

# CMSC 132: Object-Oriented Programming II

---



## Object-Oriented Design I

**Department of Computer Science**  
**University of Maryland, College Park**

# Object-Oriented Design

## ■ Goals

- Improve software design
  - Reduce implementation effort
  - Scalable to large software projects
- Try to take advantage of two techniques
  - Abstraction
  - Encapsulation

# Abstraction

- **Abstraction**
  - Provide simple high-level **model** of
    - Physical entity
    - Activity
- **Helpful for managing complexity**
- **Enables information hiding**
  - Can change implementation & representation
  - Will not affect other software components

# Encapsulation

- **Extension of abstraction**
  - Always abstract data & function together
  - Encapsulated entity  $\Rightarrow$  Abstract Data Type (ADT)
- **Examples**
  - List ADT
    - May be implemented as array, linked list, etc...
  - Java collections library

# Benefits of Encapsulation

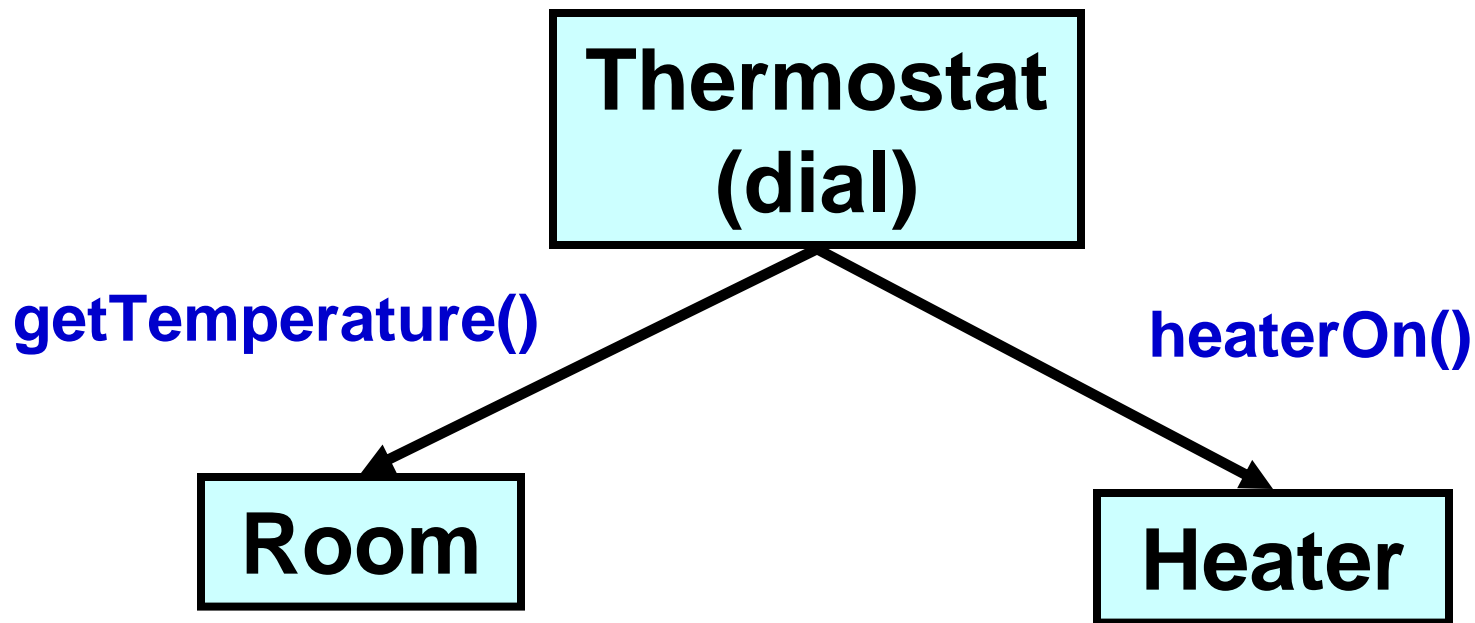
- **Easier to make code modifications**
  - **Due to information hiding**
- **Promotes code reuse**
  - **Interface to data structure clearly defined**
  - **Easier to reuse code**
- **Code reuse increases productivity**

# Object-Oriented Design

- **View software as**
  - A collection of entities (objects)
  - Functions associated with each object
  - Communication between objects
- **Exploits abstraction & encapsulation**
- **Can rely on programming language support**

# Object-Oriented View

- **Example problem description**
  - **Thermostat uses dial setting to control a heater to maintain constant temperature in room**



# History of Object-Oriented Design

- Preceded by **procedure-oriented** view
  - Earliest approach to programming
  - Uses procedure abstraction
  - Similar to actual machine instructions
  - Focus on control flow, program scope
  - Examples: Fortran, Cobol, Pascal, Basic
- Example
  - Thermostat()
    1. Get room temperature
    2. If (temperature < setting) turn heater on
    3. Else turn heater off
    4. Goto step 1

# OO Programming Languages

## ■ Development history

- Simula (Dahl & Nygaard, 1962)
  - Modeling discrete event simulation
- Smalltalk (Kay, 1972)
  - General programming
- C++ (Stroustrup, 1979)
  - Manage complexity in huge software projects
- Java (Gosling, 1991)
  - Designed for embedded processors

# Factors in Success of OO Design

- **Growing demand**
  - More experience with large software projects
- **Improvements in language design**
  - Made OO programming easier
- **Improvements compiler technology**
  - Support more language features efficiently
- **Improvements in hardware**
  - Handled inefficiencies in OO programming
  - Made performance less critical

