STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

SSW 322: Software Engineering Design VI

Object Oriented Paradigm
2020 Spring

Prof. Lu Xiao
lxiao6@stevens.edu
Babbio 513
Office Hour: Monday/Wednesday 2 to 4 pm
Software Engineering
School of Systems and Enterprises

Today's topics

- Shift in responsibility
- Abstraction
  - Abstract Class vs. Concrete Class
  - Inheritance/Realization
- Modularization/Information Hiding
  - Data Encapsulation
  - Polymorphism
  - Coupling/Cohesion

Software Design – Find a Solution

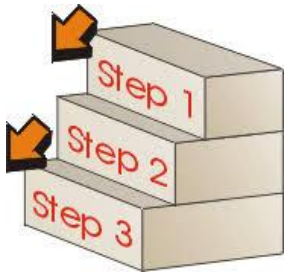
Consider the example of an instructor at a conference. Student in your class have another class to attend following yours, but don’t know where it is located. You are responsible to make sure everyone knows how to get to the next class.

A Nature Solution

Get a list of student in the class

For each student

1. Find the next class she/he is taking
2. Find the location of that class
3. Find the way to get from your classroom to the student's next class
4. Tell the student how to get to his or her next class.



A Procedural Solution

To do this would require the following procedures:

1. A way of getting the list of people in the class
2. A way of getting the schedule for each person in the class.
3. A program that gives someone directions from your classroom to any other classroom
4. A control program that works for each person in the class and does the required steps for each person

Is that reality?

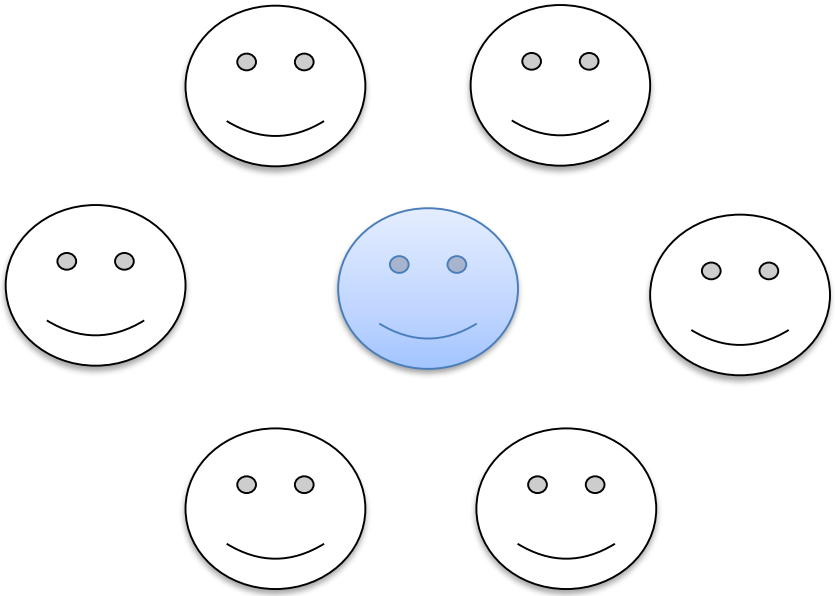
- Probably not.
- Second solution:
  - You would post directions to go from this classroom to all other classrooms
  - Tell everyone in the class what you did
  - Students use the directions to go to their next class.
- This is a philosophical difference. It's a ***shift in responsibility***.

The OO Approach to the Problem

1. Start the control program
2. Instantiate the collection of students in the classroom
3. Tell the collection to guide the students to their next class
4. The collection remind each student to go to his or her class
5. Each student
    - Finds where his next class is
    - Determines how to get there
    - Goes there
6. Done

Shift In Responsibility

- First Method: Responsibility is solely with the instructor, thus you, thus leads to insanity.
- Second Method: Responsibility is divided amongst others.



The Essene of OO Design

1. Shift in Responsibility

The Object-Oriented Paradigm

- Centered on the concept of the ***object***.
- What is an object?
  - Traditionally: data with method.
  - Better Definition: Things responsible for themselves.

The Object-Oriented Paradigm

- Conceptual View of an Object:
  - An object is a set of responsibilities.
- Specification View of an Object:
  - An object is a set of methods (behaviors) that can be invoked by other objects or itself.
- Implementation View of an Object:
  - An object is code and data and the computational interactions between them.

The Object-Oriented Paradigm

- An object's interface is its collection of methods.

- A class has the following:
  - The ***data*** elements the object contains
  - The ***methods*** the object can do
  - The way these data elements and methods can be accessed

What happens if requirements change?

