Designing Applications

Mikael Svahnberg*

2023-08-25

Contents

1 Introduction **2**

2 Discussion: Classes and Attributes **Discussion** **2**

3 Identifying Classes **3**

4 Finding Classes **3**

5 Scenarios and Responsibilities **3**

6 More on class Responsibilities **4**

7 Attributes **4**

8 Associations **4**

9 Finding Associations **5**

10 Discussion: Multiplicity **Discussion** **5**

11 More on Classes and Objects **5**

12 The UML Way **6**
12.1 Use Cases 6
12.2 Interaction Diagrams 7

13 Discussion: Example **Discussion** **8**

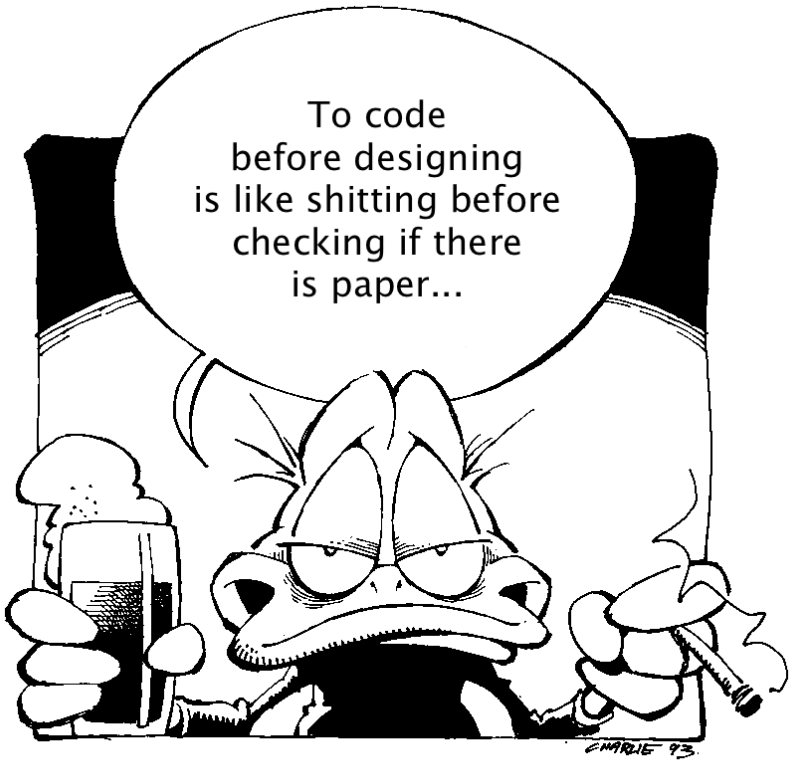
14 Introducing Design Patterns **8**
14.1 Christopher Polhem, father of Design Patterns? 8
14.2 Patterns – a brief introduction 8
14.3 Some Common Design Patterns 9
14.4 Exampe: Decorator Pattern 9

15 Summary **10**

*Mikael.Svahnberg@bth.se

1 Introduction

- Barnes & Kölling Chapter 15, Designing Applications
- Object Oriented Analysis and Design
- UML (Unified Modelling Language) / RUP
- Discovering Classes
- Designing Interfaces



2 Discussion: Classes and AttributesDiscussion

- How can we find / What are:
  - Classes
  - Attributes
  - Associations
- What is the difference between an *Attribute* and a *Class*

3 Identifying Classes

Category	Examples	
Physical Objects	POST	Aeroplane
Places	Store	Aerport
Transactions	Payment	Reservation
Containers	Basket	Aeroplane
Things in Container	Item	Passenger
Events	Sale	Flight
Description of Things	Sale Item	Flight Description
Records, Contracts	Receipt	Ticket

4 Finding Classes

- Look for *nouns*
- *verbs* may be methods on the nouns.
  - On what noun does the verb operate?

Sources:

- Textual description of problem domain
- Requirements
- Use-cases

Cave!

- Natural language is ambiguous
- Class or Attribute? Decide for now, may change later.

