Design Patterns

MSc in Communications Software

Produced by

Eamonn de Leastar (edeleastar@wit.ie)

Department of Computing, Maths & Physics
Waterford Institute of Technology

http://www.wit.ie

http://elearning.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit

Design Patterns Principles

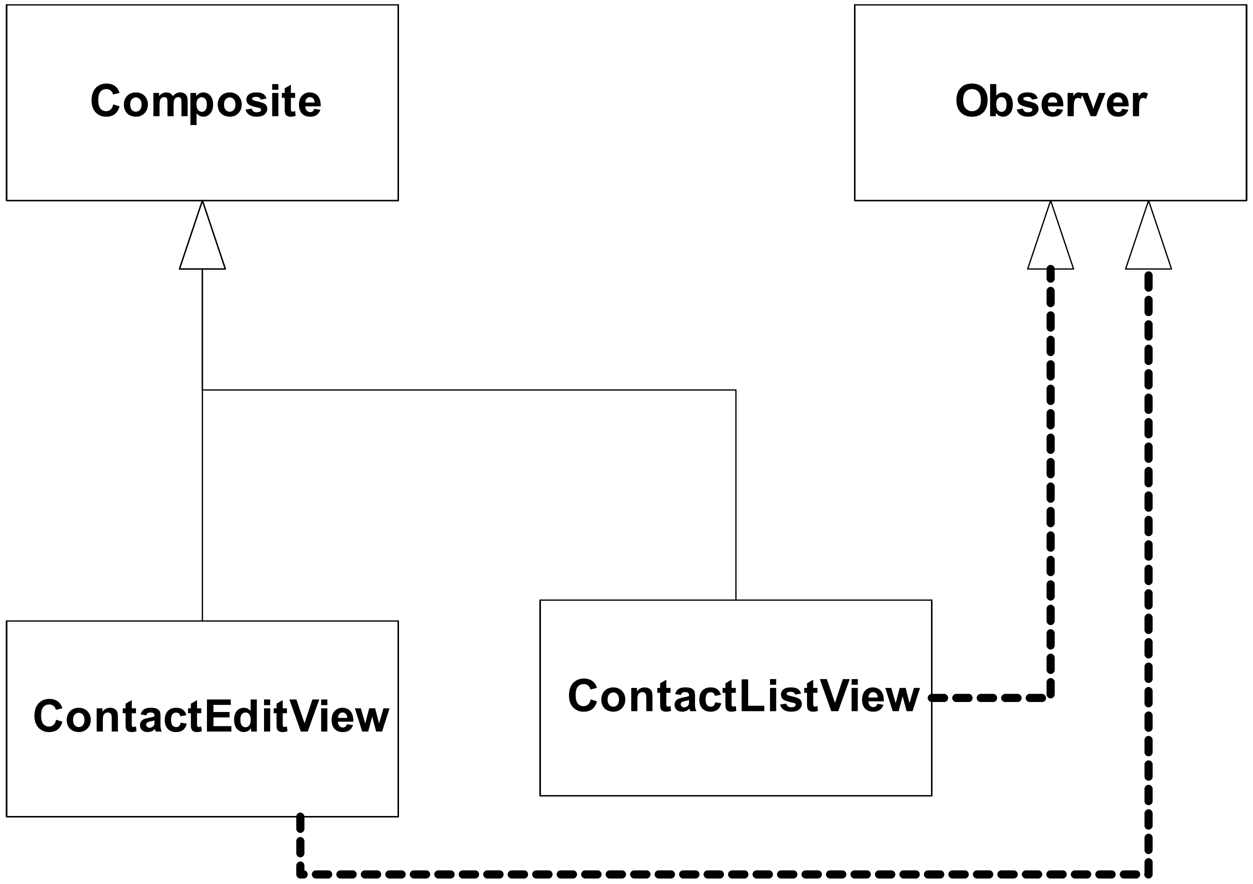
The thinking behind patterns

Five Important Principles

1. Distinguish between Classes & Types
2. Distinguish between interface & implementation inheritance (implements & extends)
3. Program to Interface not Implementation
4. Favour Composition over Inheritance
5. Find what varies & encapsulate it

(1) Classes & Types

- A Class defines how the object is implemented.
  - It defines the object's internal state and the implementation of its operations.
- A Type only refers to its interface
  - the set of requests to which it can respond.
- An object can have many types, and objects of different classes can have the same type.



(2) Interface & Implementation

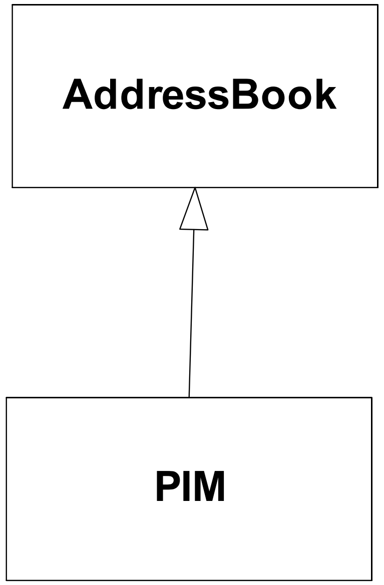
- Languages like C++ and Eiffel use classes to specify both an object's type and its implementation
- Java can separate these:
  - Interface for type
  - Class for class
- Key distinction between interface inheritance and implementation inheritance:
  - implements: Interface inheritance describes when an object can be used in place of another. – Reducing dependencies, reusability, adaptability
  - extends: Implementation inheritance defines an object's implementation in terms of another object's implementation – Localization & Reuse of code

