



# Design Patterns

Mikael Svahnberg<sup>1</sup>

2016-04-06

---

<sup>1</sup>Mikael.Svahnberg@bth.se



# Responsibility-Driven Design

## Responsibility for **Doing**

- Doing something (e.g. a calculation)
- Creating other objects
- Initiating an action in another object
- Controlling and coordinating other objects

## Responsibility for **Knowing**

- Knowing about private encapsulated data
- Knowing about related objects
- Knowing about things it can derive or calculate



# Levels of Patterns

Different levels:

- Architecture
  - Systems, subsystems
- Design
  - Classes, groups of classes
- Idioms
  - One class, functions within one class
- GRASP
  - In some sense orthogonal
  - Learning aid for OO Design
  - Advice for Assigning Responsibilities



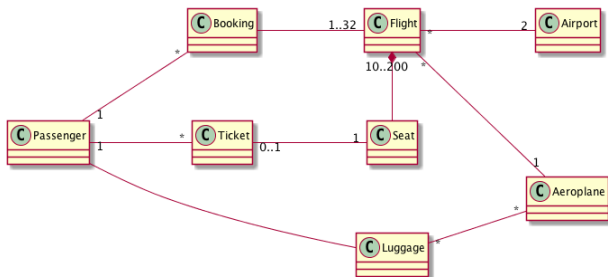
# GRASP Patterns

(Listed on the inside of the book cover)

- Information Expert
- Creator
- Controller
- Low Coupling
- High Cohesion
- Polymorphism
- Pure Fabrication
- Indirection
- Protected Variations



# Example: GRASP Patterns



## Discuss

- Who should calculate the cost of a Booking?
- Who should be responsible for creating a Ticket?
- Why should a Passenger not be aware of the Flight?
- How should a Passenger interact with this system when booking a trip?
- How would you implement first, business, and third class?



# Example: Design Patterns in Pacman

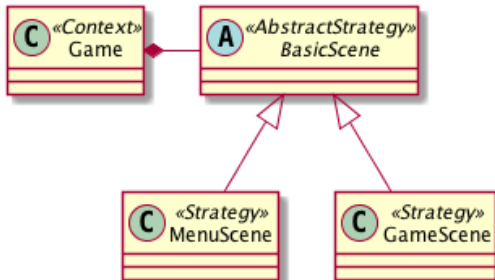
A look at the game:

- The game consists of scenes (Main Menu, Actual Game, High-Score List)
- Each scene consists of a number of [different] objects (graphical as well as audio)
- Some objects need awareness of other objects
- Some objects in each scene needs to deal with UI input



# Different Scenes

- Problem: The game consists of several scenes (Main Menu, Actual Game, High-Score List)
- Design Pattern: Strategy
- Involved Classes: Context, «abstract» Strategy, ConcreteStrategy\*
- Pacman: Game, «abstract» BasicScene, MenuScene, GameScene





# Creating Objects for different Scenes

- Problem: Set up all objects necessary for each Scene
  - Design Pattern: Builder
  - Involved classes: Director, «abstract» Builder, ConcreteBuilder\*
  - Pacman: MenuScene/GameScene, «abstract» WorldCreator, GameCreator, MenuCreator
- 
- Design Pattern: Factory Method
  - Involved classes: Creator (with «abstract»FactoryMethod()), ConcreteCreator (with instantiated FactoryMethod()\*)
  - Pacman: Scene (with «abstract»createObjects()), GameCreator (with instantiated createObjects()), ...





# Behaviour of Ghosts I

- Problem: Each ghost behaves in a different way.
- Design Pattern: Strategy
- Involved Classes: Context, «abstract» Strategy, ConcreteStrategy\*
- Pacman: Ghost, «abstract» GhostMovementStrategy, BlinkyStrategy, InkyStrategy, PinkyStrategy, ClydeStrategy



# Only one Audio/Graphics/World

- Problem: Avoid creating more than one instance of AudioManagement, GraphicsManagement, World
- Design Pattern: Singleton
- Involved Classes: Singleton (with static getInstance(), private constructor)
- **Less Optional Alternative:** Coding Pattern: Only create stuff in one place, keep central repository with pointers to these objects.



# Redirecting Input

- Problem: Different objects are interested in UI input
- Design Pattern: Observer
- Involved Classes: Observable, Observer
- Pacman: InputManager, PacmanObject, MainMenuObject



# Behaviour of Ghosts II

- Problem: When pacman eats supercandy, the behaviour of the ghosts change
- Design Pattern: State
- Involved Classes: Context, «abstract» State, ConcreteState\*
- Pacman: Ghost, «abstract» GhostState, GhostNormalState (see above, GhostStrategy), GhostChasedState



# Architecture Patterns

Examples of Systems:

- Compiler
- Interactive System, e.g. a Time Management Program



# Architecture Patterns

Examples of Systems:

- Compiler
- Interactive System, e.g. a Time Management Program

... imposed with different quality requirements:

- Performance: High Throughput
- Flexibility / Continuous Deployment



# Discuss: When to worry about Design Patterns

When should you introduce patterns into your system?

- GRASP patterns
- Design Patterns
- Architectural Patterns