Basics of Object-oriented Programming in Avenue

- Procedural -vs- Object-Oriented
- Elements of Object Oriented Programming
Procedural -vs- Object-Oriented

• Procedures and Variables
  – Procedural programs are simply a linear series of procedures that execute operations on variables.

• Objects with Attributes and Methods
  – Object oriented programs are collections of classes that define objects. These objects store attributes and know how to perform specific tasks.
Procedural Example

• Basic attributes of your rectangle
  
  minX = 10
  maxX = 15
  minY = 0
  maxY = 10

• Attributes stored separately, but brought together in procedures

• You must write a procedure that takes these variables as arguments any time that you want to do something with your rectangle
Object-Oriented Example

• Create a class that knows all about rectangles
  
  Rectangle.Make(10, 15, 0, 10)

• Now you have a rectangle object

• All procedures are stored in the rectangle class, so anytime you want to do something with it, you make a request to the object
Variables

• Procedural Languages
  – Variables are usually references to a date element of a specific data type (although collections of elements, such as lists and arrays are usually supported as well).

• Object-Oriented Languages
  – Variables are references to a specific object
Avenue: A Scripting Language

• You access object-oriented structures in Avenue
  – You can create and manipulate objects in Avenue

• You can only write procedural scripts
  – You cannot create or modify classes in Avenue
Variables in Avenue

• Many languages require you to declare your variables to be of a certain type at the beginning of your code. Avenue does not. You can create new variables at any point in your code. Additionally, you can change the type of data referenced by that variable at any given point.
Concepts in Object Oriented Programming

• Encapsulation
• Classes
  – Attributes
  – Operations
• Objects
  – Requests
• Polymorphism
• Object Hierarchies and Inheritance
Encapsulation

• In object-oriented languages, data and procedures are kept together. A class defines both data and procedures for processing those data.
Classes

• The concept of a “class” is the most basic concept of object oriented programming.
• A class groups objects based on common characteristics.
• All objects of a class share the same attributes and operations.
Attributes

• The variables that a class stores internally
  – Attributes are the salient pieces of information that define it as unique
  – Attributes can be basic data types, or other objects

• Example:
  – A rectangle objects attributes can be a point and two double-precision floating point variables. The point could be the lower left corner of the rectangle (or the insert), and the two doubles could be the width and height.
Operations (methods, services)

• Each class has a number of methods that tell that class how to do different things.

• Example:
  – A rectangle may know how to calculate its area and its perimeter, and also may know how to change its insertion point, and resize itself.
Objects

• An object is one instance of a class with a specific set of attributes
• Example:
  – I may have two rectangles. One is seven inches wide and three inches tall, and has an insertion point at (1,1). The other is four inches wide and two inches tall, and has an insertion point at (4,2). Both of these rectangles are objects of the class type rectangle.
Requests

• Requests call methods to execute a procedure.
• Example:
  – You want to get the area of a rectangle from our rectangle class. Let's say we have a GetArea method defined. You would simple take your rectangle object and send it the GetArea request as follows:
    `myRectangle.GetArea`
  – This will return the area of the rectangle.
You can make a request to an object or a class.

– Example:

• The previous example sends a request to an object. However, what would you do if you haven’t created a rectangle object yet? You need to send a request directly to the rectangle class.

```python
myRectangle = Rect.Make(1,1,4,2)
```
Arguments (parameters)

- The arguments provide specific information that is needed to execute a request. In the previous example:
  
  ```
  myRectangle = Rect.Make(1,1,4,2)
  ```
  - The arguments are the numbers 1,1,4,2
  - They provide the following information for the request:
    - X insertion point, Y insertion point, width, height
Return Objects from a request

• Many requests return a specific type of object. In the example of making a new rectangle, the return object is (obviously) a rectangle:

  \[
  \text{myRectangle} = \text{Rect.Make}(1, 1, 4, 2)
  \]

  – The variable myRectangle references the rectangle object returned by the Make request to the Rect class
Making Requests

• There are three types of notation for making a request in Avenue:
  – Postfix
    myRectangle.Move(0,1)
  – Infix
    myCount = myIndex + 1
  – Prefix
    Not true
Request Conventions in Avenue

- Avenue has naming conventions for its requests. They are based on *Actions* and *Properties*

  myRect.GetBottom

  - The action is Get and the property is Bottom. This request returns a number that represents the position of the bottom of the rectangle.
Request Actions

• While there are a large number of properties that requests access, Avenue has a few key request actions that tell you something about what the request does. These actions are listed on the following slides.
Add

• Aggregate, or add, one object to another.

    myView.AddTheme(myTheme)
As

• Convert an object from its current class to another class.
  myTheme.AsString
Can

- Return the ability of an object or class to perform a task.
  
