CptS 487
Software Design and Architecture

Lesson 2

OO Basics

WASHINGTON STATE UNIVERSITY
World Class. Face to Face.

Instructor: Bolong Zeng

Outline

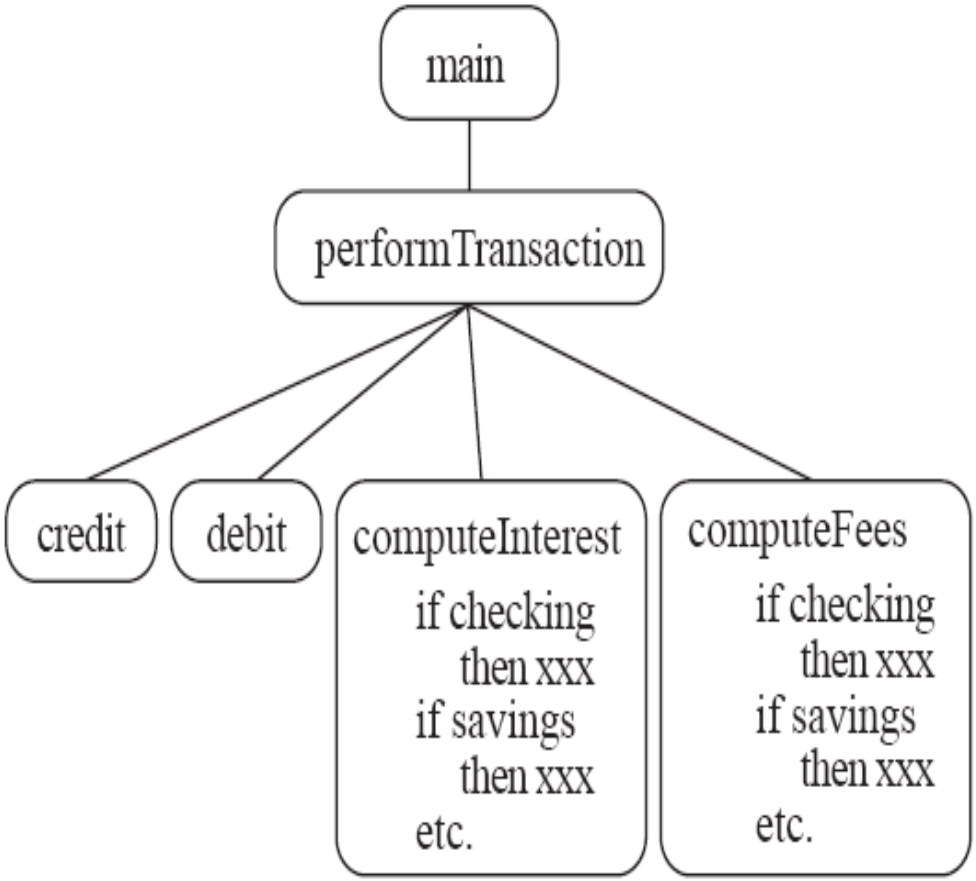
- Basic review of Object-Oriented concepts
  - And examples

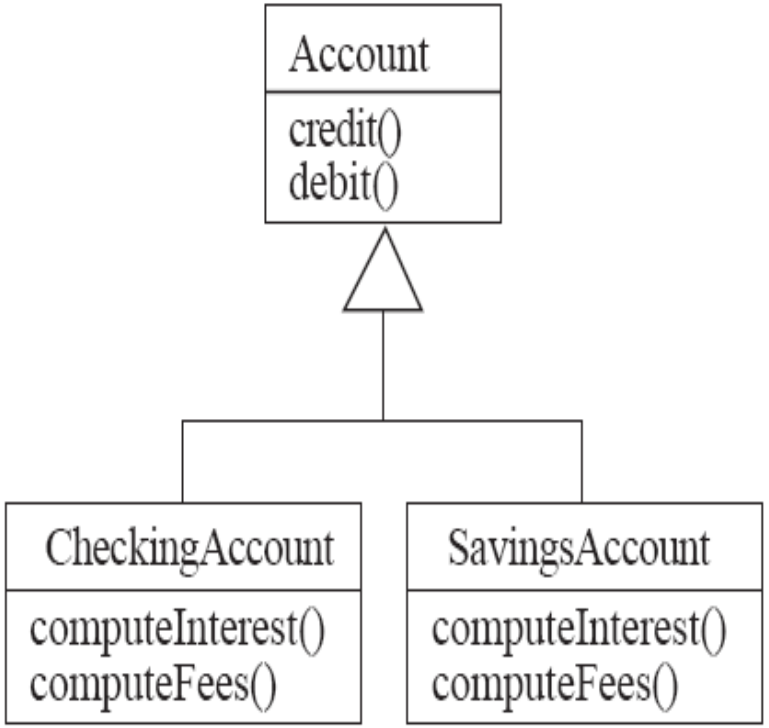
What is Object Orientation?