- This works great with students, but if the student *class* was called undergrad student, then I add a grad student, I would have a problem
  - thus there is a need for an ***abstract class***.
- Therefore, I need a general type that encompasses more than one specific type. I can have a *student* class that includes both an ***undergrad*** student and a ***graduate*** student.

The OO Approach to the Problem

- Say we have a new type of student, i.e. a grad student.
- FIRST METHOD: The control program would have to be modified to distinguish grad students from undergrad students.
- SECOND METHOD: People are responsible for themselves, therefore the change would be ***localized*** to a routine for the new duty of the grad student.
  - The control program does not change!

OO Elements-Classes/Objects

- Abstract classes define what other, related classes can do.
- These “other” classes are classes that represent a particular type of related behavior.
  - Such a class is called a concrete class.
  - Therefore, *UndergradStudent* and *GraduateStudent* would be concrete classes.

The Essence of OO Design

2. Abstraction

OO Elements--Relations

- The relationship between an Undergrad-Student and a Student class is called a ***is-a relationship***.
- An is-a relationship is a form of ***inheritance***.
- Abstract classes act as placeholders for other concrete classes.
- Abstract classes define the methods their derived classes must implement.

Inheritance: is a relationship

- Inheritance allows us to derive a new class from an existing one.
  - A class ***GraduateStudent*** may inherit from a class ***Student***: a Graduate Student is-a Student
  - A class ***Manager*** may inherit from a class ***Employee***: a Manager *is-a* Employee
  - A class ***SavingsAcctount*** may inherit from a class ***BankAcct***: an SavingsAcct *is-a* BankAcct

What does the subclass inherit?

- The subclass inherits the data and methods defined for the superclass.
- Inherited variables and methods can be used in the derived class, as if they had been declared locally.
- The subclass can override (redefine) inherited method.
- Thus, in defining the new, derived class, all we need to do is *define the data and methods that are specific to the super class*.

The Object-Oriented Paradigm

- Three levels of accessibility:
  - Public – Anything can see it
  - Protected – Only objects of this class and derived classes can see it
  - Private – Only objects from this class can see it

The Object-Oriented Paradigm

- Java and C++ have reserved key words for this.
- All of this is a matter of culture and convention in Python.
  - Protected: prefixing the name of your member with a single underscore,
  - Private: prefixed with at least two underscores and suffixed with at most one underscore

The Object-Oriented Paradigm

- Encapsulation is not just about data, it's about any kind of hiding!
- In our previous examples, the type of student is hidden from the control program
  - In essence, the control program has to instantiate it and thus know what type of object it is.
- This encapsulation is known as polymorphism.
- Polymorphism can be defined as the ability to refer to different derivation of a class in the same way, but getting the behavior appropriate to a subclass

Polymorphism

- Polymorphism means "[able to assume] many (poly) forms (morphism)".
- A polymorphic reference is one that can refer to different types of objects, at different times.

Real-life Example

- You know how to **open** a ***door***
- Under OOP point of view, you are an ***object*** (sorry, no humiliation intended)
- Abstract: you are capable of ***open something***
  - ***In essence:*** interacting with a rectangle rotating on hinges
- ***Polymorphism of Open:***

—> You can open a ***window***

—> You can open a ***car door***

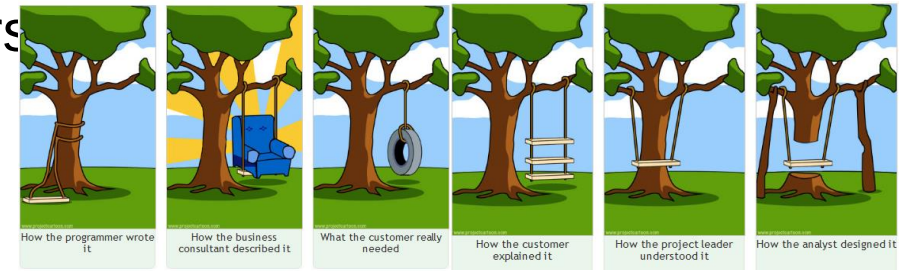
—> You can open a ***book***

Does OO ease programming?

- Doesn’t necessarily make the initial coding easier.
- Instead, it reduces the effort to maintain code in the long run.
- This should not be trivialized
  - writing a program that works the first time is easy.
  - writing a program that continues to meet the needs of its users is quite another challenge.
- Many bugs originate with changes to code.

Why do programs fail in the “real world”?

- In academics
  - The problem you are solving is clear
  - The expectations are well defined.
- This is not the case in the real world. A project can fail because:
  - Technological problems
  - Incompetent Programmers
  - Poor Management
  - Poor Development Tools



- The #1 reason projects fail is poor requirements gathering.

