Agent-Oriented Programming: Intro

Presenter:

Leigh Tesfatsion
Professor of Economics
Courtesy Professor of Mathematics
Iowa State University
Ames, Iowa 50011-1070
26 February 2008
https://www2.econ.iastate.edu/tesfatsi/
tesfatsi@iastate.edu
Outline

- What is Object-Oriented Programming (OOP)?
- Agent-Oriented Programming (AOP) vs. OOP
- AOP via “Computational Laboratories”
- Example: The Trade Network Game (TNG) Computational Lab
Object-Oriented Programming (OOP)

KEY CONCEPTS:

★ Object
  - Methods (behaviors, functions, procedures, …)
  - Attributes (data, state information, …)
  - Access: public, private, or protected

★ Class

★ Interface

★ Encapsulation

★ Inheritance (subclass, superclass)

★ Composition
Object-Oriented Programming (OOP)

- An **object** is a software entity containing **attributes** plus **methods** that act on these attributes.

- An object controls **access** to its attributes and methods by declaring them
  - public (accessible to all other objects);
  - private (inaccessible to all other objects);
  - or protected (accessible only to certain designated other objects).

- A **class** is a blueprint for an object, i.e., a template used to create (“instantiate”) an object.
Class = Object Template

Class Employee

Employee Objects (Instances of Employee)

Ann
Ping
Mario
Dan
Illustration: Employee Class
(See M. Weisfeld book cited on Syllabus)

Class EMPLOYEE
{
    Public Access:

    Methods:
    - getSocialSecurityNumber();
    - getGender();
    - getDateOfBirth();

    Private Access:

    Attributes:
    - SocialSecurityNumber;
    - Gender;
    - DateOfBirth;
    - Trustworthyness;
}
The public methods and public attributes of an object are called the **interface** of the object.

Objects *communicate* with each other via their public methods, i.e., by activating (“invoking”) the public methods of other objects.
In “good” OOP design, an object should only reveal to other objects what these objects need to know to interact with it.

Each class template specifies the interfaces for its instantiated objects -- it completely describes how users of these instantiated objects can interact with these instantiated objects.
Illustration: Employee Class
(See Matt Weisfeld book cited on Syllabus)

Class EMPLOYEE
{
    Public Access:
    
    Methods:
    
    getSocialSecurityNumber( ) ;
    getGender( ) ;
    getDateOfBirth( ) ;

    Private Access:
    
    Attributes:
    
    SocialSecurityNumber ;
    Gender ;
    DateOfBirth ;
    Trustworthyness ;
}

Illustration: Payroll Class
(involves public methods in Employee class)

Class PAYROLL
{
  Public Access:
  Methods:
  calculateEmployeePay( );
  payEmployee( );
  Employee.getSocialSecurityNumber( );
  Employee.getGender( );
  Employee.getDateOfBirth( );

  Private Access:
  Attributes:
  CurrentProfits;
  EmployeePayroll;
}
Encapsulation is the process of determining which aspects of a class are not needed by other classes, and hiding these aspects from other classes.

More precisely, encapsulation is the process of dividing each class of a program into two distinct parts:

1. (public) interface;
2. private (or protected) stuff that other classes do not need to know about.
A class C can *inherit* the attributes and methods of another class B.

The class C is then called the *subclass* of class B, and class B is called the *superclass* of class C.

A subclass can also include specialized attributes and methods that are not present in the superclass.
Class Inheritance: Example

TradeBot
attribute
Price;
method
trade();

Buyer
Price = BidPrice;
trade() = buy();
calculateUtility();

Seller
Price = AskPrice;
trade() = sell();
calculateProfits();

Superclass of Buyer and Seller

Subclass of TradeBot

Subclass of TradeBot
Composition vs. Inheritance

* Objects can be built, or “composed”, from other objects. This is called *composition*.

**Example:** A firm is composed of employees.

* A *composition* relationship between objects is often termed a “Has-A” *relationship*. A firm “has an” employee.

* An *inheritance relationship* between objects is often termed an “Is-A” *relationship*. A buyer “is a” trader.
AOP vs. OOP

- What is an agent?

How does Agent-Oriented Programming (AOP) extend conventional Object-Oriented Programming (OOP)?
What is an *Agent*?

According to Jennings (2000), an agent is an object capable of displaying...

- **(Structural) Reactivity:** Changes in internal structure in response to environmental changes.
- **Social Ability:** Interaction with other agents through some form of language.
- **Pro-Activity:** Goal-directed actions.
- **Autonomy:** Some degree of control over its own actions (“self-activation”).
Key Distinction is *Autonomy*

- Distributed control, *not* just distributed actions.
- According to Jennings, conventional objects encapsulate attributes and methods but not *self*-activation and *localized* action choice.
Autonomy means...