Procedural approach:

- Software is organized around the notion of *procedures*
- Procedural abstraction
  - Works as long as the data is simple
- Procedures send data to each other.
- Procedures and data are clearly separated.
- Functions are hard to reuse.

A View of the Two Approaches





What is Object Orientation?

Object oriented approach :

An approach to the solution of problems in which all computations are performed in the context of objects.

- Begin by modeling the problem domain as objects.
- The implementation is organized around objects.
- Related data and behavior are tied together.
- Objects send messages to each other.
- A running program can be seen as a collection of objects collaborating to perform a given task

Object Oriented Approach

- The objects are instances of classes, which:
  - are data abstractions
  - contain procedural abstractions that operate on the objects
- Powerful concepts:
  - Encapsulation, interfaces, abstraction, generalization, inheritance, delegation, responsibility-driven design, separation of concerns, polymorphism, design patterns, reusable components, service-oriented architecture, message-oriented middleware,…etc.

Classes and Objects

Object

- Is a chunk of structured data in a running software system.
- Represent *things*
- Has *properties*
  - Represent its state
- Exhibit *behavior*
  - How it acts and reacts
  - May simulate the behavior of an object in the real world
- Have *interfaces*.
- Have *identity*.
- Send *messages* to other objects.
- Should *encapsulate* their *state* and internal structures.
- Should not be complex or large.
- Should be self-consistent, *coherent*, and complete.
- Should be *loosely coupled* with other objects.

Encapsulation

Grouping data and functions together and keeping their implementation details private.

- Good fences make good neighbors.
- Exposing only the *public interface*.
- Protects the object from outside interference.
- Protects other objects from details that might change.
- Reduces complex interdependencies, promotes *loose coupling*.
- **Encapsulation Example:**
  - Best practice: Objects speak to each other by method calls not by direct access to attributes.

```
class Person {
   public int age; // yuk
}

class BetterPerson {
   private int age; // dateOfBirth ?
   public int getAge() { return age; }
}
```

Side Note

- Are getters and setters evil?
  - "Why Getter and Setter Methods are Evil"
    - https://www.javaworld.com/article/2073723/core-java/why-getter-and-setter-methods-are-evil.html
  - "Getters/Setters. Evil. Period."
    - https://www.yegor256.com/2014/09/16/getters-and-setters-are-evil.html

Classes

A class:

- A unit of abstraction in an object oriented (OO) program
- Represents similar objects
  - Its *instances*
  - An object is instantiated (created) from the description provided by its class
    - Thus objects are often called instances
- A kind of software module
  - Describes its instances' structure (properties)
  - Contains *methods* to implement their behavior

Is Something a Class or an Instance?

- Something should be a *class* if it could have instances
- Something should be an *instance* if it is clearly a *single* member of the set defined by a class

Film

- Class; instances are individual films.

Reel of Film

- Class; instances are physical reels

Film reel with serial number SW19876

- Instance of **`ReelOfFilm`**

Science Fiction

- Instance of the class **`Genre`**.

Science Fiction Film

- Class; instances include 'Star Wars'

Showing of 'Star Wars' in the Phoenix Cinema at 7 p.m.:

- Instance of **`ShowingOfFilm`**

Class Attributes and Methods

Classes contain two things:

- **Fields** (attributes, data members, class variables):
  - Data items that differentiate one object of the class from another. e.g. `name, dateOfBirth`
  - Each object has its own values for the attributes of its class
- **Methods** (behaviors):
  - Named, self-contained blocks of code that typically operate on the fields
  - Describe what an object can do
  - Each object shares the implementation of a class's methods and thus behave similarly
    - When a class is defined, a developer provides an implementation for each of its methods
    - This approach ensures that objects of the same class behave similarly

Summary

- Procedural vs. object oriented approach
- Object oriented approach : A running program can be seen as a collection of objects collaborating to perform a given task
- Object
  - Is a chunk of structured data in a running software system.
  - Represents things in real life
- Class:
  - A unit of abstraction in an object oriented (OO) program
  - Represents similar objects
- Encapsulation: Grouping data and functions together and keeping their implementation details private.