5 Scenarios and Responsibilities

- Walk through what needs to happen in the system to solve a particular task.
- A team can "role play" as objects in a scenario.
- Start with a simple high-level description
- Follow the execution to define *Responsibilities* , methods, and new classes.

A customer calls the cinema and wants to make a reservation for two seats tonight to watch the movie *Ben Hur*. The cinema employee uses the booking system to find and reserve two seats on the third row.

1. Which class do we start with? `BookingSystem`
2. What do we do? `search for show by title and day`

- `search()` is a method on `BookingSystem`
- `Show` is a new class which we did not see in the initial description.

3. How does `BookingSystem` find the shows?
   - `BookingSystem` already has one responsibility: Interface with the cinema employee
   - Better to delegate management of shows to another class: `ShowCollection`

... and so on. **Discuss** what is the next step?

6 More on class Responsibilities

- Design Principle: *High Cohesion*

The methods and attributes in a class should all work within a *small and well-defined area of responsibility.*

- More responsibilities → more methods that do not work on the same underlying data
- Add new classes instead

7 Attributes

- Logical value of an element
  - Examples: *name, quantity, status, ...*
  - Hint: Builtin data types
    - String, int, date
    - But also simple user-defined types such as *address, personnummer, ...*
- **Keep Attributes Simple**

8 Associations

An association is a

- relationship between concepts
- indicates a meaningful and interesting connection

Types

- Need-to-know (preserved for some time; needs to maintained by software)
- Comprehension-only (used to understand domain)

9 Finding Associations

Category	Examples
A – is a part of – B	Salesitem – Sale
	Wing – Aeroplane
A – is contained in – B	Item – Store
	Seat – Flight
A – is a description for – B	ItemDescription – Item
	FlightInformation – Flight
A – is known/recorded in – B	Sale – POST
	Booking – Flight
A – is owned by – B	Store – Company
A – related transactions – B	Payment – Sale
	Booking – Ticket

10 Discussion: Multiplicity DISCUSSION

- One object of type A has an association to a number of objects of type B

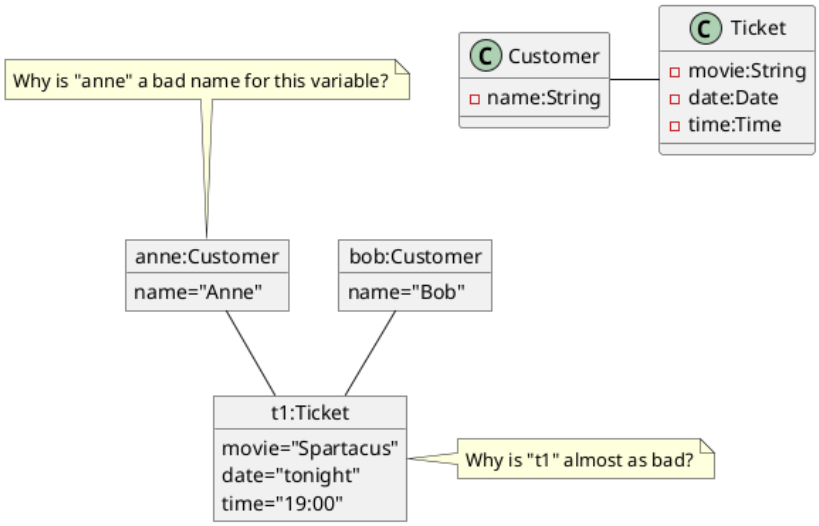
- one to one A – B
- one to specific number A – "5" B
- one to range A – "1..10" B
- one to at least one A – "+" B
- one to at least specific number A – "10+" B
- one to any number A – "*" B
- many to many A "10" – "5" B
- *Always look at multiplicity from the perspective of a single object*

11 More on Classes and Objects

- Real world is full of *Objects* and no classes
- A *Class* is a collective description of objects that look the same or behave in the same way
- When a new object is introduced, we have the choice to:
  1. Create a new object of an existing class, e.g. `new Ticket()` and set the attributes on this new object to whatever makes this object unique.