- Each agent effectively has its own persistent thread of control.
- Each agent decides for itself which actions to perform at what time, based in part on external environmental conditions and in part on private internal aspects (current beliefs, desires,…).
- Thus, in multi-agent systems, a potential source of uncertainty for each agent is not knowing for sure what other agents will do (called “behavioral” or “strategic” uncertainty).
## Example: Worker Agent

### Public Access:

// Public Methods
- Protocols governing job search
- Protocols governing negotiations with potential employers
- Protocols governing unemployment benefits program
- Methods for retrieving Worker data

### Private Access:

// Private Methods
- Method for calculating my expected utility assessments
- Method for calculating my actual utility outcomes
- Method for updating my worksite strategy (learning)
- Methods for updating my methods (learning to learn)

// Private Attributes
- Data about myself (my history, utility fct., current wealth...)
- Data recorded about external world (employer behaviors,...)
- Addresses for potential employers (permits communication)
AOP via Computational Laboratories

- **Computational Laboratory** = Computational framework for the study of complex system behaviors by means of controlled and replicable experiments.

- **Graphical User Interface (GUI)** permits experimentation by users with no programming background.

- **Modular/extensible form** permits framework capabilities to be changed/extended by users who have programming background.
Example: The Trade Network Game Lab (TNG) Laboratory

- Evolution of trade networks among strategically interacting traders (buyers, sellers, and dealers)

- Traders are instantiated as “tradebots” (autonomous software entities with internal attributes and methods)

- The tradebots engage in event-driven communication

- The tradebots evolve their trade methods over time, starting from initially random trade methods
TNG Lab Architecture

- Four-Layer Architecture:
  - SimBioSys (C++ class framework)
  - TNG/SimBioSys (extension classes)
  - TNG/COM (permits interactive display)
  - TNG Lab (graphical user interface)

- Downloadable as Freeware (Zip file includes automatic installation wizard)
  www2.econ.iastate.edu/tesfatsi/tnghome.htm
TNG Lab 4-Layer Architecture
(McFadzean, Stewart, and Tesfatsion, IEEE-TEC, 2001)

- TNG Lab
- TNG/COM
- TNG/SimBioSys
- SimBioSys class framework
SimBioSys (McFadzean, 1995)

- Simulation toolkit
- C++ class library
- Designed for artificial life simulations (populations of autonomous interacting agents evolving in a virtual spatial world)
TNG/SimBioSys
(McFadzean/Tesfatsion 1997)
Each Tradebot has...

- **Internalized social norms** (market protocols) taken as given

- **Internally stored state data** that can change through experiences

- An **internal trade method (personality)** that the tradebot evolves over time in an attempt to increase its profits
TNG Flow Diagram

- **INITIALIZATION**

- **LOOP** Through TMax Trade Cycles
  - *Trade Cycle:*
    - Search for Trade Partners;
    - Interactions with Trade Partners;
    - Update Expectations about Trade Partners.

- **EVOLUTION STEP (Update Trade Methods)**

- **LOOP** Through Tmax Trade Cycles . . .
TNG Settings Screen

- Genetic Algorithm:
  - Generations: 50
  - Trade Cycles: 150
  - Seed: 19
  - Mutation Rate: 0.005

- Payoffs:
  - Initial Expected: 1.4
  - Refusal: -0.5
  - Inactive: 0
  - Experience Gain: 0
  - Both Co-op: 1.4
  - Both Defect: -0.6
  - Temptation: 3.4
  - Sucker: -1.6

- FSM:
  - States: 16
  - Memory: 1

- Trade Network Game:
  - Buyer: 12
  - Seller: 12
  - Dealer: 0
  - % Elite: 0.67
  - Quotas:
    - Buyer: 1
    - Seller: 1
TNG Chart Screen

The TNG Chart Screen displays the average scores over generations for different scenarios.

- **50/50 gens**: The graph shows the average scores for a scenario with 50% buyers and 50% sellers.
- **150/150 cycles**: The graph displays the average scores for a scenario with 150 cycles.

The chart includes options for viewing different statistics and settings for the simulation.

- **Run**, **Pause**, and **End** buttons for simulation control.
- **Animation** options to visualize the changes over generations.
- **Settings**, **Results**, **Chart**, **Animation**, and **Physics** tabs for different views.
- **Average Scores** graph for various scenarios:
  - **Buyer**, **Seller**, and **Dealer** scores are represented.
  - The graph indicates trends and variations in average scores across generations.
- **Generation** label along the x-axis with data points for each generation.
- **All Traders**, **Buyer**, **Seller**, **Dealer**, **Min**, **Avg**, and **Max** buttons for data filtering and analysis.

This screen is part of a simulation tool used to analyze market behavior and decision-making strategies in trading scenarios.
TNG Network Animation Screen
TNG Physics Screen

File Edit View Help

50/50 gens

Run Pause End

150/150 cycles

Animation: Stop

Settings Results Chart Animation Physics

Physics

Springs

Length: Strength:

Latched 150 200

Recurrent 250 100

Network Settings

Frequency Threshold:

Latched 6

Transient 6

Repulsion

Boundary 100

Trader 200

Friction 0.25

Reset
Related Online Resources

- ACE/CAS General Software and Toolkits
  https://www2.econ.iastate.edu/tesfatsi/acecode.htm
- ACE/CAS Computational Laboratories
  https://www2.econ.iastate.edu/tesfatsi/acedemos.htm
- Research Area: Development and Use of Computational Laboratories
  https://www2.econ.iastate.edu/tesfatsi/acoplabl.htm
- TNG Lab Home Page
  https://www2.econ.iastate.edu/tesfatsi/tnghome.htm