SENG 350

Arfaz Hussain / V00984826

Case Study: Online Banking System Using Patterns

```
src/
├── models/
│   ├── Transaction.ts
│   └── TransactionValidator.ts
├── validators/
│   ├── AuthenticationCheckValidator.ts
│   ├── BalanceCheckValidator.ts
│   ├── FraudDetectionValidator.ts
│   └── ComplianceValidator.ts
├── transactions/
│   ├── FundTransfer.ts
│   ├── BillPayment.ts
│   └── LoanPayment.ts
├── factories/
│   └── TransactionFactory.ts
├── TransactionProcessor.ts
└── index.ts
```

1. Implementations

```typescript
export abstract class Transaction {
    abstract execute(): string;
}
```

```typescript
import { Transaction } from './Transaction';

export abstract class TransactionValidator {
    protected nextValidator: TransactionValidator | null = null;

    setNext(validator: TransactionValidator): TransactionValidator {
        this.nextValidator = validator;
        return validator;
    }

    abstract validate(transaction: Transaction): boolean;
}
```

```typescript
import { Transaction } from '../models/Transaction';

export class FundTransfer extends Transaction {
```

```
    execute(): string {
        return 'Fund Transfer executed';
    }
}
```

```
import { Transaction } from '../models/Transaction';

export class BillPayment extends Transaction {
    execute(): string {
        return 'Bill Payment executed';
    }
}
```

```
import { TransactionValidator } from '../models/TransactionValidator';
import { Transaction } from '../models/Transaction';

export class AuthenticationValidator extends TransactionValidator {
    validate(transaction: Transaction): boolean {
        console.log('Authentication validation passed');
        return this.nextValidator ? this.nextValidator.validate(transaction) :
true;
    }
}
```

```
import { TransactionValidator } from '../models/TransactionValidator';
import { Transaction } from '../models/Transaction';

export class BalanceValidator extends TransactionValidator {
    validate(transaction: Transaction): boolean {
        console.log('Balance validation passed');
        return this.nextValidator ? this.nextValidator.validate(transaction) :
true;
    }
}
```

```
import { Transaction } from '../models/Transaction';
import { FundTransfer } from '../transactions/FundTransfer';
import { BillPayment } from '../transactions/BillPayment';

export class TransactionFactory {
    createTransaction(type: string): Transaction {
        switch (type.toLowerCase()) {
            case 'fund transfer':
                return new FundTransfer();
            case 'bill payment':
                return new BillPayment();
            default:
                throw new Error(`Invalid transaction type: ${type}`);
        }
    }
```

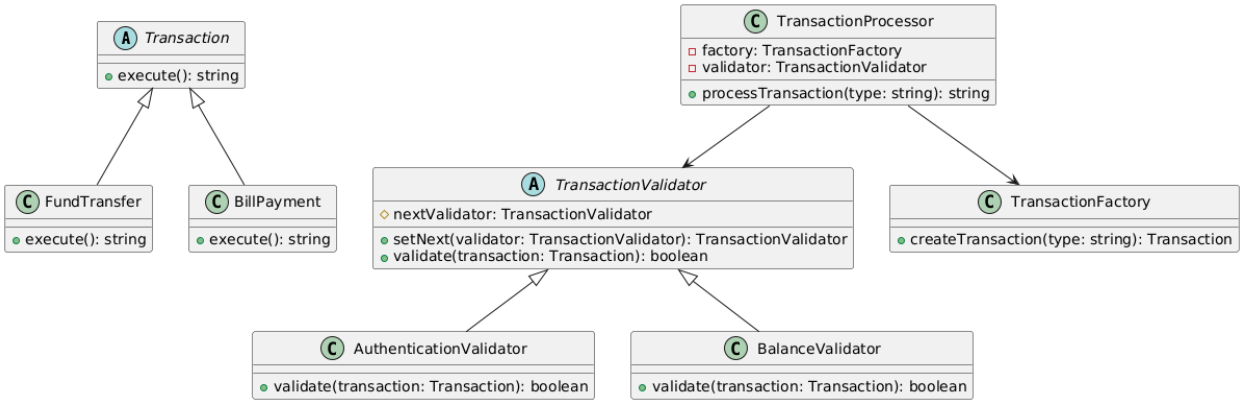
```
}
```

```
import { TransactionFactory } from './factories/TransactionFactory';
import { TransactionValidator } from './models/TransactionValidator';
import { AuthenticationValidator } from './validators/AuthenticationValidator';
import { BalanceValidator } from './validators/BalanceValidator';

export class TransactionProcessor {
    private factory: TransactionFactory;
    private validator: TransactionValidator;

    constructor() {
        this.factory = new TransactionFactory();
        this.validator = new AuthenticationValidator();
        this.validator.setNext(new BalanceValidator());
    }

    processTransaction(type: string): string {
        const transaction = this.factory.createTransaction(type);
        if (this.validator.validate(transaction)) {
            return transaction.execute();
        }
        return 'Transaction failed validation';
    }
}
```

```
import { useState } from 'react';
import { TransactionProcessor } from '../src/TransactionProcessor';

export default function Home() {
    const [transactionType, setTransactionType] = useState('');
    const [result, setResult] = useState('');
    const processor = new TransactionProcessor();

    const handleSubmit = (e: React.FormEvent) => {
        e.preventDefault();
        const processResult = processor.processTransaction(transactionType);
        setResult(processResult);
    };

    return (
        <div>
            <h1>Online Banking System</h1>
            <form onSubmit={handleSubmit}>
                <select
                    value={transactionType}
                    onChange={(e) => setTransactionType(e.target.value)}
                >
                    <option value="">Select transaction type</option>
                    <option value="fund transfer">Fund Transfer</option>
```

```
                    <option value="bill payment">Bill Payment</option>
                </select>
                <button type="submit">Process Transaction</button>
            </form>
            {result && <p>{result}</p>}
        </div>
    );
}
```

2. UML Diagram:



PlantUML for the above design:

```
@startuml
abstract class Transaction {
  +execute(): string
}

abstract class TransactionValidator {
  #nextValidator: TransactionValidator
  +setNext(validator: TransactionValidator): TransactionValidator
  +validate(transaction: Transaction): boolean
}

class FundTransfer {
  +execute(): string
}

class BillPayment {
  +execute(): string
}

class AuthenticationValidator {
  +validate(transaction: Transaction): boolean
}

class BalanceValidator {
  +validate(transaction: Transaction): boolean
}
```

```
class TransactionFactory {
  +createTransaction(type: string): Transaction
}

class TransactionProcessor {
  -factory: TransactionFactory
  -validator: TransactionValidator
  +processTransaction(type: string): string
}

Transaction <|-- FundTransfer
Transaction <|-- BillPayment
TransactionValidator <|-- AuthenticationValidator
TransactionValidator <|-- BalanceValidator
TransactionProcessor --> TransactionFactory
TransactionProcessor --> TransactionValidator
@enduml
```