  `myFeatureTable.CanAddFields`
Find

• Finds an object in an aggregation using the object’s name.
  
  myProject.FindDoc("Scatterplot")
Has

- Checks the state of an associated object, or an attribute.

  myProject.HasDoc(myDoc)
Is

• Checks the state of an object.
  myTheme.IsActive
Make

• Applied to a class to create a new object.

Theme.Make
Set

- Set allows you to change an attribute of an object.
  
  ```
  myTheme.SetActive(true)
  ```
Get

• Retrieves the value of an object’s attribute, or a reference to an associated object.

• Gets Attribute:
  myRect.GetBottom

• Gets Associated Object:
  myDoc.GetWin
Return

• Works similar to get with one MAJOR difference:
  – While *Get* retrieves a reference to an attribute of an object
  – *Return* creates a new object that is a copy of the attribute
Get -vs- Return: An Example

• Get
  myLeft1 = myRect.GetLeft
  myLeft1 = myLeft1 + 3
  myLeft2 = myRect.GetLeft
  – In this case, myLeft1 is equal to myLeft2

• Return
  myLeft1 = myRect.ReturnLeft
  myLeft1 = myLeft1 + 3
  myLeft2 = myRect.ReturnLeft
  – In this case, myLeft1 is not equal to myLeft2
Polymorphism

• Polymorphism allows different objects or classes to respond to the same request in different ways.
Polymorphism

• Between Classes
  – In procedural languages, if you wanted to make a rectangle and make a circle, you would need to have different names for those procedures. In an object-oriented language, you can simply use the “make” request on different types of classes. The language will automatically process the request correctly.
Polymorphism

• Within Classes
  – Furthermore, (in some OO languages) you can make the same request to the same class or object using different parameters, and the language will know how to parse it.
Structures Between Classes

• Inheritance
• Association
• Object Modeling Techniques (OMT)
• Types of Associations
  – Generalization
  – Multiplicity
  – Aggregation
Inheritance

• Attributes and operations can be shared among objects in a hierarchical relationship.

• Super ordinate classes
  – The classes that a class inherits attributes and methods from

• Subordinate classes
  – The classes that inherit a class’ attributes and methods
Inheritance – Example

• Doc – The document class has an attribute specifying the document window object.
• View – is a subclass of Doc, and thus you can access the document window object that the View inherited from the Doc.
Association

- Associations describe relationships between classes.
- Associations can be unidirectional or bi-directional.
- Associations can be named with verbs to indicate the type of relationship between classes.
Object Modeling Techniques (OMT)

- The graphical representation of the relationships between classes.
- OMT symbols will be demonstrated for each of the following associations
Generalization

- Generalization associations are the hierarchical relationships generated through inheritance.
Generalization OMT

• The OMT graphic for a generalization relationship between Doc and its subclasses is depicted below.
Multiplicity

- Multiplicity indicates the number of class instances (objects) that can be involved in an association.
  - One to one
  - Optional one to one
  - One to many
Multiplicity OMT

- The OMT graphic for a multiplicity relationships are depicted below.
  - One to one
    ![Diagram of one to one relationship]
  - Optional one to one
    ![Diagram of optional one to one relationship]
  - One to many
    ![Diagram of one to many relationship]
Aggregation

• An aggregation indicates that one object of one class can houses many objects of a second class. For instance, a View object can contain any number of themes.
  • This is not the same as a one to many multiplicity relationship, as one DocGUI can be related to many Doc objects, but doesn’t house a collection of them.
Aggregation OMT

- The OMT graphic for the aggregation relationship is depicted below.
Abstract and Concrete Classes

• Abstract classes are never instantiated in objects, but allow subclasses to inherit their attributes and methods.
• Concrete classes can be instantiated in objects.
Conclusion

• Classes define objects that are self-contained units of attributes and methods.
• Classes are related to one another in a number of ways defined by:
  – The number of objects that can be involved in a relationship (multiplicity)
  – The hierarchical structure of the relationship (generalization – inheritance)
  – Whether or not one class can contain a collection of objects from another class (aggregation)