Object Oriented & Event-Driven Programming with MATLAB

Ameya Deoras
Agenda

Object-oriented programming in MATLAB

- Classes in MATLAB
- Advantages of object oriented design
- Example: Designing a portfolio tracker

Events in MATLAB

- Event-driven programming fundamentals
- Writing event handlers
- Example: Building a real-time portfolio tracker
Case Study: Portfolio Tracker

- Subscribe to real-time quotes for 4 equities from Reuters service
- Track real-time combined portfolio value Visualize instrument & portfolio history graphically in real-time
What is a program?

```
x = 12
while (x < 100)
    x = x+1
    if (x == 23)
        disp('Hello')
    end
end
```

**Code**

**Data**

```
x = 12
while (x < 100)
    x = x+1
    if (x == 23)
        disp('Hello')
    end
end
```

**Actions**

- Assignment
- Loops Test
- Increment
- Test to Act
- Take Action
- End
- End
Progression of Programming Techniques

Data

- value
- variable
- structure

Algorithm

- command line
- script
- function

Level of Abstraction / Sophistication
Progression of Programming Techniques

Level of Abstraction / Sophistication:
- command line
- script
- function
- structure
- variable
- value

Data:
- (properties)
- class
- (methods)
Object-Oriented Terminology

- **Class**
  - Outline of an idea
  - Properties (data)
  - Methods (algorithms)

- **Object**
  - Specific example of a class
  - Instance

An element of the set – *object*

Defined set – *class*

Eg. *Stock*
From Structures to Objects

Example: Class **MarketInstrument** with properties *Ticker* and *Price*.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Structure</th>
<th>Class/Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization, Property Access</td>
<td>x = struct; x.Ticker = 'GOOG'; x.Price = 550;</td>
<td>x = MarketInstrument; x.Ticker = 'GOOG'; x.Price = 550;</td>
</tr>
<tr>
<td>Erroneous property name</td>
<td>x.ticker = 'MSFT';</td>
<td>x.ticker = 'MSFT'; No public field ticker exists for class MarketInstrument</td>
</tr>
<tr>
<td>Erroneous property value</td>
<td>x.Price = 'blah';</td>
<td>x.Price = 'blah'; Error using MarketInstrument/set.Price Price must be a scalar numeric value</td>
</tr>
</tbody>
</table>
Unique Advantages of Classes & Objects

- Property Access Control, Error Checking
  - Specify default values
  - Create constants
  - Make values interdependent
  - Execute methods when properties change

```plaintext
properties
  Components % Components of portfolio
  Quantity % Number of units of each component
end

properties (SetAccess = protected)
  CurrentCompPrice % Up-to-date component price
end

properties (Dependent)
  CurrentPrice % Up-to-date price of portfolio
end
```
Unique Advantages of Classes & Objects

- Property Access Control, Error Checking
  - Specify default values
  - Create constants
  - Make values interdependent
  - Execute methods when properties change

```plaintext
>> x.Price = 'twelve';
set.Price(obj, val)
assert(isnumeric(val))

Error using set.Price
Price must be a numeric value
```
Unique Advantages of Classes & Objects

- Property Access Control, Error Checking
- Reference Semantics

```matlab
classdef Instrument

function myFun(y)
    y.Price = 15;
end

>> x = Instrument;
>> x.Price = 12;
>> myFun(x);

>> x.Price
ans =
    12
```
Unique Advantages of Classes & Objects

- Property Access Control, Error Checking
- Reference Semantics

```matlab
classdef Instrument < handle
    function myFun(y)
        y.Price = 15;
    end
end

>> x = Instrument;
>> x.Price = 12;
>> myFun(x);

>> x.Price
ans =
    15
```
Unique Advantages of Classes & Objects

- Property Access Control, Error Checking
- Reference Semantics
- Overload default functions & operations
  - plot, disp, size
  - [ ], +, /, &, ()

```plaintext
>> x.Price = 12;
>> x
My price is $12
```

```matlab
function disp(obj)
    disp(['My price is $'... num2str(obj.Price)])
```
Unique Advantages of Classes & Objects

- Property Access Control, Error Checking
- Reference Semantics
- Overload default functions & operations
- Automatic documentation

**MarketInstrumentFinal**

Class to represent live market instruments

Usage:
```
x = MarketInstrument('YHOO', 34)
```

**Property Summary**

- **LastUpdated**: Time stamp for the last update to the price (datenum)
- **Price**: Current price of the instrument (numeric scalar)
- **Ticker**: Ticker identifying instrument (any string)

**Method Summary**

- **disp**: Display instrument properties
- **setPrice**: Set the price of the instrument

**Event Summary**

- **ObjectBeingDestroyed**: Notifies listeners that a particular object has been destroyed.
- **priceChanged**: Notification that a price change has occurred
Unique Advantages of Classes & Objects

- Property Access Control, Error Checking
- Reference Semantics
- Overload default functions & operations
- Automatic documentation

- Support for MATLAB-like constructs
  - Dynamic allocation
  - Vectorized access of properties

```matlab
>> x(2) = MarketInstrument;

>> x(1).Price = 12;
>> x(2).Price = 15;
>> x(3).Price = 20;
>> y = [x.Price]
y =
    12    15    20
```
Unique Advantages of Classes & Objects

- Property Access Control, Error Checking
- Reference Semantics
- Overload default functions & operations
  - plot, disp, size
  - [ ], +, /, &,
- Automatic documentation
- Support for MATLAB-like constructs
  - Dynamic allocation
  - Vectorized access of properties