Additional OO Concepts

“static” attributes and methods

- Class wide

- A static attribute is shared among all instances of a class
  - Each object has the same value for the static attribute

- Static attributes are useful for:
  - Default or ‘constant’ values (e.g. PI)
  - Lookup tables and similar structures

- A static method does not have to be accessed via an object; you invoke static methods directly on a class
  - What are the usage of static methods?

Inheritance

- Subclasses have an "**IS-A**" relationship with their superclass.

- The subclass may provide additional ***state*** or ***behavior***, or it may ***override*** the implementation of ***inherited*** methods
  - Single inheritance systems (java) only allow a class to have one parent, or superclass. Multiple inheritance systems (C++) allow a class to have multiple parents.

- Example:
  - a **Lion** *is an* **Animal**, not vice-versa
  - A **County** *is a* **Municipality**, but it's not a **State.**

Inheritance

- The inheritance relationship between classes forms a class hierarchy
  - In models, hierarchy should represent the natural relationships present in the problem domain
  - In a hierarchy, all the common features of a set of objects can be accumulated in a superclass
- This relationship is also known as a generalization-specialization relationship
  - Since subclasses specialize (or extend) the more generic information contained in the superclass

Polymorphism

- Property of object oriented software by which an method may be performed in different ways in different classes.
- Method Overriding
  - A method would be inherited, but a subclass may define its own implementation of said method
- Method Overloading
  - Methods may have the same name, but with different parameters (type, number of arguments, etc.). Methods may be in the same class or among super/sub-classes.
- See here for a clear explanation:
  - http://stackoverflow.com/questions/154577/polymorphism-vs-overriding-vs-overloading

Interfaces and Abstract Classes*

An abstract class is simply one which cannot be instantiated, but can be subclassed.

- An abstract class serves as a basis (that is, a superclass) for a group of related subclasses.
- Abstract class enforces certain hierarchies for all the subclasses
- Abstract class defines an abstract interface (set of methods) that subclasses must implement
  - May provide code for some of the methods (enabling code reuse)

*C++ is slightly different

Interfaces and Abstract Classes

An interface is a type of class definition in which only method signatures are defined.

- An interface has no implementation; it only has the definition of the methods without the body.
- Similar to abstract class, it is a contract that is used to define hierarchies for all subclasses.
- However, a class can implement more than one interface but can only inherit from one abstract class.

Interfaces and Abstract Classes

	Interface	Abstract class
Multiple inheritance	A class may inherit several interfaces.	A class may inherit only one abstract class.
Default implementation	An interface cannot provide any code, just the signature.	An abstract class can provide complete, default code and/or just the details that have to be overridden.
When to use	If various implementations only share method signatures then it is better to use interfaces.	If various implementations are of the same kind and use common behaviour or status then abstract class is better to use.

Choosing between Int. and Abs. Class

- Recommendation from MSDN:

- If you anticipate creating multiple versions of your component, create an abstract class. Abstract classes provide a simple and easy way to version your components. By updating the base class, all inheriting classes are automatically updated with the change. Interfaces, on the other hand, cannot be changed once created. If a new version of an interface is required, you must create a whole new interface.

- If the functionality you are creating will be useful across a wide range of disparate objects, use an interface. Abstract classes should be used primarily for objects that are closely related, whereas interfaces are best suited for providing common functionality to unrelated classes.

- If you are designing small, concise bits of functionality, use interfaces. If you are designing large functional units, use an abstract class.

- If you want to provide common, implemented functionality among all implementations of your component, use an abstract class. Abstract classes allow you to partially implement your class, whereas interfaces contain no implementation for any members.

Note on Inheritance

- Easy to understand
- Practical and commonly used…

- But: don't over use it.*

*We'll explain this later eventually.

Object Identity

All objects have identity

- A property that allows us to distinguish one object from another
- Considering two cups from the same set of china, we might say that they are equal but not identical
- In object-oriented programming languages, all objects (i.e. instances) have a unique identifier
  - This identifier may be, for instance, the object's location in memory; or a unique integer that was assigned to it when it was created

Concepts that Define Object Orientation

The following are necessary for a system or language to be OO

- Identity
  - Each object is *distinct* from each other object, and *can be referred to*
  - Two objects are distinct *even if they have the same data*
- Classes
  - The code is organized using classes, each of which describes a set of objects
- Inheritance
  - The mechanism where features in a hierarchy inherit from superclasses to subclasses
- Polymorphism
  - The mechanism by which several methods can have the same name and implement the same abstract operation.