2. Create a complete new class to describe this new object, then create a new object based on this new class.
3. Inherit from an existing class and only define what differs. then create a new object based on this new sub-class.

- Either way, there are a few terms we need to keep in mind:
  - The *name of the variable* in which we store a reference to the object
    - This doesn't really have anything to do with the actual object.
  - The *Class* we use as a template for the object
  - The *attributes* that a class declares that an object must have
  - The *values* of the attributes in an object
    - **These values can be, and often are, references to other objects.**
- Interactions between objects are modelled as associations between classes



12 The UML Way

12.1 Use Cases

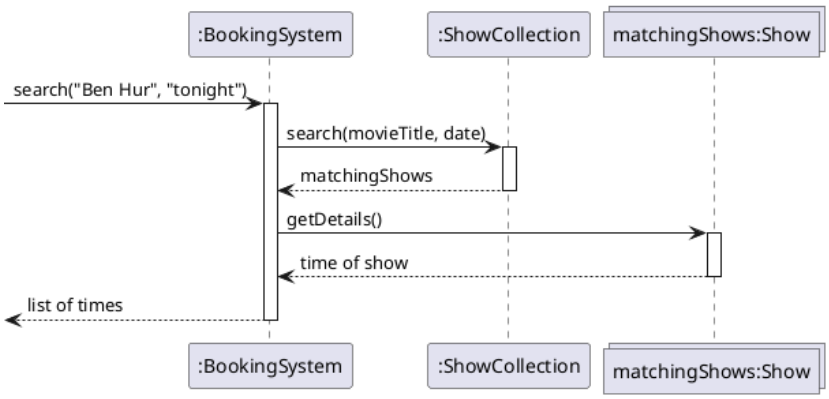
- The starting scenario we used earlier
- More details on the specific interaction

Use Case Reserve Seats for Show **Actors** Customer, Cinema Employee **Desciption** A customer calls the cinema and wants to make a reservation for two seats tonight to watch the movie *Ben Hur*. The cinema employee uses the booking system to find and reserve two seats on the third row.

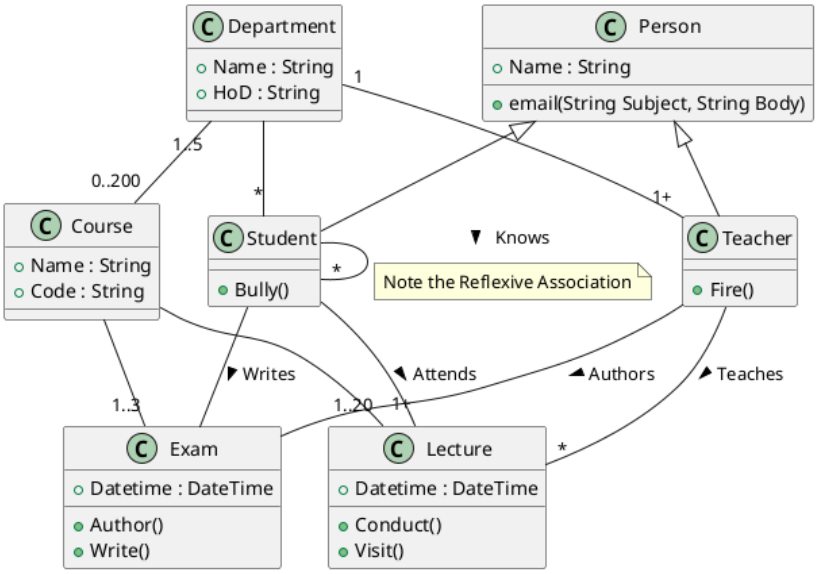
Main Course of Events

Actor	System
1. The employee searches for showings of “Ben Hur” today	
	2. The system lists all shows
3. The employee asks the customer which show they wish to see	
4. The employee selects the “19:00” show.	
	5. The system lists available seats
6. The employee asks the customer how many seats they want and where they want to sit.	
7. The employee selects two seats on the third row.	
	8. The system reserves the two se
	9. The system confirms the reserv and returns a reservation code.
10. The employee gives the reservation code to the customer and ends the all.	

