Tentamen i Objektorienterad modellering och design – Helsingborg

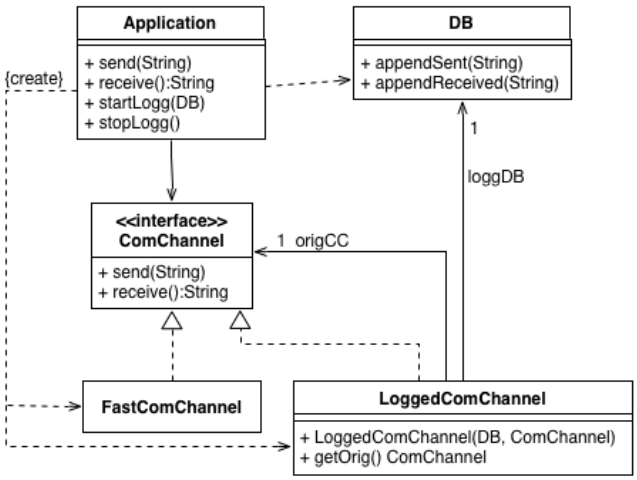
Lösningar

1. B, A, B, D

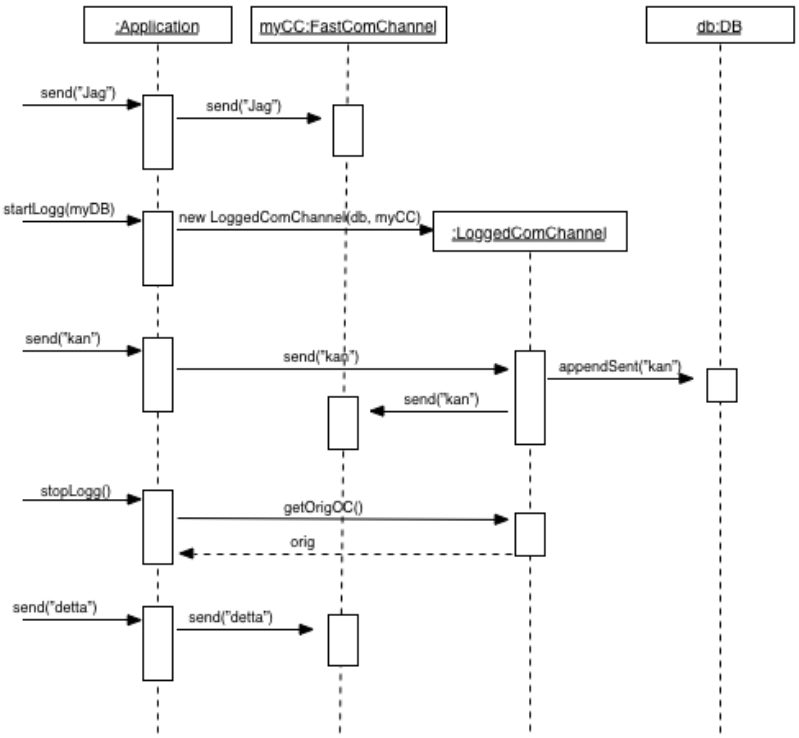
2. Template Method

3. **a**. Decorator

b. Klassdiagram



c. Sekvensdiagram



4.

```
class Stock extends Observable {

    private double value;

    void update(double value) {
        this.value = value;
        setChanged();
        notifyObservers();
    }

    public double value() {
        return value;
    }
}

class Portfolio extends Observable {

    private List<Stock> stocks = new LinkedList<>();
    private double buyValue;
    private double yieldLimit;

    public Portfolio  (double yieldLimit) {
        this.yieldLimit = yieldLimit;
    }

    public void add(Stock stock) {
        stocks.add(stock);
        buyValue += stock.value();
        stock.addObserver((obs,obj) -> {
                check();
            });
    }

    public double currentYield() {
        double value = stocks
            .stream()
            .mapToDouble(s -> s.value())
            .sum();
        return value / buyValue;
    }

    private void check() {
        if (currentYield() < yieldLimit) {
            setChanged();
            notifyObservers();
        }
    }
}
```

5.

```
interface DrawCommand {

    void draw(Drawing d);
    void undo();
}

interface Drawing {

    void clear();
    void setForegroundColor();
    void setBackgroundColor();
    void moveTo(int x, int y);
    void circle(int radius);
    void rect(int width, int height);
}

abstract class DrawShape implements DrawCommand {

    protected int x, y;
    protected Drawing d;

    public DrawShape  (int x, int y) {
        this.x = x;
```

```
        this.y = y;
    }

    protected abstract void shape(Drawing d);

    public void draw(Drawing d) {
        this.d = d;
        d.setForegroundColor();
        draw();
    }

    public void undo() {
        d.setBackgroundColor();
        draw();
    }

    private void draw() {
        d.moveTo(x, y);
        shape(d);
    }
}

class DrawCircle extends DrawShape {

    private int radius;

    public DrawCircle  (int x, int y, int radius) {
        super(x, y);
        this.radius = radius;
    }

    protected void shape(Drawing d) {
        d.circle(radius);
    }
}

class DrawSquare extends DrawShape {

    private int side;

    public DrawSquare  (int x, int y, int side) {
        super(x, y);
        this.side = side;
    }

    protected void shape(Drawing d) {
        d.rect(side, side);
    }
}
```

6.

```
interface Action {
    void execute(Drawing d, Stack<DrawCommand> history);
}

class DrawAction implements Action {

    private DrawCommand command;

    public DrawAction  (DrawCommand command) {
        this.command = command;
    }

    public void execute(Drawing d, Stack<DrawCommand> history) {
        history.push(command);
        command.draw(d);
    }
}

class Undo implements Action {

    public void execute(Drawing d, Stack<DrawCommand> history) {
        history.pop().undo();
```

```
    }
}

class Exit implements Action {
    public void execute(Drawing d, Stack<DrawCommand> history) {
        System.exit(0);
    }
}

interface GUI {

    Action next();
}

class DrawDispatcher {

    void run(GUI gui, Drawing d) {
        Stack<DrawCommand> history = new Stack<>();
        while (true) {
            gui.next().execute(d, history);
        }
    }
}
```

7. **a**. Vi kan låta nycklarna utgöras av noderna d.v.s. strängar. Vidare associeras varje sträng med en mängd strängar (nodens grannar). (1)

b. Bredden-först: Eftersom vi behandlar en nivå i taget kommer alla noder på samma avstånd från startnoden att behandlas i en följd.

Börja genomgången i startnoden (lägg den till mängden noder på aktuell nivå, markera den som besökt och sätt avståndet till 0). Gå igenom grafen nivå för nivå (öka avståndet med 1 för varje nivå), om destinationsnoden finns bland noderna på en nivå är vi klara och avståndet kan returneras. För varje nod på en nivå lägg dess obesökta grannar till nästa nivå och markera dem som besökta. (1.5)

Vi kan använda två mängder för att hålla reda på noderna på aktuell nivå respektive på nästa nivå. När vi gått igenom alla noderna på aktuell nivå ökar vi avståndet med ett och sätter nästa nivå till aktuell nivå.

c. Vi kan anropa get med aktuell nod (sträng) som parameter (0.5)

d. Inför ett interface med en metod som returnerar grannarna till en nod. (1)