155

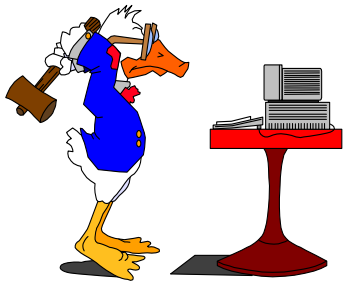
Chapter 5

Object-Oriented Programming

156

Traditional vs. Object Oriented Programming Paradigm

- **Traditional Paradigm**
  - Algorithm identification
  - Functional decomposition
    - Control Flow construction
    - Data Structure required for functions

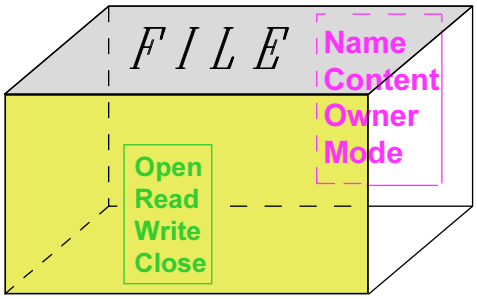


- **Object Oriented Paradigm**
  - Object identification
    - » ADTs that represent the problem domain.
  - Object Characteristic definitions
    - » Internal attributes of object.
  - Object Method (operation) definitions
    - » Abstract operations (i.e., interface) for the ADTs.
  - Solution is a sequence of calls to methods.

157

Example: The *File* Object

- A set of Characteristics
- A set of Methods (operations).



158

Why OOP ?

- The objects are the most stable factor in the problem
  - Algorithms may be improved
  - Implementations may be modified
  - Interface may undergo major changes
- If the data structure for the objects is defined according to *algorithm / implementation / interface* it is very likely to backfire when one of them is changed.
- Objects are the appropriate level at which decisions about encapsulation (information hiding) are made.

159

Example: A Complex Number

Characteristics:

A real part

An imaginary part.

Methods (operations):

Get / Set real / imaginary part

Get argument / absolute-value

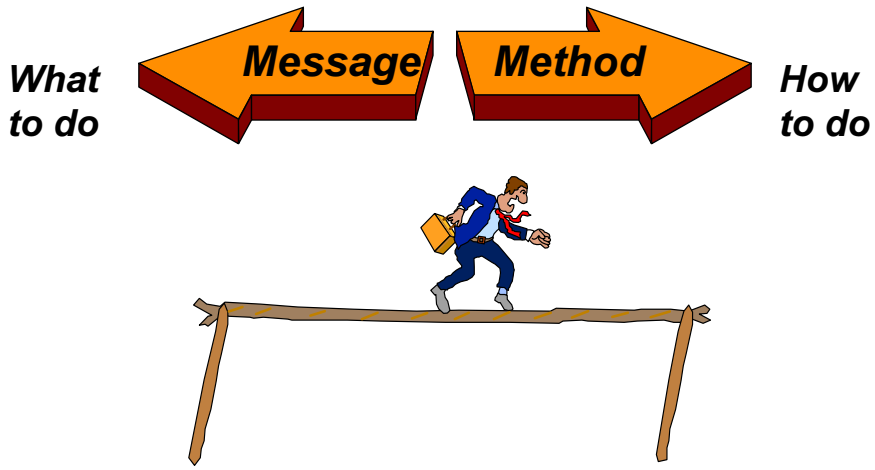
Compare to another complex / real number

Add to another number (real / complex)

Print yourself

160

Messages vs. Methods



161

Examples of Messages

A Set Object can receive messages such as:

- **Add** a given element
- **Remove** a given element
- **Is** a given element **a** member ?
- What is your **size** ?
- **Unite** yourself with another set object

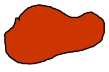
A Vehicle Object can receive messages such as:

- **Move** forward at a given speed (a certain distance)
- **Accelerate**
- **Stop**

162

Messages vs. Methods

In Object Oriented Programming,
the message "rotate n degrees clockwise"
would be implemented differently by different objects:



Shape: Do nothing if n = 360k, otherwise: rotate . . .

Circle: Do nothing.

Square: Do nothing if n = 90k, otherwise: rotate . . .

163

OOP Will

- Encourage use of ADTs
  - Make code predictable to use
  - Ease improving program implementation
  - Ease understanding of program structure
  - Help in documentation
- Make design decisions easier
- Encourage code reuse (within same/other project(s))

164

OOP Will NOT

- Design your program for you
- Prevent you from designing a bad program
- Provide algorithms
- Perform data management