("4. netbank");
                    System.out.print("Enter choice: ");
                    int methodChoice = sc.nextInt();
                    sc.nextLine();
                    String method;
                    switch (methodChoice)
                    {
                        case 1 -> method = "card";
                        case 2 -> method = "upi";
                        case 3 -> method = "banktransfer";
                        case 4 -> method = "netbank";
                        default ->
                        {
                            System.out.println("Invalid payment method
choice.");
                            continue;
                        }
                    }
                    System.out.println("Available payers:");
                    for (Payer p : payers)
                    {
                        System.out.println("ID: " + p.getId() + " Name: "
+ p.getName());
                    }
                    System.out.print("Enter payer ID: ");
                    String pid = sc.nextLine();
                    Payer payer = null;
                    for (Payer p : payers)
                    {
                        if (p.getId().equals(pid))
                        {
                            payer = p;
                            break;
                        }
                    }
                    if (payer == null)
                    {
                        System.out.println("Invalid payer ID.");
                        continue;
                    }
                    System.out.print("Enter amount in paise: ");
                    long amount = sc.nextLong();
                    sc.nextLine();
                    PayResp resp = facade.pay(method, payer, amount,
"INR");
                    System.out.println("Payment Result: " + (resp.isOk()
? "SUCCESS" : "FAILED") + " | "
                            + resp.getMsg() + " | TxID: " +
resp.getTxId());
                }
                case 2 ->
                {
                    System.out.println("\n--- Refund a Transaction ---");
                    System.out.println("Choose payment method for
refund:");
                    System.out.println("1. card");
                    System.out.println("2. upi");
                    System.out.println("3. banktransfer");
                    System.out.print("Enter choice: ");
                    int refMethodChoice = sc.nextInt();
                    sc.nextLine();
                    String refMethod;
                    switch (refMethodChoice)
                    {
                        case 1 -> refMethod = "card";
                        case 2 -> refMethod = "upi";
                        case 3 -> refMethod = "banktransfer";
                        default ->
                        {
                            System.out.println("Invalid refund method
choice.");
                            continue;
                        }
                    }
                    System.out.print("Enter transaction ID to refund: ");
                    String txId = sc.nextLine();
                    System.out.print("Enter refund amount in paise: ");
                    long refAmt = sc.nextLong();
                    sc.nextLine();
                    PayResp refResp = facade.refund(refMethod, txId, new
Money(refAmt, "INR"));
                    System.out.println(
                            "Refund Result: " + (refResp.isOk() ?
"SUCCESS" : "FAILED") + " | " + refResp.getMsg());
                }
                case 3 ->
                {
                    System.out.println("\n--- View Transaction Details --
-");
                    System.out.print("Enter transaction ID: ");
                    String viewTxId = sc.nextLine();
                    Transaction tx = facade.find(viewTxId);
                    if (tx != null)
                    {
                        System.out.println("Transaction: " + tx);
                    }
                    else
                    {
                        System.out.println("Transaction not found.");
                    }
                }
                case 4 ->
                {
                    System.out.println("\n--- Schedule Recurring Payment
---");
                    System.out.println("Available payers:");
                    for (Payer p : payers)
                    {
                        System.out.println("ID: " + p.getId() + " Name: "
+ p.getName());
                    }
                    System.out.print("Enter payer ID: ");
                    String recPid = sc.nextLine();
                    Payer recPayer = null;
                    for (Payer p : payers)
                    {
                        if (p.getId().equals(recPid))
                        {
                            recPayer = p;
                            break;
                        }
                    }
                    if (recPayer == null)
                    {
                        System.out.println("Invalid payer ID.");
                        continue;
                    }
                    System.out.print("Enter amount in paise: ");
                    long recAmt = sc.nextLong();
                    System.out.print("Enter interval in days: ");
                    int interval = sc.nextInt();
                    sc.nextLine();
                    facade.scheduleRecurring(recPayer, recAmt, "INR",
interval);
                    System.out.println("Recurring payment scheduled.");
                }
                case 5 ->
                {
                    System.out.println("\n--- Process Recurring Payments
---");
                    facade.processRecurring();
                    System.out.println("Recurring payments processed.");
                }
                case 6 ->
                {
                    System.out.println("\n--- Generate Report ---");
                    facade.generateReport();
                    System.out.println("Report generated.");
                }
                case 7 ->
                {
                    System.out.println("\n--- Exit ---");
                    System.out.println("Exiting...");
                    sc.close();
```

