

# Object Oriented Programming – Design

Dr. Robert Lowe

Division of Mathematics and Computer Science  
Maryville College

# Outline

- 1 Basic Concepts
- 2 The Design Process
- 3 Designing the Stock Program

# History & Definitions

- Developed slowly from the 1950s – 1960s.
- Simula (1962) is largely regarded as the first true object oriented language.
- OOP Rose to prominence in the 1980s – 1990s with languages such as C++, Objective C, & Java
- **Object Oriented Programming** – A design paradigm where we decompose a problem into a set of objects which work together to solve a problem.
- **Object** – An entity with state (attributes) and behavior (methods).
- **Class** – A set of objects with the same attributes and methods.

# Basic Principles of Object Oriented Programming

- ➊ **Encapsulation/Data hiding** – Access to attributes is controlled so objects maintain a valid state at all times.
- ➋ **Abstraction** – Objects act as a “black box”, and their use is separated from their implementation. (We neither know nor care how the object works!)
- ➌ **Inheritance** – Classes can be related in a hierarchy of “is-a” relationships. For example: “A squirrel is a mammal.”
- ➍ **Polymorphism** – An object can simultaneously belong to multiple classes, yet still act as itself. For example, if we know how to find the volume of a sphere, we can also find the volume of a basketball.

# Design Goals

- ❶ **High Cohesion** – Each object should have a well defined purpose for existing. No swiss army knives allowed!
- ❷ **Loose Coupling** – Objects can use each other, but all code should depend upon only the public facing interface of an object. Keep the internals of objects on a strict “need to know” basis.

# The Object Oriented Analysis and Design Process

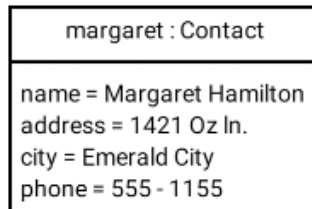
- ➊ Reason about the problem at hand, and identify objects from sample instances of the problem.
- ➋ Identify attributes within each object.
- ➌ Identify behaviors for each object.
- ➍ Group objects into classes based on their attributes and methods.
- ➎ List classes.
- ➏ Define class attributes.
- ➐ Define class methods & constructors.
- ➑ Identify relationships among classes.

# UML – Unified Modeling Language

- **UML** is a graphical design language which allows us to plan objects and classes in a programming language-agnostic way.
- There are many UML diagrams; the most common being:
  - Use Case Diagram
  - Object Diagram
  - Class Diagram
  - Sequence Diagram
- UML was developed by Rational Software from 1994-1996.
- UML became an ISO standard in 2005.

# UML – Object Diagram

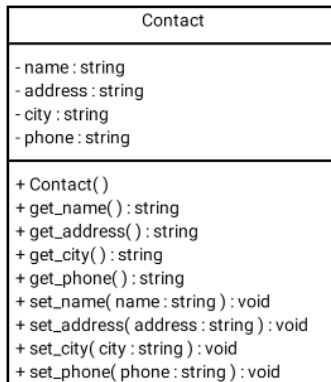
- An **Object Diagram** shows the state of various objects at some point in the program.
- Attributes and values are specified in the format:  
`name = value`
- Names of objects are specified in the format:  
`object : class`
- Lines between objects show that they are contained within each other.





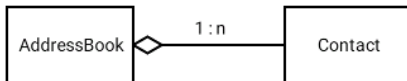
# UML – Class Diagram

- A class diagram shows classes and attributes.
- Attributes are listed in the format:  
name: type
- Methods are listed as:  
name (p1:type, p2:type, ...) : type
- Access modifiers are specified:
  - + public
  - - private
  - # protected



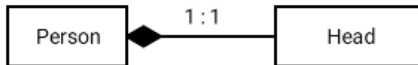
# Class Relationships – Aggregation

- Aggregation is a basic “has-a” relationship.
- Objects of an aggregate class contain objects from another class.
- The objects are created and then given to the aggregate class.
- This is represented with an open diamond on the side of the aggregate.



# Class Relationships – Composition

- Composition is a stronger “has-a” relationship.
- Objects of an aggregate class contain objects from another class.
- The objects are created by the aggregate class.
- Composite objects fully own their parts.
- This is represented with a filled diamond on the side of the aggregate.



# An Intuitive Approach

- ➊ Write out several scenarios for the use of your program.  
(This is a use case).
- ➋ Search for parts of speech:
  - **Nouns** - Objects
  - **Adjectives** - Attributes
  - **Verbs** - Methods
- ➌ Draw out objects.
- ➍ Look for container objects.
- ➎ Rework things until it is flexible enough.
- ➏ Design your classes from the new set of objects.
- ➐ Identify composition and aggregation relationships.

# Pattern: Accessors and Mutators

- One common design pattern is the use of accessors and mutators.
- Recall that all attributes should be private!
- An **accessors** is a function which returns the current value of an attribute.
- A **mutator** is a function which sets the value of an attribute.
- Accessors are typically named `get_<attribute>`.
- mutators are typically named `set_<attribute>`.
- Why go to this level of trouble?
- This allows us to control/validate values and also make some attributes hidden and/or read only!

# Class Activity: Design an Address Book

- What sorts of thing do we want to do with an address book?
- What do the contact records look like?
- What do these things do with each other?
- Construct:
  - Object Diagram
  - Class Diagram
- What relationships exist between the classes?

# The Stock Program (Program 5)

- Play with the stock program solution and read the program specification.
- Draw an object diagram.
- Draw a class diagram.
- Identify relationships.