Assignment 2 — Design Patterns Implementation (Version 1)

Course: S26CS6.401 — Software Engineering
Team No:: 27

Repository: https://github.com/Sidx-sys/Reservation-System-Starter

---

1. Code Quality Metrics — Before (Baseline)

Tools used: **DesigniteJava**, **Checkstyle 13.0.0** (Google style)

1.1 Type-Level Metrics (DesigniteJava)

Class	NOF	NOM	LOC	WMC	DIT	LCOM	FANIN	FANOUT
Airport	5	8	37	8	0	0.25	2	0
Customer	3	9	57	10	0	0.00	2	1
Passenger	1	2	9	2	0	0.00	0	0
Flight	4	8	52	12	0	0.00	1	1
Schedule	1	7	31	9	0	0.00	1	2
ScheduledFlight	**3**	**11**	**61**	**17**	**1**	**0.18**	**1**	**1**
FlightOrder	**2**	**10**	**86**	**19**	**1**	**0.30**	**1**	**3**
Order	5	10	37	10	0	0.50	0	1
CreditCard	1	5	29	5	0	0.00	1	0
Paypal	1	0	7	0	0	-1.0	1	0
Helicopter	2	3	22	5	0	0.00	0	0
PassengerDrone	1	1	11	2	0	0.00	0	0
PassengerPlane	1	1	28	5	0	0.00	0	0

NOF = Number of Fields, **NOM** = Number of Methods, **LOC** = Lines of Code, **WMC** = Weighted Methods per Class, **DIT** = Depth of Inheritance Tree, **LCOM** = Lack of Cohesion of Methods, **FANIN** = incoming dependencies, **FANOUT** = outgoing dependencies.

1.2 Method-Level Metrics — High Complexity Methods

Class	Method	LOC	CC	Params
`PassengerPlane`	constructor	23	**5**	1
`Flight`	`isAircraftValid`	19	**4**	1
`ScheduledFlight`	`getCapacity`	12	**4**	0
`ScheduledFlight`	`getCrewMemberCapacity`	12	**4**	0
`FlightOrder`	`processOrderWithCreditCard`	13	**4**	1

Class	Method	LOC	CC	Params
FlightOrder	processOrderWithPayPal	13	**4**	2
FlightOrder	payWithCreditCard	15	**3**	2
Helicopter	constructor	12	**3**	1
Customer	isOrderValid	16	1	2
FlightOrder	isOrderValid	16	1	3

CC = Cyclomatic Complexity. Methods with CC >= 3 are highlighted.

1.3 Design Smells (DesigniteJava)

Class	Smell
Customer	Cyclic-Dependent Modularization
Passenger	Unutilized Abstraction
ScheduledFlight	Broken Hierarchy, Missing Hierarchy
FlightOrder	Unutilized Abstraction, Broken Hierarchy, Cyclic-Dependent Modularization
Order	Unutilized Abstraction
Paypal	Unnecessary Abstraction, Deficient Encapsulation
PassengerPlane	Unutilized Abstraction, Deficient Encapsulation
Helicopter, PassengerDrone	Unutilized Abstraction
Runner	Unutilized Abstraction

1.4 Implementation Smells (DesigniteJava)

Class	Method	Smell
Schedule	removeFlight	Complex Conditional, Long Statement
Schedule	scheduleFlight	Long Statement
ScheduledFlight	constructor	Long Parameter List (x2)
ScheduledFlight	getCrewMemberCapacity, getCapacity	Magic Number
FlightOrder	processOrderWithPayPal	Complex Conditional
CreditCard	constructor	Magic Number
Helicopter	constructor	Magic Number (x2)
PassengerPlane	constructor	Magic Number (x8), Missing Default

1.5 Key Observations

- **`FlightOrder`** has the highest LOC (86) and WMC (19) — it handles both order management and all payment processing.
- **`ScheduledFlight`** has WMC=17 due to `instanceof` chains in `getCapacity()` and `getCrewMemberCapacity()`.
- **`FlightOrder`** LCOM=0.30 — low cohesion; payment methods don't share fields with order methods.
- **`isOrderValid()`** is duplicated in both `Customer` and `FlightOrder`.
- **No common Aircraft interface** — all three aircraft types are unrelated classes, forcing `Object` type usage and `instanceof` checks.

---

2. Design Pattern: Adapter Pattern

2.1 Problem Identified

`PassengerPlane`, `Helicopter`, and `PassengerDrone` have **no common interface**:

Class	Model Access	Capacity Access	Crew Access
`PassengerPlane`	`public` field `model`	`public` field `passengerCapacity`	`public` field `crewCapacity`
`Helicopter`	`getModel()`	`getPassengerCapacity()`	(none — hardcoded as `2` in ScheduledFlight)
`PassengerDrone`	field `model`	(none — hardcoded as `4` in ScheduledFlight)	(none — hardcoded as `0` in ScheduledFlight)

This forces: - `Flight.aircraft` to be stored as **`Object`** (no type safety) - **`instanceof` chains** in 3 methods: `Flight.isAircraftValid()`, `ScheduledFlight.getCapacity()`, `ScheduledFlight.getCrewMemberCapacity()` - **Magic numbers** scattered in ScheduledFlight

2.2 Before — Current Code

```
// Flight.java
protected Object aircraft;  // stored as Object!

private boolean isAircraftValid(Airport airport) {
    return Arrays.stream(airport.getAllowedAircrafts()).anyMatch(x -> {
        String model;
        if (this.aircraft instanceof PassengerPlane) {
            model = ((PassengerPlane) this.aircraft).model;
```

```
        } else if (this.aircraft instanceof Helicopter) {
            model = ((Helicopter) this.aircraft).getModel();
        } else if (this.aircraft instanceof PassengerDrone) {
            model = "HypaHype";  // hardcoded!
        } else {
            throw new IllegalArgumentException("Aircraft is not recognized");
        }
        return x.equals(model);
    });
}
```

```
// ScheduledFlight.java
public int getCapacity() throws NoSuchFieldException {
    if (this.aircraft instanceof PassengerPlane)
        return ((PassengerPlane) this.aircraft).passengerCapacity;
    if (this.aircraft instanceof Helicopter)
        return ((Helicopter) this.aircraft).getPassengerCapacity();
    if (this.aircraft instanceof PassengerDrone)
        return 4;  // magic number!
    throw new NoSuchFieldException("...");
}

public int getCrewMemberCapacity() throws NoSuchFieldException {
    if (this.aircraft instanceof PassengerPlane)
        return ((PassengerPlane) this.aircraft).crewCapacity;
    if (this.aircraft instanceof Helicopter)
        return 2;  // magic number!
    if (this.aircraft instanceof PassengerDrone)
        return 0;  // magic number!
    throw new NoSuchFieldException("...");
}
```