

# OO-Design

Steven R. Bagley & Colin A Higgins

# Introduction

- Object-Oriented Design:  
Splitting problem into classes
- Techniques for breaking the problem down
- Applying those techniques in practice

# Object System

- OO Programs consists of Objects that work together
- Messages sent between objects to perform tasks
- Each object responsible for separate parts of the program



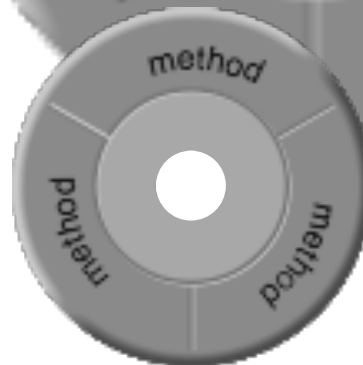
GraphView



Data



Graphics



Window

# Objects

- Loose coupling & single (or related) responsibility
- Data only knows about the stored data
- UI objects only know about UI
- `GraphView` links data and UI

# Class Design

- Finding classes and class relationships
- Some may already be given  
e.g. UI Framework
- Some may be obvious

# How to start

- Sit-down and start coding...
- Solve problems as they arrive
- Surprisingly this can work...
- Single Developer
- Solid understanding of user needs, good judgement and strong code organization

# Structured approach

- Not always one developer
- Long project time
- Need to clearly define classes, responsibilities and collaborations
- Problem Analysis
- Formal methods defined for these tasks



# Design

- Analysis broken down into a set of:
  - Classes
  - Relationships
  - Operations

# Identify classes

- Nouns — good candidates
  - Relevant
  - Irrelevant
  - Fuzzy
- Not all objects will be found this way

# CRC Cards

- Class, Responsibility, Collaboration
- Index cards
- One card per class
  - Responsibilities in one column
  - Collaborators in another
  - Fields on the back

# PdfVM

## Operations

ProcessOperator

AttachResourceLocator

GetTreeRoot

## Collaborators

PdfToken, PdfTokenStack

PdfResourceLocator

PdfGraphicNode

# CRC cards

- Card created as each class discovered
- Operations and Collaborations added as they are found
- Work in teams (or develop a split personality)
- Rip them up, start again
- Unlikely to hit a perfect design first time

# Using CRC cards

- Move them around
- Layout classes near to where classes they collaborate with
- The arrangement will give you an idea as to the quality of the design

# VoiceMailAutomated Teller Machine

- That's the theory
- Let's try it in practice
- VoiceMail System (Horstman)

OR

- Automated Teller Machine (Designing O-O Software, Wirfs-Brock et al, Prentice Hall ISBN 0-13-629825-7)

# Common Class Candidates

- Tangible things
- System Interfaces and Devices
- Agents
- Events and Transactions
- Users and Roles
- Systems
- Containers
- Foundation Classes
- Collaboration Patterns



# What next...

- Pile of CRC cards, giving an outline of the classes
- Formalize the relationships between classes
- Refactor classes

# Association

- Easiest to recognize
- Collaboration implies association
- MailSystem uses IOHandler

# Aggregation

- Stronger than Association
- Class B is a *part* of Class A
- Implies Containment or Management

# Composition

- Stronger Aggregation
- Truck composed of four wheels and engine
- Implies that the implementation is based on the implementation of another
- Black-box code reuse
- Composed class unseen from outside

# Inheritance

- Class B is a specialization of class A
- Base-class may not have been discovered
- White-box reuse
- Base-class visible to outside

# Refactoring

- The ‘Write Once’ rule
- Move similarity between classes into a shared base class
- New class unlikely to be in the problem analysis
- Crucial to building a clean OO-design

# Alternative Design...

- Write user stories
- Adapt these to Use-Cases
- Use Bob Martin's S.O.L.I.D. principles
- Use an ORM (Object-relational mapping)
- Use patterns
- Looks at classes and see if they fit the principles (eg loose coupling, coherent responsibilities etc)

# Summary

- Breaking problem into objects/classes relationships
- CRC card approach
- Formalizing the class relations