Object Oriented Programming and Exception Handling

Martin Phillips
Ladybridge Systems Ltd

International Spectrum Conference, 2014

OO Programming and Exception Handling

Why do we need this?

Most developers that come to the multivalue environment from other programming languages expect to find features similar to those that they already know.

Multivalue Basic is an excellent language for rapid application development with some very powerful capabilities but may appear alien at first.

Object Oriented Programming

What’s It All About?

OO programming does not replace “conventional” methods.

A new addition to the developer’s toolbox.

An integral part of the QMBasic language.

Object Oriented Programming

What Is An Object?

Subroutine:

Program operations that work on supplied data.

Object:

Data that has associated program operations.

Object Oriented Programming

What Is An Object?

Defined by a CLASS module.

The CLASS module is a container for...

- **The persistent data definitions.**
- **The program operations that run against this data.**

Object Oriented Programming

What Is An Object?

An object is a run time instance of the class.

var = OBJECT(“myobj.cls”)

There may be many concurrent instances of the same class each with their own independent data.

Object Oriented Programming

The Objref Operator (->)

References an element of the object.

var->name

var->(*expr*)

Object Oriented Programming

Persistent Data

- Class modules may use common blocks but these are shared across programs, subroutines and object instances.
- Class modules also have persistent data that is separate for each instance and shared data that is visible to all instances of the same class.

Object Oriented Programming

Persistent Data

Private data...

- **Not visible outside the class module.**
- **Hides internal operation of the object.**

```
PRIVATE  A,  B,  C(2,3)
```

Object Oriented Programming

Persistent Data

Public data...

- **May be visible to programs using the object.**

```
PUBLIC  P,  Q,  R(2,3)

PUBLIC  X  READONLY
```

Object Oriented Programming

Shared Data

Public or private

Shared across all instances of the class.

```
SHARED  PUBLIC  P,  Q,  R(2,3)

SHARED  PUBLIC  X  READONLY
```

Object Oriented Programming

Persistent Data

Referenced from calling program:

```
result = var->item

var->item = 12

var->item(3) = 12
```

Object Oriented Programming

Public Subroutines and Functions

Program operations contained within the class module.

May access or update persistent data.

Public subroutines store values or perform tasks.

Public functions return a result value.

Object Oriented Programming

Public Subroutines and Functions

```
PUBLIC SUBROUTINE name
   ... Program operations ...
END
```

```
var->name
```

Object Oriented Programming

Public Subroutines and Functions

```
PUBLIC SUBROUTINE name(a,b)
   ... Program operations ...
END
```

```
var->name(x,y)

var->name(x) = y
```

Object Oriented Programming

Public Subroutines and Functions

```
PUBLIC FUNCTION name(a,b)
   ... Program operations ...
   RETURN value
END
```

```
p = var->name(q, r)
```

Object Oriented Programming

Public Subroutines and Functions

Variable length named argument lists...

```
PUBLIC FUNCTION name(a,b) VAR.ARGS
   ... Program operations ...
   RETURN value
END
```

Object Oriented Programming

Public Subroutines and Functions

Variable length unnamed argument lists...

```
PUBLIC FUNCTION name(a, ...)
   ... Program operations ...
   RETURN value
END
```

Object Oriented Programming

Public Subroutines and Functions

Access arguments by position...

```
ARG.COUNT()

ARG(n)

SET.ARG n, value
```

Object Oriented Programming

Dual Identity

A name may refer to a public data item when reading and program operations when writing...

...Or vice versa

Allows easy data validation or event triggers.

Object Oriented Programming

Inheritance

- One class may want to use the data and public routines of another.
- The inherited class remains a “black box” where the outer class cannot see how it works.

Object Oriented Programming

Inheritance

Static Inheritance...

```
CLASS name INHERITS other.class
```

Object Oriented Programming

Inheritance

Dynamic Inheritance...

```
obj = object("otherclass")
INHERIT obj
```

Object Oriented Programming

Inheritance

Dis-inheritance...

DISINHERIT obj

Object Oriented Programming

“Automatic” Handlers

CREATE.OBJECT

DESTROY.OBJECT

UNDEFINED (Subroutine / Function)

Object Oriented Programming

“Automatic” Handlers

CREATE.OBJECT

Run when the object is instantiated.

Arguments to OBJECT() are passed to this subroutine.

Object Oriented Programming

“Automatic” Handlers

DESTROY.OBJECT

Run when the last variable referencing the object is released.

Guaranteed execution, even at program abort.

Object Oriented Programming

“Automatic” Handlers

UNDEFINED

Run for references to undefined names.

Both FUNCTION and SUBROUTINE can exist.

Caller’s arguments passed, plus name.

Object Oriented Programming

Example Class Module

There is a standard class module in the BP file of the QMSYS account to walk through an alternate key index one record id at a time.

