CARLSON SCHOOL OF MANAGEMENT
UNIVERSITY OF MINNESOTA

Object-Oriented Programming

Carlson School of Management

Outline

- Why Object-Oriented Programming?
  - Procedural Programming
  - Object-Oriented Programming
- Classes and Objects
  - Defining a class: attributes and methods (private vs. public)
  - Encapsulation (data hiding)
  - Creating (constructing) an object (instance) from a class
- Universal Modeling Language (UML)

Carlson School of Management

Procedural (Functional) Programming

- Focus
  - On creating functions (procedures) for solving specific tasks
- Advantages
  - Modularized design of a larger program (top-down approach)
  - More natural way of breaking the problem down (decomposition)
  - Better suited for step-by-step sequential types of problems
  - Relatively fast execution compared to OOP approach
- Disadvantages
  - Less well suited for user-driven problems with unpredictable execution paths
  - Limited code reuse, copy/paste modify approach
  - Complex code operating on unprotected data separated from functionality
  - Increasingly complex programs very difficult to manage
    - A change request may require checking and rewriting a significant portion of the code

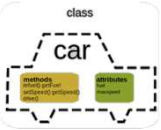
Carlson School of Management

Object Oriented Programming

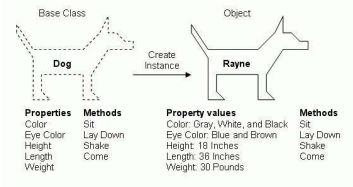
- Focus
  - On creating objects -> software constructs containing both data and procedures (functions)
- Advantages
  - Data are integrated and hidden parts of objects, exposed when needed
  - Internal data structure could be changed, the interaction with the data through methods stays the same
  - Object focused and more appropriate for user-driven problems with non-sequential execution paths
  - Significant code reuse, higher reliability and extensibility
  - Improved development productivity, maintainability, speed, lower cost and higher quality
- Disadvantages
  - Generally slower execution compared to functional approach
  - Relatively steep learning curve with complex designs

Carlson School of Management

Defining a Class (Conceptual)

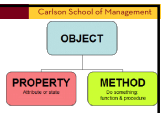


- A template (blueprint) for creating objects
  - Provides a list of attributes (data)
  - Implements methods (actions/behaviors)
- Classes: nouns
  - Dog (an abstraction)
- Attributes: adjectives
  - Color, height, weight
- Methods: verbs
  - Sit, come
- Object
  - An instance of a dog class (ayne object)



Carlson School of Management

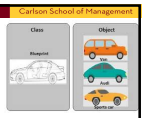
Examples of Objects



- Student
  - Attributes: name, SSN, midterm and final exam grades
  - Methods: **calc_sem_grade** – calculate semester grade
- Textbook
  - Attributes: name, author, quantity in stock, wholesale price
  - Method: **calc_retail_price** – calculate selling price
- Loan
  - Attributes: interest rate, term, amount borrowed
  - Method: **calc_mth_pmt** – calculate monthly payment

Carlson School of Management

Defining a Class (Technical)



- Class – a template from which objects created
  - Specifies attributes and methods in common to all objects

```
class Car:
```

  - `__init__` initializer method defining `color`, `make`, `speed`, `direction`, `mph`, … attributes for `self` reference

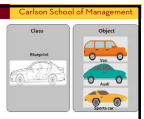
```
self.color = 'Green'
```

  - Functions defining `drive`, `turn`, `park`, `break`, … methods

```
def turn(self, dir):
  self.direction = dir
```

Carlson School of Management

Using Attributes and Methods



- Object – an instance of a class

```
audi = Car()
```

  - An object created in memory
  - `__init__` method automatically executed and `self` parameter automatically references the object just created
- Reassign different value to a public attribute (writing)

```
audi.color = 'Red'
```

- Retrieve value of a public attribute (reading)

```
mph = audi.mph
```

- Carry out a method (initiate behavior)
  - No need for `self` parameter when a method is called

```
audi.turn(dir)
```

Carlson School of Management

Loan Class: Attributes & Methods

- Class attributes (data, properties, characteristics)
  - Interest rate, term, amount borrowed (hard-coded)
  - Default object constructor: `__init__(self)`
    - Automatically called when object is instantiated (created)
- Class methods (actions, behaviors, functionality)
  - Calculate monthly payment ($2,026.74)

```
def calc_mth_pmt(self):
```

- **Lect10_Loan_Objects.py**

```
my_mortg = Loan()
```

Carlson School of Management

Loan Class: Data Encapsulation

- Store class definitions in a separate module (**loan.py**)
- Data Encapsulation – Hiding Attributes
  - Stored as private attributes starting with __ (2 underscores)
  - Cannot be accessed directly from the code
  - Indirect access through **Accessor** and **Mutator** methods
- Accessor methods
  - Gets the value of the attribute (read-only)
  - Provides safe way to access data from outside the class
- Mutator methods
  - Sets the value of the attribute (write-only)
  - Additional code validates data before assigning it to attribute

Carlson School of Management

Loan Class: Object Constructors

- Non-default object constructors: special initializer method
  - `__init__(self, other parameters)`
  - Typically used to initialize the default values for attributes

- Object instantiation and use

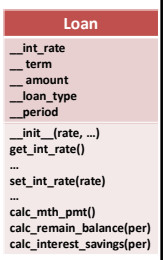
```
my_loan = loan.Loan(rate, years, amt, l_type)
mth_pmt = my_loan.calc_mth_pmt()
```

  - $400,000 for 30 years at 4.5% has $2,026.74 monthly payment
  - Refinance $300,000 for 15 year at 4% to get $2,219.74 payment
- **loan1.py**
  - Additional attribute: current payment period (180, half-way through)
  - Additional methods: calculating remaining balance ($264,935.82) and interest savings ($99,877.60)

Carlson School of Management

Universal Modeling Language (UML)

- Designing complex object-oriented systems
  - Beyond the scope of this class
  - Represent classes with UML diagrams
- Identify all the needed classes
  - Loan, Credit, Borrower, …
- See section 10.4
  - Define problem domain
  - Identify all the nouns as potential classes
  - Refine / reduce list to relevant nouns
  - Identify class responsibilities
    - Things class is responsible for knowing: data attributes
    - Things class is responsible for doing: actions (methods)



Carlson School of Management

Summary

- Defined the concept of classes as object templates with attributes and methods
- Explained the pros and cons of both procedural and object-oriented programming approaches
- Defined Loan class with attributes and methods
  - Extended the class with additional attributes and methods
- Defined Credit class that contains Loan class
  - A loan object of Loan class is an attribute of the Credit class
- Defined Borrower class also containing Loan class
  - A dictionary of loan objects of Loan class is an attribute of Borrower class
- A list of Borrower objects
  - Each element in the list is an object of Borrower class
- Brief mention of Universal Modeling Language (UML)