Easing into OOPs

Richard Taylor
Director, Technical Communications
TopSpeed Corporation
Sept. 24, 1997

Object Oriented Programming Using *Clarion 4*

Procedural Code

- PROCEDUREs have Data and Code Sections
- Local Data exists only in the PROCEDURE
- Local Data is allocated stack memory at the PROCEDURE's CODE statement
- Local Data memory is de-allocated at the PROCEDURE's RETURN statement

What if you want multiple Code Sections to operate on your variables?

- Make the variables Global
- Make the variables Module data

[redacted] corporation

Make the Variables Global

- Plus points:
  - Available to multiple PROCEDUREs
- Out points:
  - No limit to their lifetime, memory is allocated for them for the duration of the program
  - Too much Global data is a "bad thing"

[redacted] corporation

What if you need multiple SETs of data?

- You *could* place them all in a QUEUE
  - Too complex too quickly
- There is a BETTER WAY

[redacted] corporation

The CLASS Structure

```
MyClass   CLASS
Property     LONG
QueProperty &SomeQueue
Method1      PROCEDURE
Method2      PROCEDURE(LONG),LONG
          END
```

5-6, a better example
Circle
X, Y, R Long
Area
Circum

Buzzword Alert!

- Property, Method, Object
- Encapsulation
  - "To enclose (as) in a capsule" -- OED
- INSTANTIATE
  - "To represent by an instance" -- OED
- INSTANTIATION
  - "Representation by an instance" -- OED

Create

is it wrong to INST w/o Rep

Object Instantiation

- Declare a CLASS without the TYPE attribute
- Declare an object using the name of a previously declared CLASS as its data type
- Declare a reference to a previously declared CLASS

```
     PROGRAM          !Global Data and Code
MyClass   CLASS       !Declare an object and
Property    LONG      ! a type of object
Method      PROCEDURE
          END
ClassA    MyClass     !Declare MyClass object
ClassB    &MyClass    !Declare MyClass reference
     CODE             !MyClass and ClassA are
                      ! automagically instantiated
     ClassB &= NEW(MyClass)
                      !Instantiate object
     ! execute some code
     DISPOSE(ClassB)!Destroy object (required)
     RETURN           !MyClass and ClassA
                      ! automagically destroyed
```

[redacted]

```
     MEMBER('MyApp')!Module Data

MyClass   CLASS       !Declare an object and
Property    LONG      ! a type of object
Method      PROCEDURE
          END
ClassA    MyClass     !Declare MyClass object
ClassB    &MyClass    !Declare MyClass reference

SomeProc  PROCEDURE
     !MyClass and ClassA are instantiated and
     !  desttroyed at the same time as the
     !  Global objects
     !ClassB must be explicitly instantiated
     !  and destroyed in some PROCEDURE in
     !  the module
```

[redacted]

```
SomeProc  PROCEDURE !Local Data and Code
MyClass   CLASS       !Declare an object and
Property    LONG      ! a type of object
Method      PROCEDURE
          END
ClassA    MyClass     !Declare MyClass object
ClassB    &MyClass    !Declare MyClass reference
     CODE             !MyClass and ClassA are
                      ! automagically instantiated
     ClassB &= NEW(MyClass)
                      !Instantiate object
     ! execute some code
     DISPOSE(ClassB)!Destroy object (required)
     RETURN           !MyClass and ClassA
                      ! automagically destroyed
```

[redacted]

```
      PROGRAM
Employee  CLASS,TYPE      !Declare object TYPE
Pay         DECIMAL(7,2)
Hours       DECIMAL(3,1)
CalcPay     PROCEDURE
Work        PROCEDURE(*DECIMAL),DECIMAL
          END
Fred      Employee
Barney    Employee
     CODE
     Fred.Pay = Fred.Work(Fred.Hours)

Employee.CalcPay  PROCEDURE  !Method Definition
     CODE
     SELF.Pay = SELF.Work(SELF.Hours)
```

[redacted]

Buzzword Alert!

- Property, Method, Object
- Encapsulation, Instantiation
- CONSTRUCTORS
- DESTRUCTORS

[redacted]

Automatic Constructors and Destructors

```
MyClass   CLASS
Property    LONG
Method      PROCEDURE
Construct   PROCEDURE  !Automatic Constructor
Destruct    PROCEDURE  !Automatic Destructor
          END
```

→ use to init certain values

[redacted]

Private Property!

```
     PROGRAM
MyClass   CLASS
MyProperty  LONG,PRIVATE     !Private Property
Method      PROCEDURE
MyMethod    PROCEDURE,PRIVATE !Private Method
          END
     CODE
     MyClass.MyMethod          !Invalid here
     MyClass.Method            !Valid here
     MyClass.MyProperty = 10   !Invalid here
MyClass.Method
     CODE
     SELF.MyMethod             !Valid here
     SELF.MyProperty = 10      !Valid here
```

Private outside the class

Buzzword Alert!

