Object-Oriented Design Strategies

20 Jan 2010
CMPT166
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

Method overloading

- Overloading is giving multiple definitions for a method with the same name, but different signature: num arguments or argument types

```
public int square( int x ) {
    return x*x;
}
public double square( double x ) {
    return x*x;
}
int y=5; double z=2.3;
square(y); square(z)
```

  - Do we need a float version as well?

Default parameter values

- Overloading is Java's way of letting you specify default values to parameters:

```
public class Student {
        String name;
        int ID;
        public Student( String name, int ID ) {
                this.name = name
                this.ID = ID
        }
        public Student() {
                name = “Joe Smith”;
                ID = 1001;
```

Object-oriented design

- Writing software is not just about the code!
- It is an intentional process including:
  - Client interviews to develop a problem statement and plan
  - Software design (charts, algorithms, etc.)
  - Coding
  - Testing
  - Maintenance, documentation

OO design is NOT:

- OO design is not based on:
  - Language syntax
  - Implementation details
  - Platform considerations
  - Manipulation of global entities
  - OO language features
    - Don't do something just because the language lets you!

OO design IS:

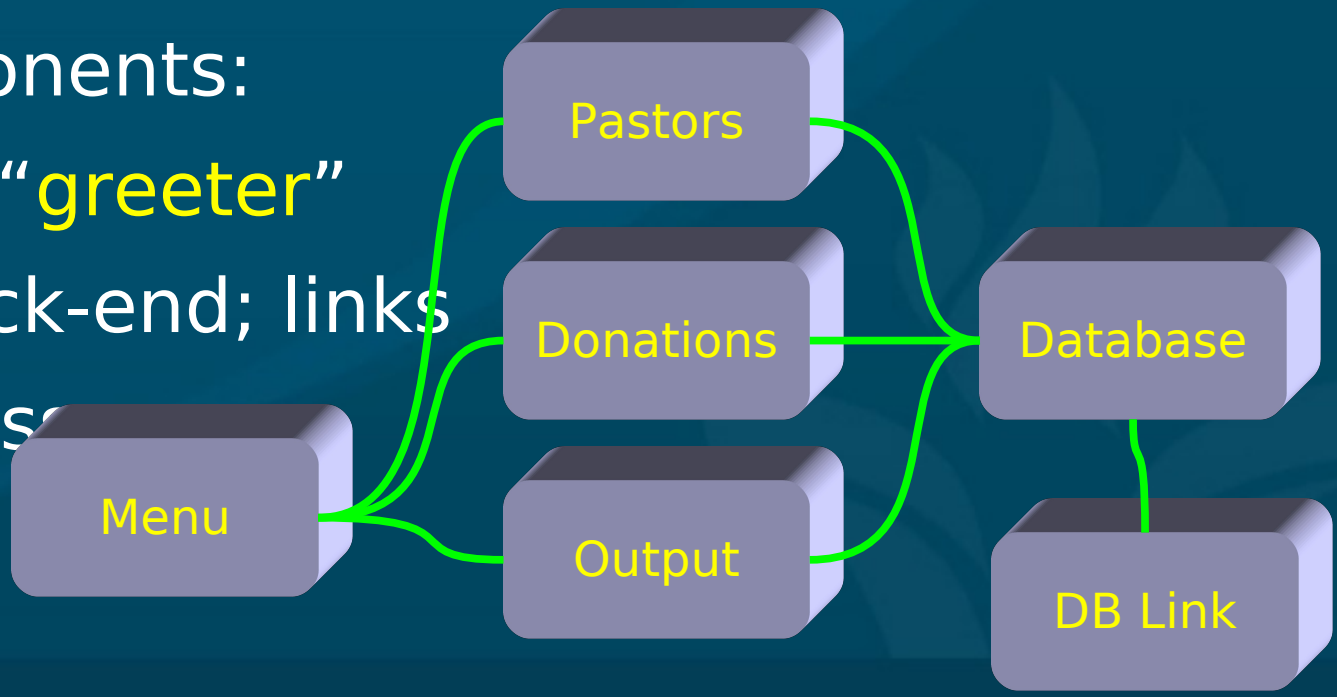
- OO design is based on:
  - Delegation of responsibility
    - No monolithic code block does everything
  - Independence of objects
    - Not connected via globals: simplifies testing!
    - Not supervised elsewhere
  - Security of state (stored data values)
    - private/public
  - Portability, reusability
    - Abstract platform details
    - Use general design principles

Steps in OO design: 1

- Describe overall system behaviour
  - Write for the non-technical end-user
  - User interface: look and feel
  - Not about data structures, classes, methods, ..
- e.g., Church Information Manager (CIM):
  - database of members and affiliates
    - data entry on a simple form
    - public access to basic info
    - protected access to confidential information
      - Pastor's notes; financial information; etc.
    - Create church directory

Steps in OO design: 2

- Refine behavioural description into components
  - Each component holds a set of related tasks
  - Components isolated, self-contained!
  - Components have thinly-coupled interactions
- e.g., CIM components:
  - Main menu / "greeter"
  - Database back-end; links
  - Pastors' acces
  - Donations
  - Output



Factoring into components

- Suggestion: use 3x5” index cards, one for each component
  - Name of component
  - Primary responsibility
  - Collaborating components
- If it won't fit on a 3x5” card, it's too complex to implement!
  - Break it down into smaller components
- Write down every design decision, w/ pros/cons
- Postpone implementation detail decisions

Steps in OO design: 3

- From components to classes:
  - Each component may have many class types
  - Each class defines:
    - Behaviour (methods)
    - Stored state (instance variables)
  - Behaviour is common to all instances
  - State is unique to each instance
- Principle of least privilege:
  - Provide only enough information to clients to achieve desired behaviour, nothing more!

Writing classes

- Design your data structures and relationships
  - Person: name, birthdate, link to Household
  - Household: phone, address, link to Persons
- Basic methods for each class:
  - Display and edit its own information (set/get)
    - Access restrictions
    - __str__() or toString() method for debuggging
  - Initializer/constructor: set default values
- Helper classes (support components)
  - Only for one class; hidden to rest of world

Top-down coding

- Start with the basic user-interface
  - Event-driven GUI: user clicks --> call method
  - Stub callbacks: fill in functionality later
  - Stub methods: return default values
- Incremental testing
  - Test each component before moving on!
  - May need to write small separate testbed programs
- Integration testing (regression testing)
  - Test interaction between components

Source control, build control

- Source control (e.g., Subversion):
  - Central repository for all code, and changes
  - Programmers check out components
  - When revisions are tested and safe, check-in commits changes back to the repository
  - Concurrent revisions: may need to merge with other programmers' changes
    - Importance of thinly coupled components
    - Each component has one project leader
- Build control: automated regression testing, multiplatform compilation