

6. Design

6. Design

- 6.1 Design Principles
- 6.2 Object Concepts
- 6.3 UML Odds and Ends
- 6.4 Design Patterns
- 6.5 Mapping UML to Code
- 6.6 Code Reuse
- 6.7 Class Design
- 6.8 Refactoring
- 6.9 User Interfaces
- 6.10 Persistence

What is Design?

- Take the analysis model and make it more detailed
- Take the analysis model and add technical considerations:
 - programming language
 - operating system
 - class library
 - database
 - architecture
 - distribution
 - interoperability
 - other nonfunctional requirements
- Prepare implementation
 - the implementer should not have to worry about design issues
- The “How” of the program

Three-Tier Architecture

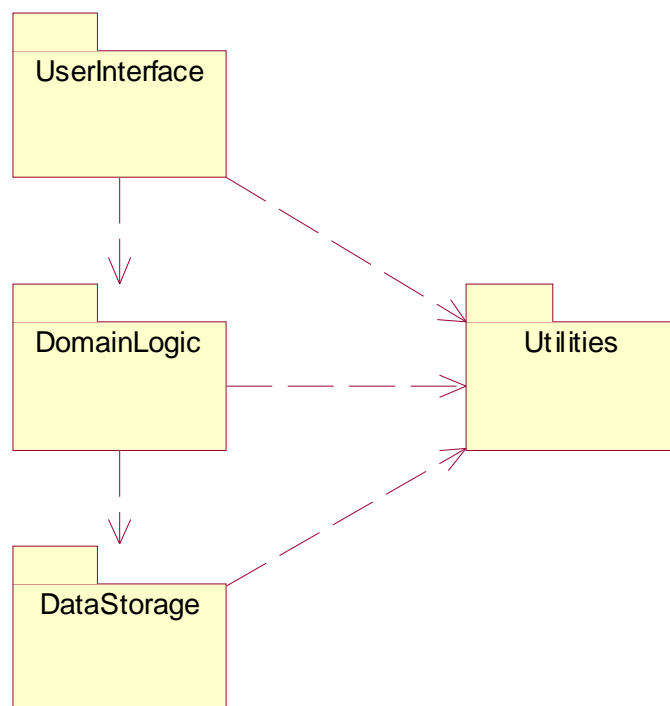
- A very common architecture for application software
- Simple
- Top tier: user interface
 - windows, dialogs, keyboard shortcuts
 - determines overall control flow
- Middle tier: domain logic
 - all the classes from analysis
 - here the actual task of the program is solved
- Bottom tier: data storage
 - database and database connectivity
- Constraint:
 - each tier knows only about the tier directly below

Two-Tier Architecture

- used for very simple Web applications
- not recommended for more complex applications
- Top Tier: user interface
 - including a small amount of functionality, if needed
- Bottom Tier: database

- Advantage: quick and easy
- Disadvantage: becomes ugly if more functionality added later

Example: Three-Tier Architecture



6.1 Design Principle: Abstraction

- An abstract thing represents several less abstract things
- Example:
 - abstract: Person
 - less abstract: Chair, Professor
 - not abstract at all: Jay Bagga
- Abstraction is the process of selecting relevant information
- Example: abstract from “Jay Bagga” to “Person”
 - relevant: has a name, birth date, address, is human
 - not relevant: starts with B, has a beard
 - these properties are incidental to one object
 - they don’t hold for all persons
- Purpose of abstraction
 - representing a bunch of different things by one

Design Principle: Information Hiding

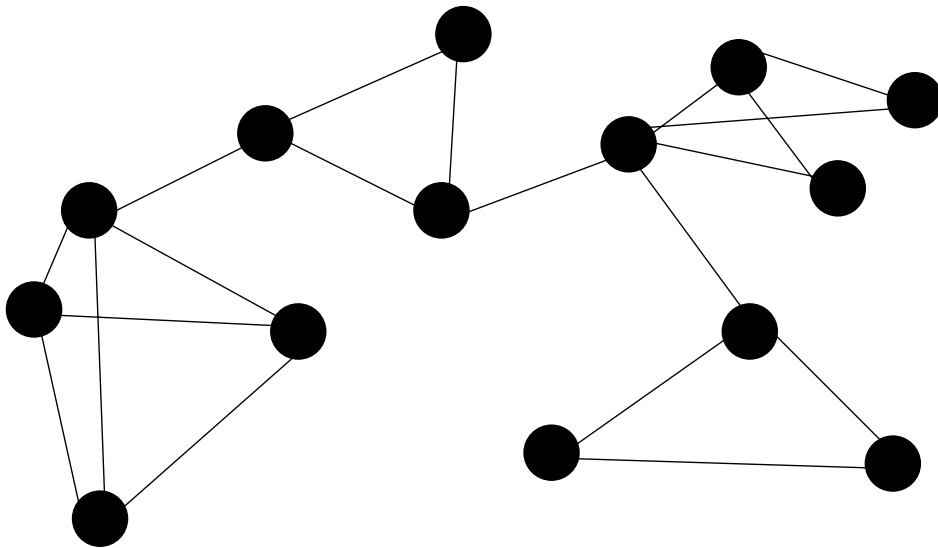
- A class (or subsystem) should show as little information to the outside as possible
 - show the specification
 - hide the implementation
- For example:
 - make most operations public
 - make attributes and some operations private
- Purpose:
 - implementation can be changed later without breaking other modules

Design Principle: Low Coupling, High Cohesion

- High cohesion
 - the inner parts of a module talk to each other a lot
- Low coupling
 - the modules themselves talk to each very little
- Draw the lines between classes and subsystems so that this is true
 - identify nodes of high communication -> modules
 - identify places of low communication -> between modules
- Purpose:
 - to make it easy to exchange modules
 - a module with lots of connections to other modules is hard to exchange

Coupling / Cohesion Example

- Dots: methods, lines: calls between methods
- Where to draw the module boundaries?