*Plus all the benefits of OOP in general (encapsulation, inheritance…)*
 Agenda

- Object-oriented programming in MATLAB
  - Classes in MATLAB
  - Advantages of object oriented design
  - Example: Designing a portfolio tracker

- Events in MATLAB
  - Event-driven programming fundamentals
  - Writing event handlers
  - Example: Building a real-time portfolio tracker
Introduction to Event-Driven Programming

Terminology

- **Event handler/Callback**: Function called to respond to event
- **Listener**: Object that monitors for a specific event & calls handler
- **Event data**: Data associated with event
## Procedural vs Event-Driven Implementation

### Instrument

<table>
<thead>
<tr>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>setPrice(newPrice)</td>
</tr>
<tr>
<td>display()</td>
</tr>
</tbody>
</table>

Goal: When Instrument price changes, display new price

<table>
<thead>
<tr>
<th>Procedural</th>
<th>Event-Driven</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>function</code> setPrice(obj, newPr)</td>
<td></td>
</tr>
<tr>
<td><code>obj.price = newPr;</code></td>
<td></td>
</tr>
<tr>
<td><code>display(obj)</code></td>
<td></td>
</tr>
<tr>
<td><code>end</code></td>
<td></td>
</tr>
<tr>
<td><code>function</code> setPrice(obj, newPr)</td>
<td></td>
</tr>
<tr>
<td><code>obj.price = newPr;</code></td>
<td></td>
</tr>
<tr>
<td><code>notify(obj, 'priceChanged')</code></td>
<td></td>
</tr>
<tr>
<td><code>end</code></td>
<td></td>
</tr>
<tr>
<td><code>addlistener(obj,...</code></td>
<td></td>
</tr>
<tr>
<td><code>'priceChanged', @obj.display)</code></td>
<td></td>
</tr>
</tbody>
</table>
Procedural vs Event-Driven Implementation

Goal: When Instrument price changes, display new price and update value of Portfolios

<table>
<thead>
<tr>
<th>Instrument</th>
<th>Portfolio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Price</td>
<td>Instruments</td>
</tr>
<tr>
<td>setPrice(newPrice)</td>
<td>Value</td>
</tr>
<tr>
<td>display()</td>
<td>updateValue()</td>
</tr>
</tbody>
</table>

Event-Driven

```plaintext
function setPrice(obj, newPr)
    obj.price = newPr;
    notify(obj,'priceChanged')
end
```

addlistener(instObj,'priceChanged', @instObj.display)
addlistener(instObj,'priceChanged', @portObj.updateValue)
Sources of Events in MATLAB

- MATLAB Graphics
- MATLAB Classes
- MATLAB timer Object
- External Interfaces (COM, Java, .NET)
- Datafeed Toolbox (Bloomberg, Reuters)
- Trading Toolbox (Bloomberg, TT, IB, CQG)
Putting it Together: Portfolio Tracker

- Subscribe to real-time quotes for 4 equities (MarketInstrument)
- Track real-time combined portfolio value (MarketPortfolio)
- Handle market events from Reuters service
- Visualize instrument & portfolio history graphically in real-time
Benefits of OOP in MATLAB

- Separate interface from implementation
  - More checks and balances and control over how your code is used

- Manage complex data types with familiar interfaces
  - Custom objects that override default operators and functions

- Pass by reference
  - Maintain one true copy of your data

- Use event-driven programming
  - Simplify design of complex real-time applications
Additional Resources

Object-Oriented Programming in MATLAB

Develop complex technical computing applications

Object-oriented programming is a formal programming approach that combines data and associated actions (methods) into logical structures (objects). This approach improves the safety and maintainability of code, particularly important when developing and maintaining large applications.

The object-oriented programming capabilities of the MATLAB® language allow you to develop technical computing applications faster than with other languages, such as C or FORTRAN. With object-oriented programming, you can create classes and apply standard object-oriented design patterns in MATLAB, including encapsulation, and reference behavior without engaging in the low-level details of the underlying languages.

Object-oriented programming in MATLAB involves using:

- Class definition files, enabling definition of properties, methods, and properties
- Classes with reference behavior, aiding the creation of data structures
- Events and listeners, allowing the monitoring of object property changes

Examples and How To

- Introduction to Object-Oriented Programming in MATLAB (MATLAB Central)
- Inside MATLAB Objects (MATLAB Digest article)
- Sample code comparisons (MATLAB Central)
  - MATLAB and C++

Training Services

MATLAB Programming Techniques

This course provides hands-on experience using the features in the MATLAB® language to write efficient, robust, and well-organized code. These concepts form the foundation for writing full applications, developing algorithms, and extending built-in MATLAB capabilities. Details of performance optimization, as well as tools for writing, debugging, and profiling code, are covered. Topics include:

- Creating robust applications
- Utilizing development tools
- Structuring code
- Structuring data
- Efficient data management
- Classes and objects

Training Formats

<table>
<thead>
<tr>
<th>Format</th>
<th>Course Length</th>
<th>Course Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classroom</td>
<td>2 Days</td>
<td>View details</td>
</tr>
<tr>
<td>Online instructor-l</td>
<td>2 days</td>
<td>View details</td>
</tr>
<tr>
<td>Online self-paced</td>
<td>90 days</td>
<td>View details</td>
</tr>
</tbody>
</table>
Questions and Answers