Object Oriented Programming

Step 1 – Data Definitions

```
CLASS INDEX.CLS
   PRIVATE FVAR, INDEX.NAME
   PRIVATE ITEMS, NUM.ITEMS, ITEM.INDEX
   PUBLIC KEY READONLY

   ...Subroutines & functions go here...
END
```

Object Oriented Programming

Step 2 – CREATE.OBJECT

```
PUBLIC SUBROUTINE CREATE.OBJECT(FILE,INDEX)
   FVAR = FILE                ;* Save file variable
   INDEX.NAME = INDEX         ;* and index name

   ITEMS = “”                 ;* Id cache empty
   NUM.ITEMS = 0              ;* No ids in cache
   ITEM.INDEX = 0             ;* No next id position

   SETLEFT INDEX.NAME FROM FVAR
END
```

Object Oriented Programming

Step 3 – Fetch Next Id

```
PUBLIC FUNCTION NEXT
   IF ITEM.INDEX >= NUM.ITEMS THEN
      SELECTRIGHT INDEX.NAME FROM
        FVAR SETTING KEY TO 10
      READLIST ITEMS FROM 10 ELSE NULL
      NUM.ITEMS = DCOUNT(ITEMS, @FM)
      ITEM.INDEX = 0
      IF NUM.ITEMS = 0 THE RETURN “”
   END
   ITEM.INDEX += 1
   RETURN ITEMS<ITEM.INDEX>
END
```

Object Oriented Programming

Step 4 – Position at Specified Id

```
PUBLIC SUBROUTINE SET(VALUE)
   KEY = VALUE
   SELECTINDEX INDEX.NAME, KEY FROM FVAR TO 10
   READLIST ITEMS FROM 10 ELSE NULL
   NUM.ITEMS = DCOUNT(ITEMS, @FM)
   ITEM.INDEX = 0
END
```

Object Oriented Programming

Using the Class

```
OBJ = OBJECT("!INDEX.CLS", FVAR, INDEX.NAME)

OBJ->SET(VALUE)

LOOP
   ID = OBJ->NEXT
UNTIL ID = ""
   DISPLAY  OBJ->KEY, ID
REPEAT
```

Exception Handling

What is an Exception?

An exception is a named event, often an error, that can be trapped by an application in a controlled manner.

Exception handling is based on the concept of a TRY/CATCH block in which the TRY clause contains program statements to be attempted and the CATCH clause traps specific exceptions.

An exception is “thrown” by the program in which it occurs.

Exception Handling

Example - No error handling

```
TOTAL += NEW.VALUE
```

If NEW.VALUE is not numeric, a run time error will occur, aborting the program

Exception Handling

Example - Explicit error handling

```
IF NUM(NEW.VALUE) THEN
   TOTAL += NEW.VALUE
ELSE
   ….Error action….
END
```

The developer must explicitly test for each error condition that they need to trap.

Exception Handling

Example - Exception Handling

```
TRY
   TOTAL += NEW.VALUE
CATCH SYS.PROGRAM.DATATYPE
   ….Error action….
END
```

This example still requires the developer to identify the error conditions that they need to trap

The SYS.PROGRAM.DATATYPE exception occurs at any data type error.

Exception Handling

Generic Exception Handling

```
TRY
   TOTAL += NEW.VALUE
CATCH SYS$ANY
   ….Error action….
END
```

Use of SYS$ANY traps any exception raised by the statement(s) in the TRY clause.

Exception Handling

Scope of Exception Handlers

```
TRY
   CALL MYSUB
CATCH SYS$ANY
   ….Error action….
END
```

The exception handler covers all actions in the TRY clause including exceptions thrown in other programs.

Exception Handling

Exception Names

Exception names can be long. The names are formed from a hierarchy of component names.

Any error that would normally cause an abort with a “non-numeric where numeric required” message can be trapped as exception SYS.PROGRAM.DATATYPE.NOT_NUMERIC

Each period separated element of this name forms an exception group.

Exception Handling

Exception Groups

SYS.PROGRAM.DATATYPE.NOT_NUMERIC

This can be caught as

- **SYS.PROGRAM.DATATYPE.NOT_NUMERIC**
- **SYS.PROGRAM.DATATYPE**
- **SYS.PROGRAM**
- **SYS**
- **SYS$ANY**

Exception Handling

Throwing an Exception

A program throws an exception with

```
    THROW "NAME"
```

or

```
    THROW "NAME", QUALIFIER
```

The qualifier may be any QM data item

All subroutines are discarded back as far as the exception handler

The DESTROY.OBJECT subroutine of an OO programming object will be executed.

Exception Handling

Exception Information

@EXCEPTION

The exception name

@EXCEPTION.ORIGIN

Program name and line number

@EXCEPTION.DATA

The qualifier to THROW

Exception Handling

Is there a Handler?

The CAUGHT() function tests whether there is a handler for a named exception

```
IF CAUGHT('NAME') THEN …
```

Exception Handling

The SYS$UNHANDLED Handler

If there is no other handler that catches the exception, the optional SYS$UNHANDLED handler is used.

Exception Handling

Exceptions and Aborts

An exception for which there is no handler results in an abort

An abort will look for a SYS.ABORT exception handler.

An EXECUTE with TRAPPING.ABORTS forms a barrier beyond which the search for an exception handler will not pass.

QUESTIONS?

Open QM

Ladybridge Systems

taking multivalue where it has never been before ...