```java
                        return;
                    }
                    default ->
                    {
                        System.out.println("Invalid choice. Try again.");
                    }
                }
                System.out.println();
            }
        }

        public static void main(String[] args)
        {
            Scanner sc = new Scanner(System.in);

    System.out.println("==================================================================")
    ;
            System.out.println("   ██████╗  █████╗ ██╗   ██╗███╗   ███╗███████╗███╗   
    ██╗████████╗");
            System.out.println("   ██╔══██╗██╔══██╗╚██╗ ██╔╝████╗ ████║██╔════╝████╗  
    ██║╚══██╔══╝");
            System.out.println("   ██████╔╝███████║ ╚████╔╝ ██╔████╔██║█████╗  ██╔██╗ ██║   
    ██║   ");
            System.out.println("   ██╔═══╝ ██╔══██║  ╚██╔╝  ██║╚██╔╝██║██╔══╝  ██║╚██╗██║   
    ██║   ");
            System.out.println("   ██║     ██║  ██║   ██║   ██║ ╚═╝ ██║███████╗██║ ╚████║   
    ██║   ");
            System.out.println("   ╚═╝     ╚═╝  ╚═╝   ╚═╝   ╚═╝     ╚═╝╚══════╝╚═╝  ╚═══╝   
    ╚═╝   ");

    System.out.println("=====================================================
    =============");
            System.out.println("   Welcome to Payment Gateway System");
            System.out.println("   Built with SOLID Principles in Java");

    System.out.println("=====================================================
    =============");
            System.out.println();