The Reality

- Why are requirements poor?
    - Requirements are incomplete
    - Requirements are often wrong
    - Requirements (and users) are misleading
    - Requirements do not tell the whole story
- While one tries to document requirements completely and properly, the reality is, for many reasons you will not achieve a perfect set of requirements.
- If you do not have good requirements, how can programmers possibly be expected to develop an application that meets the needs of the end user.
- The reality is: Requirements Always Change!!

How do you handle change?

- Example of change: bug is discovered!
- We really do not spend much time fixing the bug
- Instead, we spend a lot of time:
  - Discovering how the code works
  - Finding the bug
  - Taking time to avoid unwanted side effects when we correct the code

Battling the change problem

- The best weapon is **modularity**
  - modular code isolates the effect of change
  - makes the code more maintainable

- Modularity = resilience to change

- How do you measure modularity?

Good OO Design

- The most important promise of OO is improve modularization
  - that is, enhance maintainability
  - while still supporting functional decomposition (which is good for cohesion)
  - while also promoting re-use

- Good OO = high cohesion + low coupling

Coupling and Cohesion

- Cohesion refers to how “closely the operations in a routine are related.”

- Coupling refers to “the strength of a connection between two routines.”

Software Modularity

- So do we want our code to have weak or strong cohesion?

- Do we want our code to have tight or loose coupling?

- Code should have ***strong cohesion*** and ***loose coupling***.

The Essence of OO Design

3. Modularization

Software Modularity

The main goal (almost not evident) of design is to minimize coupling and maximize cohesion.

Coupling is the level of interdependency between a method and its environment, (other methods, objects and classes), while Cohesion is the level of uniformity of the method goal.

Yourdon, E.; Constantine, L L. (1979). *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*, copyright 1979 by Prentice-Hall, Yourdon Press.

In Class Exercise

You are a developer for the back end of a banking system.

Problem: Your bank offers different types of credit cards with different discounts when shopping at groceries. The basic credit offers 0 discount. There are *two special credit card types*: the bronze credit card offers 0.1 discount, while the gold credit card

offers 0.2 discount rate.

Discuss: How do you design this part of your system?

Hint

1. You need a “CreditCard” class
2. Your “CreditCard” class knows how to calculate the final purchase amount based on its discount rate.
3. Keep in mind, your bank may offer other types of credit cards in the future.

Two Solutions

```
class CreditCard:

    cardType=0

    def __init__(self, id, type):
        self.cardType = type

    def getCardType(self):
        return self.cardType

    def getDiscount(self, monthlyCost):
        if self.cardType == 1:
            return monthyCost * 0.02
        if self.cardType == 2:
            return monthyCost * 0.01
```

```
class CreditCard:

    def getDiscount(self, monthlyCost):
        return 0

class BronzeCreditCard(CreditCard):

    def getDiscount(self, monthlyCost):
        return monthlyCost * 0.01

class GoldCreditCard(CreditCard):

    def getDiscount(self, monthlyCost):
        return monthlyCost * 0.02
```

What's the key difference between the two solutions?

Two Solutions

```
class CreditCard:

    cardType=0

    def __init__(self, id, type):
        self.cardType = type

    def getCardType(self):
        return self.cardType

    def getDiscount(self, monthlyCost):
        if self.cardType == 1:
            return monthyCost * 0.02
        if self.cardType == 2:
            return monthyCost * 0.01
```

```
class CreditCard:

    def getDiscount(self, monthlyCost):
        return 0

class BronzeCreditCard(CreditCard):

    def getDiscount(self, monthlyCost):
        return monthlyCost * 0.01

class GoldCreditCard(CreditCard):

    def getDiscount(self, monthlyCost):
        return monthlyCost * 0.02
```

What if we want to add a platinum card with discount rate 0.3?

Two Solutions

```
class CreditCard:

    cardType=0

    def __init__(self, id, type):
        self.cardType = type

    def getCardType(self):
        return self.cardType

    def getDiscount(self, monthlyCost):
        if self.cardType == 1:
            return monthyCost * 0.02
        if self.cardType == 2:
            return monthyCost * 0.01
```

```
class CreditCard:

    def getDiscount(self, monthlyCost):
        return 0

class BronzeCreditCard(CreditCard):

    def getDiscount(self, monthlyCost):
        return monthlyCost * 0.01

class GoldCreditCard(CreditCard):

    def getDiscount(self, monthlyCost):
        return monthlyCost * 0.02
```

What if we want to add a platinum card with discount rate 0.3?

Modifications should be done by adding new code and incorporating this new code in the system in ways that does not require old code to be changed!

A Closing Question

What is software made for?

Software Is Made for Change!

- New requirements
- Fix bugs
- Update technology
- …


Thank
you