CSC 335 - Object-Oriented Programming

Dr. Jonathan Misurda
jmisurda@cs.arizona.edu

Programming Language History

- Programming languages abstract the actual function of the machine
- Originally computers were hardcoded to solve a single problem (1940s)
  - Eventually could have their program altered by changing their wiring or flipping switches
- Stored program (von Neumann architecture) used memory for code as well as data storage

Programming Language History

- Now we can represent instructions as data
  - Like all non-numeric quantities in a computer, code is numbered and stored as an integer (machine code)
- Machine code is hard to learn, so replace machine instruction numbers with small mnemonics
  - A program then can convert the mnemonics to the numbers (an Assembler)

Unstructured Programming

Commodore 64 BASIC

```
10 DIM A(10): N=9
11 FOR I=0 TO N: A(I)=INT(RND(1)*10)+1: NEXT
12 GOSUB 50
20 FOR J=1 TO N: KEY=A(J): I=J-1: GOSUB 30: A(I+1)=KEY: NEXT
25 GOSUB 50: END
30 IF I=-1 THEN RETURN
31 IF A(I)>KEY THEN A(I+1)=A(I): I=I-1: GOTO 30
32 RETURN
50 PRINT: FOR I=0 TO N: PRINTA(I): NEXT: RETURN
```

Structured Programming

C

```c
#include <stdio.h>

void insertion_sort(int a[], int n) {
  for(int i = 1; i < n; ++i) {
    int tmp = a[i];
    int j = i;
    while(j > 0 && tmp < a[j - 1]) {
      a[j] = a[j - 1];
      --j;
    }
    a[j] = tmp;
  }
}

int main () {
  int a[] = {4, 65, 2, -31, 0, 99, 2, 83, 782, 1};
  int n = sizeof(a) / sizeof(a[0]);
  int i;
  for (i = 0; i < n; i++)
    printf("%d%s", a[i], i == n - 1 ? "\n" : " ");
  insertion_sort(a, n);
  for (i = 0; i < n; i++)
    printf("%d%s", a[i], i == n - 1 ? "\n" : " ");
  return 0;
}
```

Structure Programming Limitations

1. It's rarely possible to anticipate the design of a completed system before it is implemented.
   - The system must be restructured as it grows.
   - The larger the system, the more restructuring takes place.
2. Software development had focused on the modularization of code, rather than data
   - The data was either moved around between functions via argument/parameter associations, or
   - The data was global.
     - Works okay for smaller programs or for big programs when there are few global variables.
     - Not good when variables number in the hundreds.

Modular Programming

- Break large-scale problems into smaller components that are constructed independently
- Sharing data (global variables) is a violation of modular programming
  - Sharing data makes all modules dependent on one another, which is dangerous.
- An improvement:
  - Give each subroutine its own local data
  - This data can only be "touched" by that single subroutine
  - Subroutines can be designed, implemented, and maintained independently
  - Other necessary data is passed amongst the procedures via argument/parameter associations

Object

- OO began with the Simula 67 language ("simulation language")
- Designed to build accurate models of complex working systems
- The modularization occurs at the physical object level (not at a procedural level)

Object-Orientedness is a World View

It is a way of looking at a problem and decomposing it to model within a computer system

SPD versus OOPD

- Structured Programming and Design
  - Systems are modeled as a collection of functions and procedures that pass data all over the place.
- Object-Oriented Programming and Design
  - Systems are modeled as a collection of interacting objects where the data is encapsulated with the methods that need those values.

Modularity

- A complex software system should be decomposed into a set of highly cohesive but loosely coupled modules.
- A system is **modular** when:
  - each activity of the system is performed by exactly one component, and
  - when the inputs and outputs of each component are well-defined.
- A component is **well-defined** if:
  - all inputs to it are essential to its function, and
  - all outputs are produced by one of its actions.

Software Engineering Theory and Practice 3rd Ed. by Pfleeger and Atlee (2006)

The Six 'C's

- **Cohesion**: Do the things in your class go together?
- **Completeness**: Are any major concepts missing? Does this class represent just a partial concept?
- **Convenience**: Is the class easy enough to use?
- **Clarity**: Can someone else understand the class?
- **Consistency**: Are the naming conventions and behavior the same between all classes and methods?
- **Coupling**: Are many classes tied to each other?
  - Coupling is bad; the rest of the above are good!

What is an Object?

- State
  - Represented by object's fields
- Behavior
  - Represented by object's methods
- Identity
  - Represented by object itself, and its ability to distinguish itself from similar objects

Abstraction

- **Abstraction**: separating the essential from the nonessential characteristics.
- The functionality of a module should be characterized in a succinct and precise description known as the contractual interface of the module.
- The contractual interface captures the essence of the behavior of the module.
- A class can be viewed as a service provider and other classes are clients.

Encapsulation

- **Encapsulation**: The implementation of a module should be separated from its contractual interface and hidden from the clients of the module.
- We need an efficient way to redefine the same thing many times, and hide the details from other objects.
- The class defines the behaviors and attributes.
- Each object (instance of the class) has its own state—the set of values for that instance.

Polymorphism

- **Polymorphism**: an entity with multiple forms.
- A single name can represent different behaviors based on the type of the object.
- Polymorphism is possible
  - via Java interfaces, e.g., compareTo
  - or via inheritance, e.g., toString
- New classes can be derived from existing ones through the inheritance.
  - The "top" class is called the superclass (or base class).
  - The new class in called a subclass (or derived class).

"Ad Hoc" Polymorphism

- We can also do **method overloading** in Java
- Overloaded methods have the same name, but different signatures:

```
int abs(int x);
double abs(double x);
```

Inheritance

- The superclass should pull together the common characteristics.
- The subclasses inherit methods and attributes.
- Subclasses add new data (called fields) and methods.
- Subclasses can override some methods to give the method new meaning.
  - In Java, you often override these methods of class Object:
    - toString
    - equals

Inheritance Hierarchies

- Classes can be extended to any degree
- Each new derived class accumulates the data and behavior (methods) of its ancestors
- The base class captures common data and behavior
- This naturally generates a tree structure, with the most abstract class as the root (at the top) and the most detailed derived classes as the (possibly many) leaves

Summary

- Large scale, extensible, correct, efficient, and maintainable code is extremely difficult to write.
- Many concepts have evolved over the last forty years.
- Some of the best concepts are now part of modern programming languages and design methodologies.
- There now exist accepted principles for designing quality code.