Design Principle: Separation of Concerns

- Deal with each aspect of a problem separately
- Example:
 - Bad: have one class to deal with all of the following: Gui, domain logic, and data storage of videos
 - Good: have a VideoDialog, a Video, and an SQLVideo
- Not always possible...
 - Inherent properties of code can't be separated out:
 - performance
 - reliability
 - security

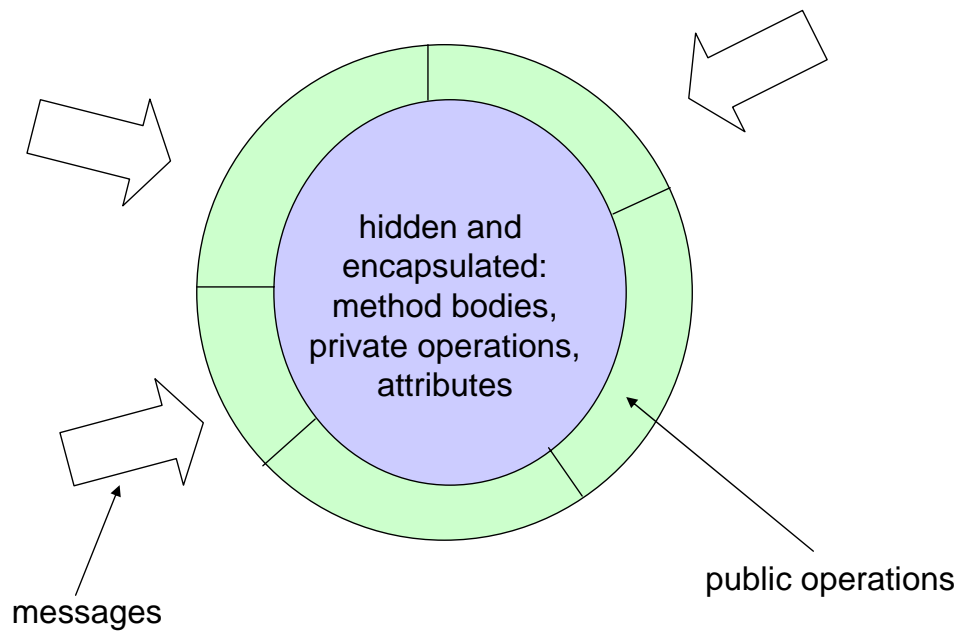
6.2 Object Concepts

- Object-Orientation
 - a way to analyze, design, implement programs
- History:
 - Simula (1960s)
 - Smalltalk (1970s)
 - C++, Eiffel (1980s)
 - Java, UML (1990s)
- Core idea:
 - objects can react to messages

Before Objects

- Before objects: separation of data and control
 - data structures: modules of data
 - procedures or functions: modules of control
- Methods such as...
 - ER diagrams (to structure data)
 - Structured Programming
 - procedures, loops, if-statements
 - records, unions, enumerations
 - languages: Pascal, C

Objects



Objects

- Can react to messages
 - someone sends a message "who are you?"
 - object replies: "I am a string!"
 - or something like that...
- Contain both data and control
 - strings contain all their characters (in an array for example)
 - can do operations on them
 - toUppercase()
 - concat()
 - substring(int start, int end)
 - indexOf(char c)

Objects and Classes

- A class is a template for creating objects
 - a class can create new objects
 - keyword “new”
 - `Java. lang. Class. newInstance()`
 - all objects that were created by the same class have things in common
 - minimum set of operations
 - minimum set of attributes
- Objects created by a class are called its instances
- Type system
 - classes also serve as types
 - one object can belong to several types
 - because of inheritance

Three Principles of Objects

- Encapsulation
 - each class decides which of the operations and attributes of its objects are public
 - private things cannot be accessed by other objects
 - purpose: information hiding
- Inheritance
 - a class can be a subclass of one or more other classes
 - subclass inherits all features from superclass
 - can override features if necessary
- Polymorphism
 - two objects with the same type can behave differently
 - same specification, different implementation
 - an objects precise behavior is determined only at runtime

Abstract Classes and Interfaces

- Abstract class
 - a class with incomplete implementation
 - cannot be instantiated
- Interface
 - an abstract class with no implementation at all
 - pure specification
 - used in Java as a work-around for lack of multiple inheritance
- Specification inheritance vs. Implementation inheritance
 - Java: “implements” and “extends”

Multiple Inheritance

- If a class has more than one superclass
 - not allowed in Java (except with interfaces)
 - allowed in C++
- Example
 - a square is a kind of rectangle
 - a square is a kind of rhombus
 - but not every rhombus is a rectangle, and not every rectangle is a rhombus
- Problem of multiple inheritance
 - multiple overrides
 - what if both superclasses have a method `getArea()`?
Which one is inherited?

Polymorphism Example

- abstract class Shape
 - double getArea() // return the area
- class Rectangle extends Shape
 - Rectangle(int upperLeftX, int upperLeftY, int width, int height) {...}
 - double getArea() { return width * height; }
- class Circle extends Shape
 - Circle(int centerX, int centerY, int radius) {...}
 - double getArea() { return Math.PI * radius * radius ;}
- getArea is specified by Shape
 - and implemented in two different ways by its subclasses
 - each implementation fulfills the spec
 - each implementation is different