            PaymentFacade facade = setupPayments();
            List<Payer> payers = managePayers(sc);
            runDemo(facade, payers, sc);
        }
    }
```

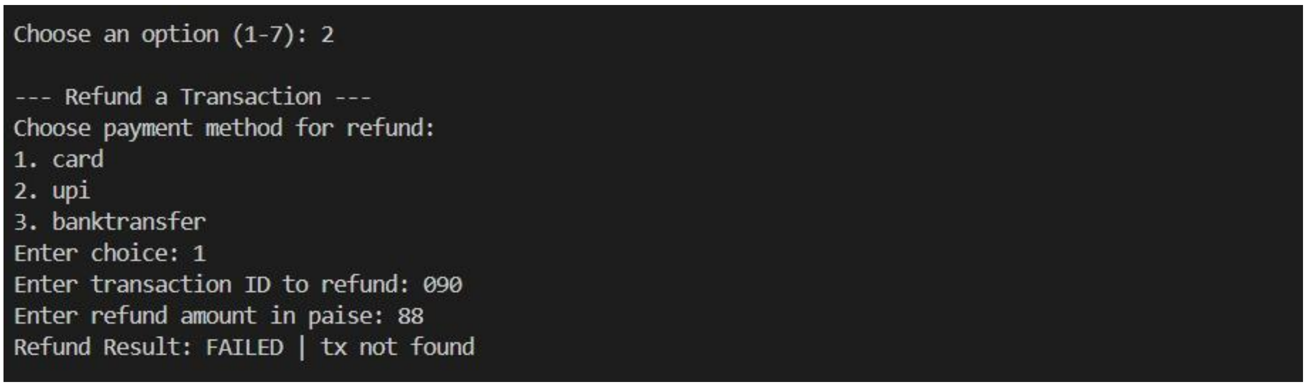
CODE OUTPUT:



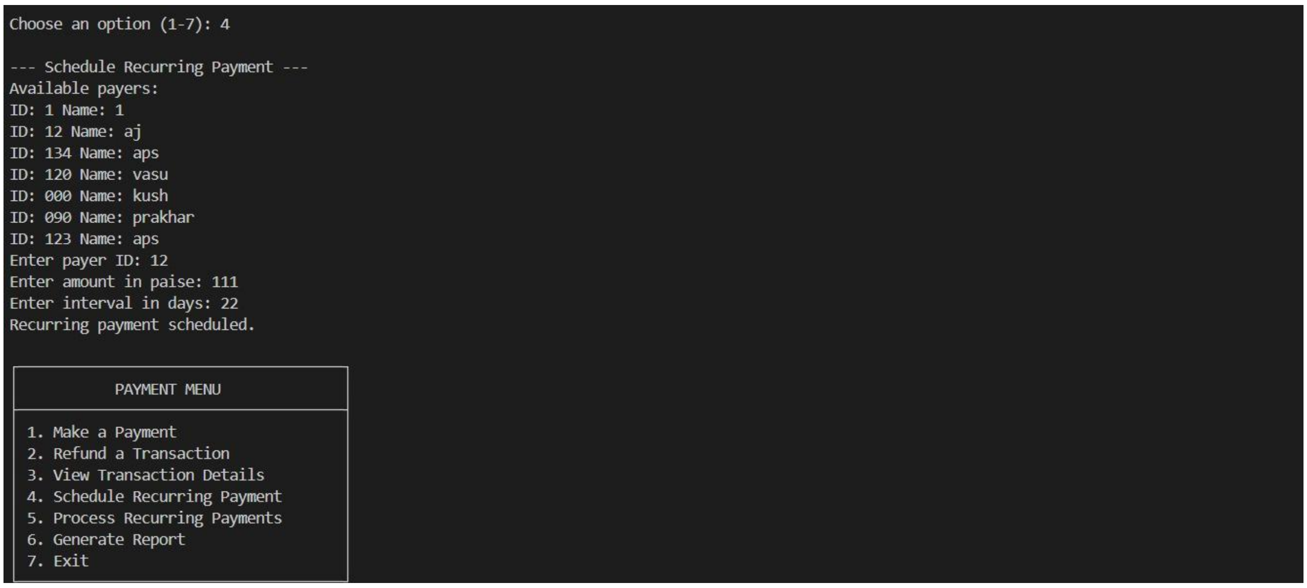
```
PS C:\Users\Aaditya\Downloads\Payment_Gateway-System> javac Main.java
PS C:\Users\Aaditya\Downloads\Payment_Gateway-System> java Main.java
==================================================================
 PAYMENT
==================================================================
   Welcome to Payment Gateway System
   Built with SOLID Principles in Java
==================================================================

Loaded 6 payers from file.
How many additional payers do you want to add? 1
Enter Payer ID: 123
Enter Payer Name: aps
Available payment methods: card, upi, banktransfer, netbank

         PAYMENT MENU

 1. Make a Payment
 2. Refund a Transaction
 3. View Transaction Details
 4. Schedule Recurring Payment
 5. Process Recurring Payments
 6. Generate Report
 7. Exit

