The Object-Oriented Design Process

Part 1: objects & classes
Program development

• Programs exist to solve problems
• 3 phase development process:
  – Analysis
  – Design
  – Implementation
Program development

• Process is not linear
  – Implementation experience often leads to design modifications
  – New requirements require additional analysis/design

• OO design process facilitates evolutionary software development:
  – Problem domain, represented by objects and classes, is relatively stable
  – Methods within domain can be changed/improved as necessary
Object-oriented design

• Two questions, initially:
  – What are the players interacting in the system?
  – How should these players be represented?
• Answer to first question is the set of objects
• Answer to second question is the set of classes that describe the objects
Example: voice messaging system

• Your textbook uses the example of a simple voice mail system, which can do the following:
  – Receive voice messages as input
  – Store messages
  – Retrieve (and play back) messages
  – Delete messages

• In addition to the tasks above, the system is also capable of recording, storing, and playing back an outgoing message, and has passcode protection to ensure that only an authorized user can access his/her own “mailbox”
Analysis Phase

• In the analysis phase, we progress from a vague understanding of the problem to a precise description of tasks for the software to carry out.

• Goal of this phase is *functional specification*: a precise description of what the software should do.
Use cases

- One form of functional specification
- Describe intended behavior of system from user standpoint
- Enumerate all tasks system does for user (one use case for each task)
- Describe what needs to be done, not how; algorithm selection occurs in implementation phase, use cases are analysis tools
Use cases

- Each use case is concerned with a specific scenario in which the system interacts with people or entities outside the system (users, for example)
- The user or entity is called an *actor*
- The use case describes the steps necessary to bring the scenario to a completion point that is of some value to one of the actors
Use cases in voice mail example

- The set of use cases in the voice messaging system might include the following scenarios:
  - Reaching an extension
  - Leaving a message
  - Logging in
  - Retrieving messages
  - Changing the greeting
  - Changing the passcode
Example: use case for “reach an extension”

1. User dials main number for voice mail system
2. Voice mail system speaks a prompt: “Enter mailbox number followed by a #.”
3. User keys in recipient’s extension number
4. Voice mail system speaks another prompt: “You have reached mailbox xxxx. Please leave a message now.”
Use case variants

- Scenarios that could deliver a valuable outcome could also fail to do so
- Use cases include variants to describe these situations
Example: use case for “reach an extension”

1. User dials main number for voice mail system
2. Voice mail system speaks a prompt: “Enter mailbox number followed by a #.”
3. User keys in recipient’s extension number
4. Voice mail system speaks another prompt: “You have reached mailbox xxxx. Please leave a message now.”

Variation #1:
1.1 In step 3, user enters an invalid extension
1.2 Voice mail system speaks prompt: “You have typed an invalid mail box number.”
1.3 Resume operation at step 2.
Design Phase

• In this phase, the program designer structures tasks identified in analysis phase into a set of interrelated classes

• Major goals of design phase:
  – Identify classes
  – Identify class responsibilities
  – Identify relationships between classes

• Again, these are goals, not steps; identification of one aspect of a class may lead to changes in, discovery of, others
Purposes of design phase

• Gather information as foundation for implementation phase
• Reduce time required for implementation and testing
• If done correctly, should be most time-consuming phase
Results of design phase

• Text descriptions of classes/responsibilities
• Diagrams depicting:
  – relationships between classes
  – usage scenarios
  – changes in class state
Implementation phase

• Goals of this phase include:
  – coding
  – testing
  – deployment
Traditional development vs. OOP

• Traditional:
  – complete & test each functional unit
  – when all are finished & tested, integrate them
  – often frustrating, as parts don’t always fit together as well as planned

• OO development:
  – program grows gradually
  – working classes and class clusters are successively attached, repeatedly tested
  – integration is part of the process, not done at the end
Rapid prototyping

- A prototype is a program that displays some of the functionality of the final product, deferring implementation of others
- Prototype can lead to new insights, leading to further analysis and design refinements
- OO design particularly suited to prototyping, since objects needed in prototype are likely to be same objects needed in final product
Objects & classes

• Objects: entities that interact in a computer program; properties include:
  – state: collection of information held by object
  – behavior: operations supported by object
  – identity: unique characteristic that differentiates two objects with identical state & behavior

• Class: describes properties of related objects
Objects & classes

• Class definition describes what an object is; includes:
  – operations allowed on the object
  – possible states of the object

• Objects formed using a particular class definition are said to be instances of the class
Identifying classes

- One of the major tasks of the design phase is finding the classes in a problem
- Need to examine functional specification to find objects, then develop classes to describe them
The Object-Oriented Design Process

Part 1: objects & classes