(3) Programming to Interfaces

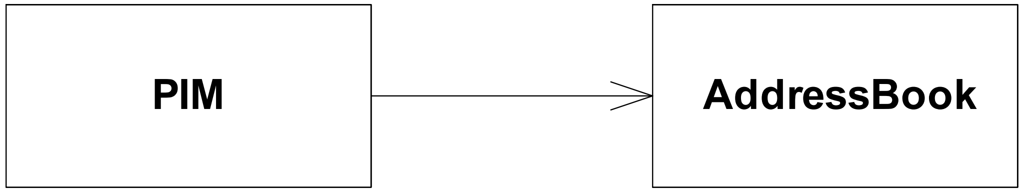
- Use interfaces to define types
- Declare object references to be associated with the types (instead of the classes implementing the types)
- Use Creational patterns
  - to associate interfaces with implementations
  - protects the package responsible for creating concrete objects from depending on specific concrete classes
- Benefits
  - Greatly reduces the implementation dependencies
  - Client objects remain unaware of the classes that implement the objects they use.
  - Clients know only about the types (interfaces).

(4) Inheritance vs Composition (1)

- Two common techniques for reusing functionality:
  - White-box reuse: Class inheritance - defines the implementation of one class in terms of another. The internals of parent classes are visible to subclasses.



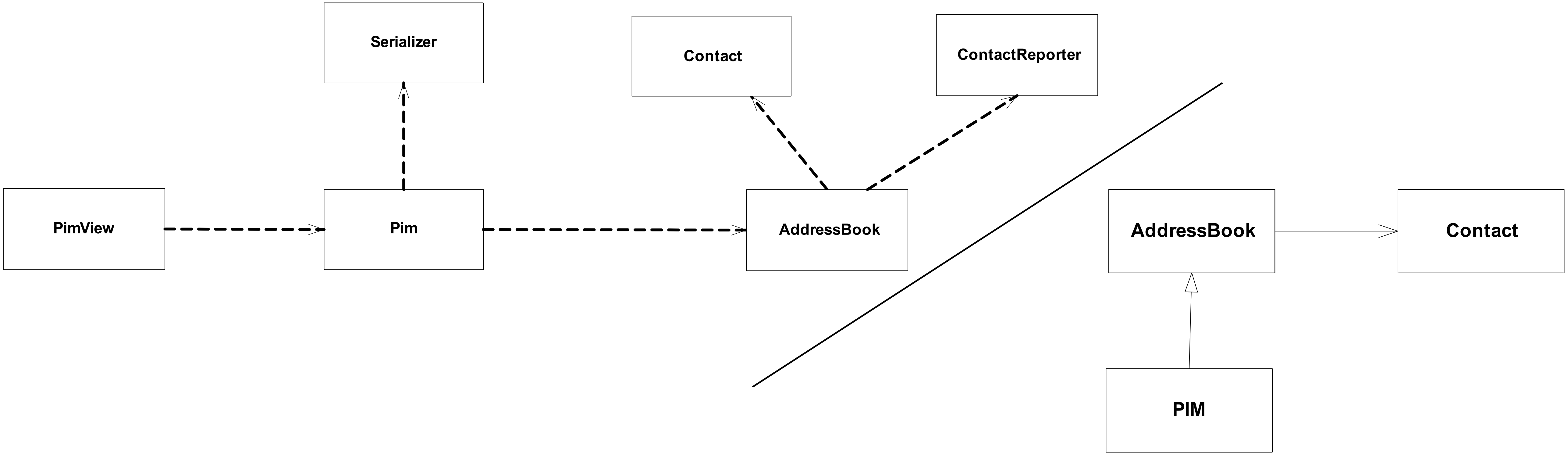
  - Black-box reuse: Object Composition - functionality is obtained by assembling or composing objects to get more complex functionality. Requires that the objects being composed have well-defined interfaces.



(4) Inheritance vs. Composition (2)

- Class Inheritance
  - easy to use ⇒ easy to modify, implementation being reused
  - change in parent ⇒ change in subclass, breaks encapsulation
  - change in subclass ⇒ change in inherited parent behaviour
- Object Composition
  - objects are accessed solely through interfaces
  - no break of encapsulation
  - any object can be replaced by another at runtime as long as they are the same type

(4) Inheritance vs. Composition (3)



- Keeps classes focused on one task – high cohesion
- Implies having more objects, with the system's behaviour captured in their interactions
- Potential for reuse increases

(5) Encapsulate the concept that varies

- Patterns typically attempt to locate the axis of change within a set of abstractions
- … and encapsulate that axis.
- E.g: Command pattern:
    - the variability is when & how a request is to be fulfilled.
    - These commands are encapsulated as first class objects
    - … and can be passed, stored, retrieved and interrogated
- E.g. Strategy
    - Identify the variability in a given algorithm (widget layout algorithm)
    - .. Encapsulate this in an interface (LayoutManager)
    - Realise alternatives as implementations of this interface
    - Recompose the algorithm in terms of this interface.

Except where otherwise noted, this content is licensed under a Creative Commons Attribution-NonCommercial 3.0 License.

For more information, please see http://creativecommons.org/licenses/by-nc/3.0/

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning
support unit