# Elements of Object-Oriented Design

## ■ **Objects**

- **Entities in program**

## ■ **Methods**

- **Functions associated with objects**

## ■ **Classes**

- **Groups of objects with similar properties**

## ■ **Inheritance**

- **Relationship between classes**

# Objects

## ■ Definition

- Entity that has state, behavior, and identity
- State (data)
  - Properties possessed by object
  - Current values of those properties
- Behavior (methods)
  - How objects react to changes in state
  - How objects interact with each other
- Identity (references)
  - Mechanism to distinguish between objects

# Object Example

## ■ Thermostat

### ■ State

- DesiredTemp
- CurrentTemp
- HeaterState

### ■ Behavior

- SetDesiredTemp()
- TurnHeaterOn()
- TurnHeaterOff()

### ■ Identity

- this

# Object Example

## ■ Thermostat

### ■ State

	Property	Value
■ DesiredTemp	integer	78°
■ CurrentTemp	integer	72°
■ HeaterState	boolean	ON

# Object State

## ■ Properties

- Static, unchanging
- May view as types

## ■ Values

- Dynamic, changes
- Within bounds set by properties

# Object Behavior

- **Methods**

- Procedures associated with object

- **Specify behavior of objects**

- **Invocation ⇒ sending message to object**

- **Example**

- `myThermostat.setDesiredTemp(78)`
- `myThermostat.turnHeaterOn()`
- `myThermostat.turnHeaterOff()`

# Method Types

## ■ Accessor

- Return state information

## ■ Mutator

- Modify state information

## ■ Constructor

- Create & initialize new object

## ■ Destructor

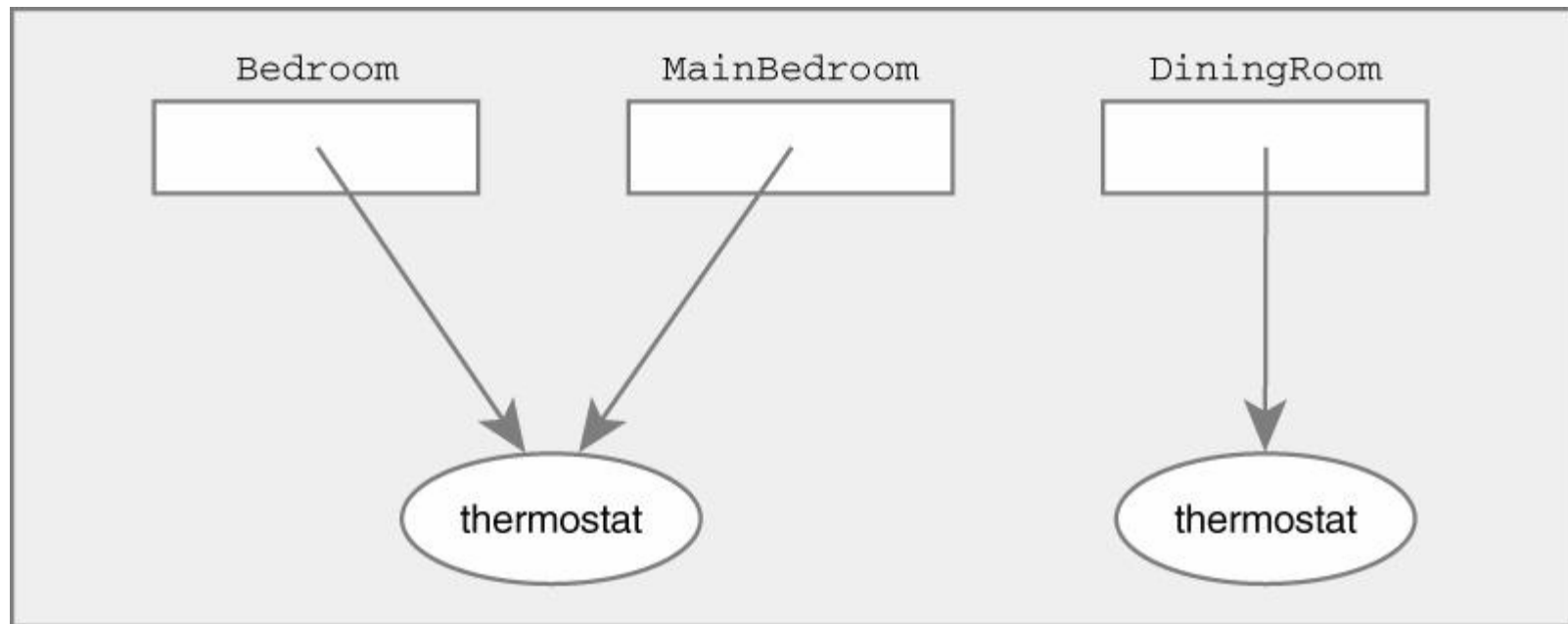
- Remove object & free up resources

# Object Identity

- **How to distinguish between objects**
- **Reference variables**
  - **Used in object-oriented programming languages**
  - **Points to objects**
  - **Multiple variables may point to same object**

# Reference Variables

## ■ Example



# Identity

## ■ Equivalence

- Whether two objects are equal

## ■ Name equivalence

- Reference variables point to same object

## ■ Content equivalence

- Objects from same class
- State in each object are identical

# Equivalence

## ■ Example

Name Equivalent

Content Equivalent

