Nine Software Concepts

- Encapsulation
- Information/Implementation Hiding
- State Retention
- Object Identity
- Messages
- Classes
- Inheritance
- Polymorphism
- Genericity
Hominoid Prototype

Hominoid Class

New: Hominoid
  // creates and returns a new instance of Hominoid
turnLeft
  // turns the hominoid counterclockwise by 90º
turnRight
  // turns the hominoid clockwise by 90º
advance (noOfSquares: Integer, out advanceOK: Boolean);
  // moves the hominoid a certain number of squares along
  // the direction that it’s facing and returns whether successful
location : Square
  // returns the current square that the hominoid is on
facingWall : Boolean
  // returns whether the hominoid is facing a wall
display
  // shows the hominoid on the screen
Grid Class

New: Grid
  // creates and returns a new instance of Grid with a random pattern
start: Square
  // returns the square that is the designated start of the path
finish: Square
  // returns the square that is the designated finish
insertHominoid(hom:Hominoid, location: Square,
  out insertOK: Boolean)
  // places the hominoid in the specified grid at the specified location
display
  // shows the grid as a pattern on the screen

The Program

var grid: Grid := Grid.New;
var hom1: Hominoid := Hominoid.New;
var insertOK: Boolean;
var advanceOK: Boolean;
var startSquare : Square
const oneSquare = 1;
startSquare := grid.Start;
grid.insertHominoid(hom1, startSquare, out insertOK);
if not insertOK
    then abort everything;
endif;
The Program (cont.)

repeat 4 times max or until not hom1.facingWall
    hom1.turnLeft;
endrepeat;
grid.display;
hom1.display;
repeat until hom1.location = grid.finish;
    if hom1.facingWall
        then hom1.turnLeft;
            if hom1.facingWall
                then hom1.turnRight; hom1.turnRight;
            endif;
    endif;
    hom1.advance(oneSquare, out advanceOK);
    hom1.display;
endrepeat;
Object-Oriented Encapsulation

- Group the data and algorithms that operate on that data into a single entity.
- The data represents the current state.
- The algorithms depend upon the state.
- The algorithms modify the state.

Information Hiding

- The use of encapsulation to restrict from external visibility certain information or implementation decisions that are internal to the encapsulation structure.
- This is a powerful technique for taming software complexity.
  - Localizes design decisions
    - Decouples content from representation.
State Retention

- In modern (block structured) languages a subroutine (function) does not have a history. (The exception being local static variables in C.)
- Communication of state information between subroutines (functions) requires the user to maintain and pass-around the data or the use of global static variables.

State Retention (2)

- In an object-oriented language, data and the functions that operate on that data are grouped together.
- The functions that need to view the data, and only the functions that need to view the data, can view the data.
Object Identity

• The property by which each object (regardless of class or current state) can be identified as a distinct software entity.
• In some languages there is a unique handle
• Some languages are not pure in that dead objects may be recycled. (Object identity is the address at which the object is stored.)

Messages

• The vehicle by which a sender object (obj1) conveys to a target object (obj2) a demand for obj2 to apply one of its methods.
• In order for obj1 to send a sensible message to obj2:
  – Must have a handle for obj1
  – Must know what method (operation) to be performed.
  – Must know the supplementary data (if any).
Messages vs. Procedure Calls

• Message
  – hom1.turnRight;
• Procedure Call
  • call turnRight(hom1);
• The two are, in fact, equivalent.

Message Arguments

• In a pure OO language, all arguments are handles to objects.
• In C++ they may be either pointers/references to objects or pointers/references to built-in types or copies of objects or built-in types.
• In Java they may be references to objects or copies of build-in types.
Types of Messages

• Informative
  – Provides data that permits an object to update itself.

• Interrogative
  – Request an object to reveal some information about itself.

• Imperative
  – Request an object to take some action.

Classes

• A class is a stencil from which objects are created (instantiated). Each object has the same structure and behavior as the class from which it is instantiated.
• Classes are what you design.
• Objects are created from classes at run-time.
Inheritance

- The facility by which a class \( D \) has implicitly defined upon it each of the attributes and operations of class \( C \), as if those attributes and operations had been defined upon \( D \) itself.
- \( C \) is termed a superclass of \( D \).
- \( D \) is termed a subclass of \( C \).

Glider derived from Aircraft

\[
\begin{array}{|c|}
\hline
\text{Aircraft} \\
\hline
\text{course:Angle} \\
\hline
\text{turn} \\
\hline
\end{array}
\]

\[
\begin{array}{|c|}
\hline
\text{Glider} \\
\hline
\text{isTowLineAttached:Boolean} \\
\hline
\text{releaseTowline} \\
\hline
\end{array}
\]

\[
\begin{array}{c}
\text{Aircraft} \\
\end{array}
\rightarrow
\begin{array}{c}
\text{Glider} \\
\end{array}
\]
Multiple Inheritance

- Aircraft
- PassengerVehicle
- PassangerAircraft

Polymorphism

- The facility by which a single operation or attribute may be defined upon more than one class at a time and may take on different implementations in each of those classes.
- The property by which an attribute (or variable) may point to (hold the handle of) objects of different classes at different times.
Overriding vs. Overloading

- Overriding is the redefinition of a method defined on class C in one of its subclasses.
- Overloading is the ability to use the same name for different operations, where the distinction is made by the types of the arguments.
Static vs. Dynamic Binding

- In static binding overriding or overloading is resolved by the compiler and does not change at run-time.
- In dynamic binding overriding is resolved at run-time based upon the actual class that a pointer/reference refers to.

Genericity

- The construction of a class so that one or more of the classes that it uses is supplied when the object is instantiated.
Binary Search Tree of Integers

class intBST{
    public:
        void insert(int K);
    private:
        struct node{int data;node *left, *right;};
        node* root;
};

Binary Search Tree of ProductID

class ProductIDBST{
    public:
        void insert(ProductID K);
    private:
        struct node{ProductID data;node *left, *right;};
        node* root;
};
Generic BST

template <typename T> class BST{
    public:
        void insert(T K);
    private:
        struct node{T data; node *left, *right;};
        node* root;
};

Contributors to OO

Constantine
Dahl and Nygaard
Kay, Goldberg, et al
Dijkstra
Liskov
Parnes
Ichbiah et al
Stroustrup
Meyer
Booch, Jacobson, and
Rumbaugh

Coupling & Cohesion
Simula
Smalltalk
Structured Programming
Abstract Data Types
Information Hiding
Ada
C++
Eiffel
UML
What is OO Good for?

- Nothing
- Everything

OO is Good for Nothing

- Since OO has built upon previous analysis and design techniques, many of its features and concepts are not new.
- There is a lot of hype.
  - X is good.
  - OO is good.
  - Therefore X is OO.
OO is good for Everything

• First and only miracle solution
• Not only does it do windows, but it also slices and dices vegetables, wax floors, and top-off deserts -- all under the latest multithread distributed Web-enabled architecture.

Fad of the Year

• Adherence espouse it hysterically as the solution to all problems.
• Skeptics are dragged aboard its bandwagon by the forces of fanaticism.
• Approach is used and abused with mediocre results.
• Abandoned in favor of the next fad.
What is OO really good for

• Analyzing user’s requirements
• Designing software
• Constructing software
  – Reusability
  – Reliability
  – Robustness
  – Extensibility
  – Distributability
• Maintaining Software
• Using Software
• Managing Software Projects