12.2 Interaction Diagrams

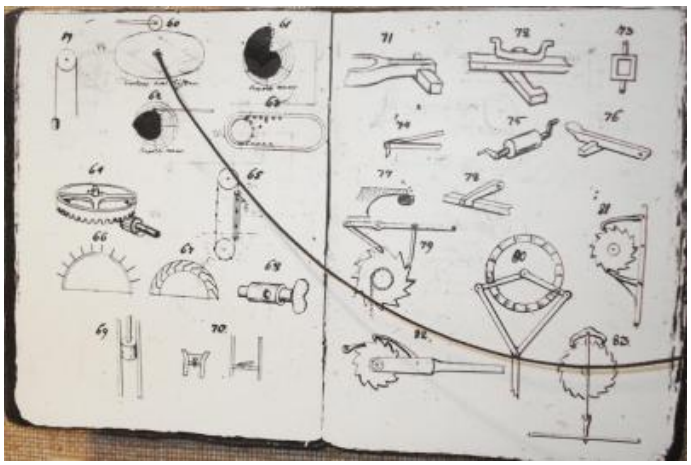


13 Discussion: Example DISCUSSION



14 Introducing Design Patterns

14.1 Christopher Polhem, father of Design Patterns?



14.2 Patterns – a brief introduction

- Design patterns are reusable solutions to known problems
  - With known consequences
  - "encoded experience"
  - Codified in a structured format
  - named

- There is nothing that *requires* you to use design patterns; they are a convenience.
- Design patterns focus primarily on structure (class view), and interaction (sequence diagrams).

Responsibility Driven Design

14.3 Some Common Design Patterns

From Gamma, E., Helm, R., Johnson, R., & Vlissides, J., *Design patterns: elements of reusable object-oriented languages and systems* (1994), Reading MA: Addison-Wesley.

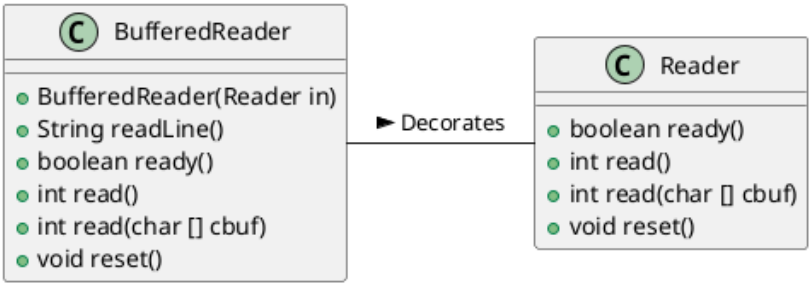
1. Strategy
2. State
3. Observer
4. Facade, Wrapper, Adapter
5. Abstract Facory
6. Singleton
7. Builder
8. Command
9. Interpreter
10. Visitor

Barnes and Kölling, Ch15:

- Decorator $\approx$ Facade, Adapter
- Singleton
- Factory Method $\in$ Abstract Factory
- Observer

14.4 Exampe: Decorator Pattern

- Commonly used in Java APIs
- A more low-level API class is *decorated* with a more advanced set of methods.



- Yes, `BufferedReader` is an extension of `Reader`
  - that adds a few methods such as `readLine()`
  - we will introduce Inheritance soon.

15 Summary

- Objects in the Real World
- Classes in a program
- What is a Class, What is an Attribute?
- Class Diagram
- Defining Responsibilities for a Class
- Design Principle: *High Cohesion*
- Design Patterns

16 Next Lecture: Collections of Objects

- Barnes & Kölling Chapter 4, Grouping Objects
- Barnes & Kölling Chapter 7, Fixed-Size Collections – Arrays
- Collections of Objects
  - ArrayList
  - Array
- Iteration
  - Iterators
  - for-each
  - `while`
  - `for`
- Java Standard Library
- C++ Standard Libraries