



Inside MATLAB Objects in R2008a

By Dave Foti

The software industry has evolved considerably since the early 1990s, when MATLAB® object-oriented programming features were first developed. For example, design patterns using objects are now commonplace. Over the intervening years, MathWorks developers have learned a great deal about how programmers use objects and what capabilities they require.

A key goal in updating the object-oriented programming capabilities of MATLAB in R2008a was to apply some of these lessons while remaining true to three core principles of the MATLAB language: the centrality of arrays and array indexing, the importance of mathematical functions, and the use of multiple named input and output parameters.

For experienced object-oriented programmers, this article explains the rationale behind some of the MATLAB design decisions in R2008a, including why MATLAB object-oriented features differ in significant ways from other popular object-oriented languages. In particular, this article examines methods and parameters, inheritance, handles, properties, and object life-cycle management.

Methods and Parameters

In R2008a, MATLAB preserves the method-definition, calling, and dispatching semantics of earlier versions of MATLAB. This

means that objects are explicit parameters to the methods that act on them. Unlike many object-oriented languages, in MATLAB the array, or matrix, takes center stage. There is no special type for arrays, and all classes inherently support arrays. Moreover, because MATLAB methods frequently need to act on multiple objects, symmetry among multiple object parameters is more important in MATLAB code. For example, in MATLAB, `plus` is implemented with

```
function C = plus(A, B)
```

Inside the method, A, B, and C are all variables that can be given names appropriate to the context and used as variables might be used in mathematical equations. After all, methods are simply functions that act on objects. The only characteristic unique to methods is that they have access to the internals (the protected and private definitions) of the class. In MATLAB, there are no implicit parameters to methods.

Product Used

- MATLAB®

Why is there no implicit "this" object in MATLAB?

In some languages, one object parameter to a method is always implicit. In such languages, a method always acts on a single scalar object without any need to access elements of an object array.

Consider a MATLAB method that acts on an object array:

```
function S = sum(X)
    S = 0;
    for k = 1:length(X)
        S = S + X(k).Value;
    end
end
```

While languages with an implicit object parameter provide a "this" keyword to access the implicit object, they usually do not require you to access a property through "this". If MATLAB had implicit properties, the logical extension to array-based objects would be to index into nothing:

```
S = S + (k).Value;
```

Inheritance

In R2008a, MATLAB introduces a new inheritance model based on the idea that a class defines a set of objects with common traits and behaviors. A subclass defines both a subset of objects that exhibit the traits and behaviors defined by the superclass and additional traits and behaviors not exhibited by all instances of the superclass. Before R2008a, the fact that MATLAB objects were implemented using `struct` arrays was highly visible and central to the way classes inherited from other classes. An object belonging to a subclass was a `struct` that contained a field for each superclass, and that field contained an object belonging to the superclass.

This approach is, in many ways, simple and elegant. For example, it is easy to construct subclass instances using superclass constructors without special syntax, and it is easy to call superclass methods from subclass methods—you simply call the method on the superclass object contained by the subclass. However, there were problems with this model of inheritance.

One deficiency with the pre-R2008a inheritance model was that a superclass method could be called from a subclass method only by passing the superclass component of the object. This meant that the superclass method did not receive the whole object,

and calls made by the superclass method would not dispatch to the subclass.

The new inheritance model addresses this problem by treating each object semantically as an indivisible whole, regardless of underlying implementation details. Calls to superclass methods are made using syntax that avoids the need to have an instance that belongs to the superclass but not to the subclass. In the new model, an instance of a subclass has all the properties and methods of its superclasses, but a subclass object is not a composite of superclass components (Figure 1).

