# CS 617 Object Oriented Systems Lecture 3 Object Abstractions, Encapsulation, Abstract Data Types 3:30-5:00pm Thu, Jan 10

**Rushikesh K Joshi**

Department of Computer Science and Engineering
Indian Institute of Technology Bombay

# Outline

# Outline

1. The Object Abstraction
2. Encapsulation
3. Abstract Data Types
4. Readings

# Objects

Every object has its own:

- Id
- State
- Behavior

# Id

- Objects are identified and distinguished from one another through their identities.
- At a given point of time during execution, two objects may have the same state and the same behavior, but they are distinguishable through their identities.
- Can objects with nil state and nil behavior exist?

# State

- Each object has its own set of local variables
- The values of these variables represents the current state of the object
- Can objects with nil state but non-nil behavior exist?

# Behavior

- How does an object undergo state changes?
- Member functions define the behavior
- Objects with nil behavior (no member functions) but non-nil state?

# Outline

# What is Encapsulation?

- How do you guarantee the integrity of an abstraction?
- Imagine an electronic equipment shipped with open access to internal circuitry
- Encapsulation is a process of concealing the implementation and making sure that access to the object occur only through the interface that represents the abstraction.
- Is encapsulation a new contribution from object orientation? Or did we know it before?

# Examples of Encapsulation

- A Process's internal data (pid,page tables etc) may not be manipulatable through the process system calls
- The body of a procedure may not be manipulatable through the prototype
- Local variables within a function are not accessible outside the function scope
- Hidden variables inside a file cannot be linked to
- Private variables and member functions in classes, of course!

# Breakage of Encapsulation

- If one gets direct access to internal state, encapsulation is broken.
- Abstraction may then no longer work.
- Pointers in C++ can cause breakage of encapsulation
- Pure object oriented languages do not permit violation of abstraction i.e. they do not permit breakage of encapsulation: **Support and Enforce Abstraction!**

# Levels of Encapsulation in Object Oriented Programs

- Internal visibility
- External visibility
- Subclass visibility
- Exclusive (Friend) visibility
- Package or module visibility

# Outline

# An ADT Example: Unbounded Stack

Let E be the element type and T be Stack type.
T holds elements of type E.

The below operations are defined for this type.

T new (void)
T push (E,T)
E top(T)
T removetop(T)
Boolean empty (T)

# Properties of the operations

- empty(new())
  new creates a nil stack
- top(push(e,t)) = e
  pushed element goes on top, top gives the recently pushed element
- removetop(push(e,t)) = t
  removetop retains the old stack prior to last push
- not empty(push(e,t))
  when a push operation is performed, the stack becomes non empty

# Partial Functions

- Some functions are not defined on all members of the input set
- Which of the those defined above are partial functions?

# Partial Functions in our Example

- *top* cannot return a value of type E for all values of input type T.
- Which one is that value?
- Similarly *removetop* does not work on all values of input type T.
- Which one is that value?
- How to handle the partially defined functions in ADT specification?

# Preconditions of Partial Functions

- T removetop (T) requires not empty (T)
- E pop (T) requires not empty (T)

# Summary of ADT Specification

- Types (used in the ADT)
- Functions (operations defined on these types)
- Axioms (properties over the functions defined)
- Preconditions

# Observations

- Nowhere we used the notion of state
- Behavior was defined in terms of a set of pure functions and their properties
- It's not easy to generate an ADT specifications
- Convert ADT specifications into classes

# Outline

# Readings

- Allan Snyder, Encapsulation and Inheritance in Object-Oriented Programming Languages, OOPSLA 1986, pages 38-45.
- Chapter 6 from Bertrand Meyer's book 'Object Oriented Software Construction'.