- Property, Method, Object
- Encapsulation, Instantiation
- Constructors, Destructors
- INHERITANCE
  - Derive: "to transmit, impart, communicate, pass on, hand on." -- OED

```
     PROGRAM
MyClass   CLASS           !Declare Base Class
Property    LONG
MyProperty  LONG,PRIVATE !Private = no inherit
Method      PROCEDURE
          END
ClassA    CLASS(MyClass) !Declare Derived Class
Aproperty   LONG          ! which inherits both
Amethod     PROCEDURE     ! MyClass.Property and
          END             ! MyClass.Method
     CODE
     ClassA.Method        !Valid method call
     ClassA.MyProperty = 10
                          !Invalid, not inherited
     MyClass.Amethod      !Invalid, inheritance
                          ! only is one way
```

Base Class has not inherited
A Derived Class always has a Parent
Does not inherit Private??

```
     PROGRAM
ApplePie  CLASS,TYPE      !Declare Base Class
Apples      STRING(20)
Crust       STRING(20)
Bake        PROCEDURE
          END
Dutch     CLASS(ApplePie)!Declare Derived Class
CrumbleTop  STRING(20)
          END
American  CLASS(ApplePie)!Declare Derived Class
TopCrust    STRING(20)
          END
Grandmas  CLASS(ApplePie)!Declare Derived Class
CaramelTop  STRING(20)
Bake        PROCEDURE    !Overridden method
          END
```

Same Prototype Name

Buzzword Alert!

- Property, Method, Object
- Encapsulation, Instantiation
- Constructors, Destructors
- Inheritance
- COMPOSITION = No Disambiguation!

Required with Multiple Inherit

Composition

```
     PROGRAM
ApplePie  CLASS,TYPE      !Declare Base Class
Apples      STRING(20)
Crust       STRING(20)
Bake        PROCEDURE
          END
IceCream  CLASS,TYPE      !Declare Base Class
Flavor      STRING(20)
Scoop       PROCEDURE
          END
AlaMode   CLASS(ApplePie) !Composition: Derive
OnTheSide   &IceCream     ! with reference to
Serve       PROCEDURE     ! an object
          END
```

```
     PROGRAM
MyClass   CLASS           !Declare Base Class
Property    LONG
MyProperty  LONG,PROTECTED !Semi-Private
Method      PROCEDURE
          END
ClassA    CLASS(MyClass) !Declare Derived Class
Aproperty   LONG
Amethod     PROCEDURE
          END
     CODE
     ClassA.MyProperty = 10
                         !Invalid out of method
MyClass.Amethod PROCEDURE
     CODE
     SELF.MyProperty = 1 !Valid within method
```

Protected

Inherited Constructor Execution Order

- Base Class Constructor executes automatically when the object is instantiated
- The Constructor for any CLASS Derived from the Base Class executes next
- The Constructor for any CLASS Derived from the Derived Class executes next
- ...

Parent/Base. Construct
then
Derived. Construct
work . . . . . .
Derived. Destruct
Base. Destruct

Inherited Destructor Execution Order

- The Destructor for the most Derived Class executes first
- The Destructor for the next most Derived Class executes next
- ...
- The Base Class Destructor executes last

```
     PROGRAM
MyClass   CLASS,TYPE      !Declare Base Class
Property    LONG
Method      PROCEDURE
Construct   PROCEDURE
          END
ClassA    CLASS(MyClass) !Declare Derived Class
Aproperty   LONG
Construct   PROCEDURE,REPLACE
          END
     CODE              !ClassA Instantiation here

ClassA.Construct PROCEDURE
     CODE
     SELF.Aproperty = 1  !Initialize then call
     PARENT.Construct    ! parent constructor
```

Don't Run Parent. Construct

PARENT Keyword

Buzzword Alert!

- Property, Method, Object
- Encapsulation, Instantiation
- Constructors, Destructors
- Inheritance, Composition
- POLYMORPHISM
  - "The condition or character of being polymorphous; the occurrence of something in several different forms." -- OED
- VIRTUAL METHODS

Function Overloading

What's a Virtual Method?

- A method whose prototype is present in the Parent CLASS
- A method whose exact same prototype is present in the Derived CLASS
- A method whose prototype in both CLASS structures has the VIRTUAL attribute

What good are Virtual Methods?

- Inheritance allows Derived Class methods to "call down" to Parent Class methods
- Virtual Methods are the opposite: they allow Parent Class methods to "call up" to execute Derived Class methods
- Again: Virtual Methods allow the code in Parent Class methods to execute Derived Class methods

[redacted]

```
ApplePie  CLASS,TYPE
PreparePie  PROCEDURE
CreateCrust PROCEDURE,VIRTUAL !Virtual Methods
MakeFilling PROCEDURE,VIRTUAL
          END
Dutch     CLASS(ApplePie)
CreateCrust PROCEDURE,VIRTUAL !Virtual Methods
MakeFilling PROCEDURE,VIRTUAL
          END
     CODE
     Dutch.PreparePie    !Will call the Dutch
                         ! object's Virtuals
ApplePie.PreparePie PROCEDURE
     CODE
     SELF.CreateCrust
     SELF.MakeFilling
```

[redacted]

Buzzword Alert!

- Property, Method, Object
- Encapsulation, Instantiation
- Constructors, Destructors
- Inheritance, Composition
- Polymorphism / Virtual Methods
- LATE BINDING = Virtual Method Table

[redacted]

Local Derived Methods

```
SomeProc  PROCEDURE
PieType   STRING(20)          !Local variable
Dutch     CLASS(ApplePie)     !Locally derived
CreateCrust PROCEDURE,VIRTUAL !Virtual Method
          END
     CODE
     !executable code
SomeRoutine ROUTINE
     !executable code

Dutch.CreateCrust PROCEDURE   !Locally derived
     CODE
     PieType = 'Dutch Apple'  !Legal, in scope
     DO SomeRoutine           !Legal, in scope
```

[redacted]

New in Beta 2

Give Variables Scope into the Derived Methods

Also Routines

Clarion 4
OOPs in a nutshell

- Encapsulation, Inheritance, Polymorphism
- Properties, Methods, Objects
- Instantiation, Base Classes, Derived Classes
- SELF, PARENT, Constructors, Destructors
- Virtual Methods, Late Binding
- And NO Disambiguation!

[redacted]