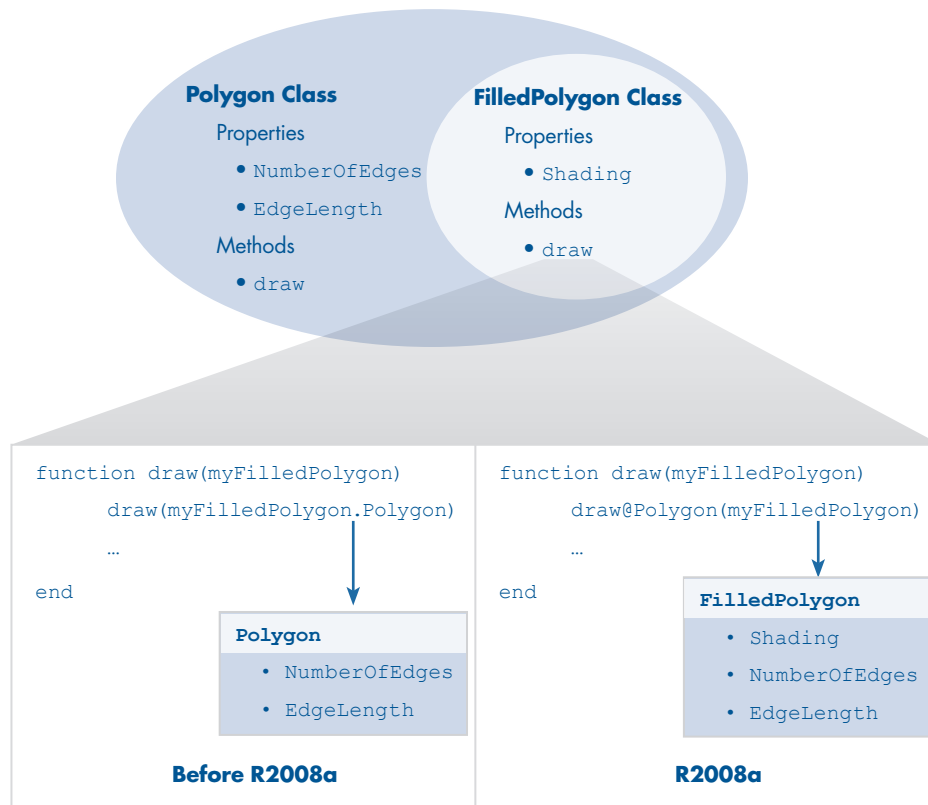


Figure 1. Comparison of subclass objects. In MATLAB, a subclass defines a subset of objects. Each object in the subclass is a member of the superclass, but has additional features beyond those defined by the superclass. Prior to R2008a, subclass instances contained actual superclass instances that could be used to invoke superclass methods. In R2008a, instances of classes defined using `classdef` don't contain instances of superclasses, but instead support the ability to call superclass methods on the whole subclass instance. In this way, the object maintains a single identity throughout the operation.

Handles

MATLAB defines the `handle` class as representing objects whose identity is independent of the values of their properties. Just like a person or thing, handle objects can change state without losing their identity. Handles tend to represent tangible things like windows or files rather than conceptual or mathematical entities like numbers (Figure 2). They are useful for creating certain kinds of data structures. For example, the nodes of a tree can be handle objects. Each node has a unique identity based on its location in the tree, but the data associated with that node can change over time.

Since MATLAB 4, handles have been available in Handle Graphics®. Handles provided access to objects that represented figures, axes, lines, and other visual elements.

Unlike numbers, each graphical element represented a unique visual display. A particular line displayed a particular piece of data with a particular style. You could change this line style without creating a new one because its handle identified it. The line handle could be passed to a function without copying the object and creating a second line, letting you write functions that acted on existing handles and use those functions in different contexts with different handles.

R2008a lets you create your own handle classes as subclasses of `handle`. This capability is useful when you want to create new kinds of objects that represent unique physical entities or abstract data structures that involve connections among objects.

Properties

MATLAB class definitions can contain properties in addition to methods. Properties are the preferred way for a MATLAB class to provide access to data stored in objects. A key goal of object-oriented programming is to hide the internal workings of a class behind a user interface. This approach allows the class implementation to evolve without forcing costly changes in the code that uses the class. To avoid these kinds of changes, programmers in languages such as C++ or Java™ are taught to keep data members private and to provide public access methods, even if these methods merely assign or return a private data member.

If MATLAB adopted a similar approach, there would be implications not seen in C++ and Java. MATLAB supports complex array indexing operations that cannot be performed as easily using method calls. For example, compare this simple and direct assignment using field notation:

```
x.Data(1:2:end, 1:2:end) = y;
```

with this more lengthy and complicated use of access methods:

```
temp = x.getData;  
temp(1:2:end, 1:2:end) = y;  
x.setData(temp);
```

Some languages provide the concept of properties in addition to fields. Properties use the same syntax for access and assignment as fields, but they allow the class to define additional behaviors when the property is accessed or assigned. A property in MATLAB can be as simple as a field—just storing a value. Access methods can be added later without changing the way the object data is accessed. There was no reason to distinguish fields from properties, and so the MATLAB `classdef` just has properties.

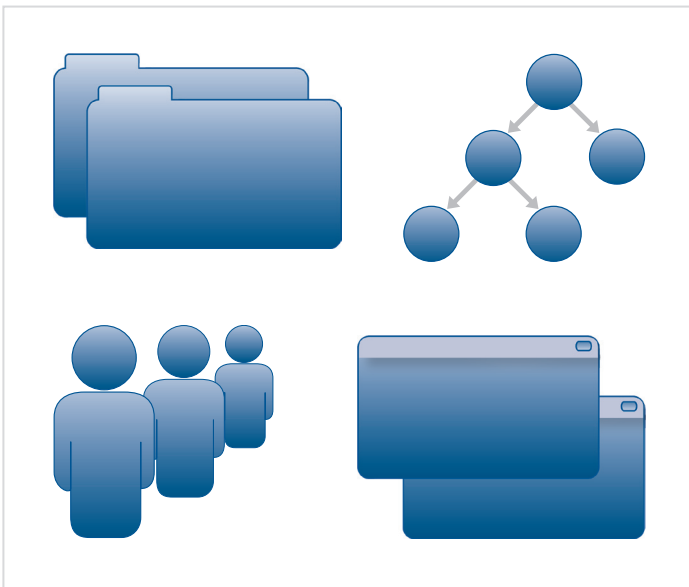


Figure 2. Examples of objects that are good candidates for representation as handles. Clockwise from upper left: folders or files, the nodes of a tree data structure, windows, and individual members of a population.