Choose an option (1-7):
```



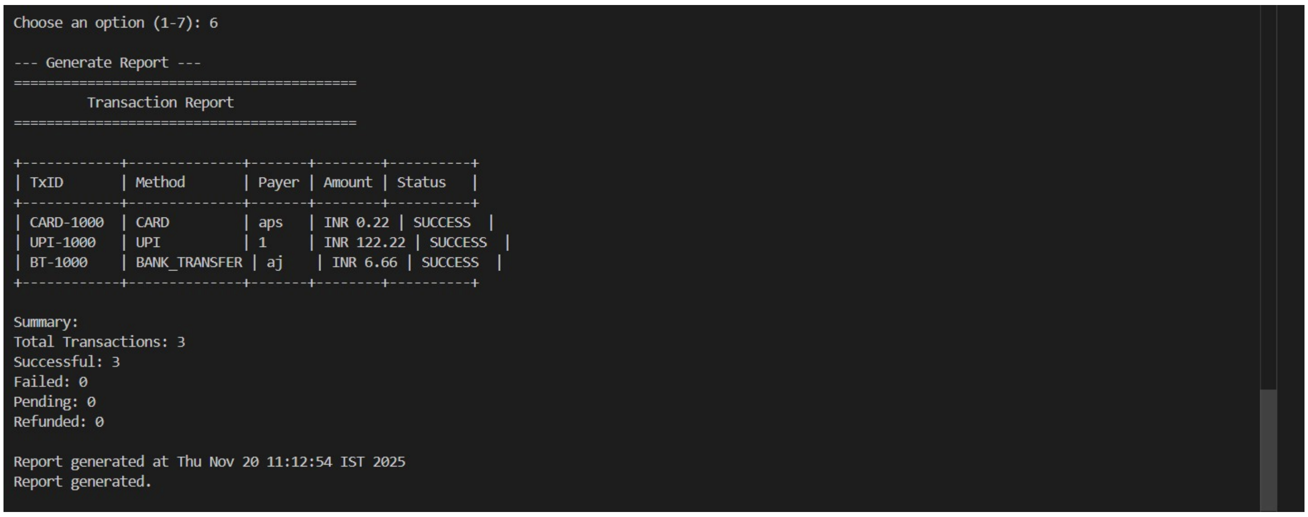
```
--- Make a Payment ---
Choose payment method:
1. card
2. upi
3. banktransfer
4. netbank
Enter choice: 1
Available payers:
ID: 1 Name: 1
ID: 12 Name: aj
ID: 134 Name: aps
ID: 120 Name: vasu
ID: 000 Name: kush
ID: 090 Name: prakhar
ID: 123 Name: aps
Enter payer ID: 123
Enter amount in paise: 22
Payment Result: SUCCESS | card success | TxID: CARD-1000
```



```
Choose an option (1-7): 2

--- Refund a Transaction ---
Choose payment method for refund:
1. card
2. upi
3. banktransfer
Enter choice: 1
Enter transaction ID to refund: 090
Enter refund amount in paise: 88
Refund Result: FAILED | tx not found
```



```
Choose an option (1-7): 4

--- Schedule Recurring Payment ---
Available payers:
ID: 1 Name: 1
ID: 12 Name: aj
ID: 134 Name: aps
ID: 120 Name: vasu
ID: 000 Name: kush
ID: 090 Name: prakhar
ID: 123 Name: aps
Enter payer ID: 12
Enter amount in paise: 111
Enter interval in days: 22
Recurring payment scheduled.

┌─────────────────────────────────────┐
│            PAYMENT MENU             │
├─────────────────────────────────────┤
│ 1. Make a Payment                   │
│ 2. Refund a Transaction             │
│ 3. View Transaction Details         │
│ 4. Schedule Recurring Payment       │
│ 5. Process Recurring Payments       │
│ 6. Generate Report                  │
│ 7. Exit                             │
```



```
Choose an option (1-7): 6

--- Generate Report ---
==========================================
        Transaction Report
==========================================

+-----------+-------------+-------+--------+----------+
| TxID      | Method      | Payer | Amount | Status   |
+-----------+-------------+-------+--------+----------+
| CARD-1000 | CARD        | aps   | INR 0.22 | SUCCESS  |
| UPI-1000  | UPI         | 1     | INR 122.22 | SUCCESS  |
| BT-1000   | BANK_TRANSFER | aj    | INR 6.66 | SUCCESS  |
+-----------+-------------+-------+--------+----------+

Summary:
Total Transactions: 3
Successful: 3
Failed: 0
Pending: 0
Refunded: 0

Report generated at Thu Nov 20 11:12:54 IST 2025
Report generated.
```

CONCLUSION

This Java project successfully demonstrates a scalable and extensible **Payment Gateway System** following best software engineering practices. By applying OOP, SOLID principles, and design patterns, the system becomes modular, testable, readable, and easy to extend with new payment methods or rules.

The project highlights how real-world payment systems can be simulated with transaction management, validations, refunds, recurring payments, balance checks, and file based persistent storage. The architecture allows future enhancements such as integrating a database, adding notifications, or extending payment types.

Overall, this project reflects strong understanding of Java design principles and shows how to build enterprise-level backend systems with clean code architecture.