MATLAB Classes (1)

OOP and Classes

Object-oriented programming (OOP):

- A programming paradigm that focuses on treating entities in a task as "objects".
- Each object has its associated data (properties, attributes) and operations (methods, functions).
- A class is a "data type" definition of a certain kind of objects. An object is an "instance" of its class.

OOP and Classes

- Some advantages of OOP:
  - Encapsulation: The collection and management of everything associated with a class together.
  - Data hiding (data and/or operations are accessible to the user of the object only when necessary). Example: class "stack"
  - Inheritance and code reuse (example: classes "vehicle", "car" and "truck")
  - Common interfaces: The same syntax or operation names that work for different classes (example: classes "car", "truck", and "ship")

OOP and Classes

- Consider a class as a special data type that contains some internal data and some operations that are specifically for it.
- An example class that we can program: Fraction
  - Data: denominator and numerator
  - Operations: (arithmetic operations), etc.
- An example class that we can program: Polynomial
  - Data:
  - Operations:

MATLAB Classes

- Even what appears to be fundamental data types (`double`, `int8`, etc.) are classes in MATLAB.
- Custom MATLAB classes are defined in files that start with the keyword `classdef`.
- Managing MATLAB classes:
  - Method 1 (easier): Put all the code for the class in a single m file. The file name is the class name. We will only use this method.
  - Method 2 (for managing more complex classes): Put the code for the class in multiple files within the same directory. The directory name is ***@ClassName***.

Components in MATLAB Classes

- The two main components of a class definition:
  - Properties: Data in an object of the class
  - Methods: Functions on objects of the class
- Example:

```
classdef Point2
  properties
    x = 0
    y = 0
  end
  methods
    function v = norm(p)
      v = sqrt(p.x*p.x + p.y*p.y);
    end
  end
end
```

Components in MATLAB Classes

- Example use:

```
a = Point2
a.x = 1;
a.y = 2;
a.norm
```

- Some explanations:
    - Property initializers (not required)
    - For methods called with the form **`object.func`**: The first input argument is the calling object and therefore does not need to be specified by the calling statement. (There is no "**`this pointer`**" as in C++.)

Example Class: Fraction

```
classdef Fraction
  properties
    num = 0;
    den = 1;
  end
  methods
    function r = value(f)
      r = f.num / f.den;
    end
  end
end
```

Note the `classdef` is a block (ending with "`end`") and the `properties` and `methods` are sub-blocks (also ending with "`end`").

Class Constructors

- A constructor is a function with the same name as the class. Function header syntax:
  - `obj = ClassName(arguments)`
- The constructor is called whenever a new object of the class is created, and is used for its initialization.
- If no constructor is given, the default constructor is called that sets all the properties to their default values (if specified, or empty arrays).
- If you provide a constructor, it should handle the task of the default constructor (no input arguments) to be safe.

Constructor Example: Fraction

```
classdef Fraction
   …
   methods
      function obj = Fraction(n, d)
         if nargin == 0
            obj.num = 0;  obj.den = 1;
         elseif nargin == 1 && isscalar(n)
            obj.num = n;  obj.den = 1;
         elseif isscalar(n) && isscalar(d) && d ~= 0
            obj.num = n;  obj.den = d;
         else
            error('Input error!');
         end
      end
      …
   end
end
```

Overloading MATLAB Operators

- This is how we accomplish the "common interfaces".
- Each MATLAB operator has an associated function name. For example, binary "`+`" is `plus`.
- Provide functions with these particular names to overload the operators. Example function header:
  - `out = plus(a,b)`
  - This is called when at least one of `a` and `b` is an object of this class.
- Although operator overloading allows you much freedom in defining operator behaviors, keep them consistent with their "common meanings" as much as possible to avoid confusion.

Overloading MATLAB Operators

Some useful operators overloading function names (see the documentation for a complete list):

- Binary element-wise arithmetics: `plus`, `minus`, `times`, …
- Unary negation: `uminus`
- Relational: `eq`, `ne`, `lt`, `le`, `gt`, `ge`.
- Logical: `and`, `or`, `not`.
- Type casting: Use the target data type name as the function name.
- Others (more MATLAB specific): `disp`.

Operator Overloading Example: Fraction

```
classdef Fraction
  …
  methods
    …
    function f = plus(f1, f2)
      f = Fraction(f1.num*f2.den + f2.num*f1.den, ...
                   f1.den*f2.den);
    end
    function t = eq(f1, f2)
      t = (f1.value == f2.value);
    end
    function disp(f)
      fprintf('%d/%d\n', f.num, f.den);
    end
    function v = uint8(f)
      v = uint8(f.value);
    end
    …
  end
end
```