Object Life-Cycle Management

MATLAB implements a strategy for object life-cycle management that supports destructors and destroys objects as soon as they become unreachable from any MATLAB workspace. The MATLAB language has always managed memory allocation by destroying workspace variables when a function exits, either through an error or through a normal return to the calling context. We wanted to preserve this simple model but also extend it to handle objects that might persist beyond the execution of a function (by being returned to a calling function or stored in some other object returned to a calling function).

A MATLAB handle class can define a delete method that behaves very much like a destructor in languages like C++ (Figures 3a and 3b). In MATLAB, a delete method is called just before an object is destroyed because the object can no longer be accessed from any MATLAB variable. The delete method can be used to close a file, close an

external application, or notify another object that needs to react to the destruction of the first object. It is defined by the handle class, and only handle classes have destructors in MATLAB. This is because MATLAB uses the handle class to represent tangible objects and objects with unique identities. Conceptual objects never really cease to exist—for example, the number 5 doesn't cease to exist because it is no longer held within a variable.

Some programmers might find it strange that non-handle objects have no destructors. Since MATLAB object destruction is always automated, a non-handle object that holds other objects does not need to do anything to destroy those other objects. If a class represents some external resource that must be released when no longer needed, then a handle class should be used to represent that resource since it cannot really be copied and passed around by value in MATLAB.

Why not use a garbage collector in MATLAB?

We wanted to avoid the use of a garbage collector in MATLAB because of the complexity associated with managing object lifecycles in a garbage-collected environment. For example, current garbage-collected environments give up destructors. This means that they automate memory management but not total object management. Objects frequently use resources other than memory, and it is difficult to implement garbage collection that can manage all kinds of resources with varying levels of scarcity. Another disadvantage is that garbage collection can make program testing and debugging more difficult because activities that happen during garbage collection cannot easily be repeated.

C++

```
class Widget {
public:
    Widget() : m_RefCount(1) {
    }

    void acquireRef() {
        m_RefCount++;
    }

    void releaseRef() {
        m_RefCount--;
        if (m_RefCount == 0) {
            delete this;
        }
    }
    ...

private:
    size_t m_RefCount;
    ...
};

class WidgetHandle {
public:
    WidgetHandle() {
        m_WidgetRef = new Widget();
    }

    WidgetHandle(const WidgetHandle &other) {
        m_WidgetRef = other.m_WidgetRef;
        m_WidgetRef->acquireRef();
    }

    WidgetHandle &operator=(
        const WidgetHandle &other) {
        Widget *w = other.m_WidgetRef;
        w->acquireRef();
        m_WidgetRef->releaseRef();
        m_WidgetRef = w;
    }

    ~WidgetHandle() {
        m_WidgetRef->releaseRef();
    }

private:
    Widget *m_WidgetRef;
};
```

Figure 3a. C++ code using a wrapper class. `WidgetHandle` provides a stack-based object that can automatically manage multiple references to a `Widget` pointer so that the `Widget` object gets destroyed when no longer in use.

MATLAB

```
classdef WidgetHandle < handle
    properties
        ...
    end

    methods
        ...
    end
end
```

Figure 3b. An equivalent MATLAB class. The MATLAB implementation uses one class because the `handle` superclass implements automatic destruction when the handle is no longer in use.

Summary

In R2008a, MATLAB adds many new features for defining classes of objects that build on and preserve the essential qualities of MATLAB. When creating MATLAB classes, it is useful to keep in mind the key features of MATLAB variables and functions. These include the fact that parameters and variables are commonly arrays and support array indexing and that all function

parameters are explicitly declared and named. Handle classes with destructors can represent external systems or resources. MATLAB handles provide some of the capabilities of references and pointers in other languages, including automatic memory management, while avoiding some of the pitfalls associated with pointers and garbage collection. ■

For More Information

- Introduction to Object-Oriented Programming in MATLAB
www.mathworks.com/8amatlab_oopintro
- Object-Oriented Programming Resources
www.mathworks.com/8amatlab_oopres
- Documentation on getting started with classes in MATLAB
www.mathworks.com/8amatlab_oopdoc

Resources

VISIT

www.mathworks.com

TECHNICAL SUPPORT

www.mathworks.com/support

ONLINE USER COMMUNITY

www.mathworks.com/matlabcentral

DEMOS

www.mathworks.com/demos

TRAINING SERVICES

www.mathworks.com/training

THIRD-PARTY PRODUCTS AND SERVICES

www.mathworks.com/connections

Worldwide CONTACTS

www.mathworks.com/contact

E-MAIL

info@mathworks.com

© 2008 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

91586v00 09/08