Inheritance in Matlab Classes

E177 February 2, 2006

http://jagger.me.berkeley.edu/~pack/e177

Recall, classes give data an identity, and associate functions to the identity. A class can also inherit some parts of its identity and associated functions from another class. This is called inheritance, and allows for hierarchical layout of classes, and promotes reuse of methods.

@vehicle class (front wheel steering)

Simple constructor vehicle class

```matlab
function M = vehicle(xyloc,v,theta,omega,wh)
    M.Location = xyloc;     % 2-by-1
    M.Velocity = v;         % 2-by-1
    M.Orientation = theta;  % 1-by-1
    M.AngularRate = omega;  % 1-by-1
    M.RectDim = wh;         % 2-by-1
    M = class(M,’vehicle’);
end
```

An ismoving method

```matlab
function B = ismoving(A)
    B = norm(A.Velocity)>0;
end
```

An isrearwheelskidding method...

SomeCarMethod.m

A method for objects of class car can reference and modify the inherited vehicle object

```matlab
function M = SomeCarMethod(A)
    % This function can access fields...
    A.BreakPressure % class=double
    A.SteeringAngle  % class=double
    A.DriveTorque    % class=double
    A.WheelRadius    % class=double
    A.Vehicle        % class=vehicle
end
```

Example: FrontWheelSlipAngle

Scope for inherited classes

At this point, we have

```matlab
vehicle.m
ismoving.m
isrearwheelskidding.m
@vehicle
SomeCarMethod.m
@car
```

Consider

```matlab
V1 = vehicle([2;3],[10;3],1.2,0.1,[1;4]);
C1 = car([0,0.2,100,190],[1;2],[5;4],[1,0],[2;4]);
ismoving(V1)
ismoving(C1)
SomeCarMethod(V1)
SomeCarMethod(C1)
```

Inside ismoving, only the fields of vehicle class can be accessed directly, even though input arg is a car object. Methods in @car can be called.
@bike class
Simple constructor bike class, inheriting vehicle

```matlab
function M = bike(FCR, RCR, F, R, SA, RA, xy, v, theta, wh)
    Veh = vehicle(xy, v, theta, wh);
    M.FrontChainRing = FCR;
    M.RearChainRing = RCR;
    M.FrontGear = F;
    M.RearGear = R;
    M.SteeringAngle = SA;
    M.RollAngle = RA;
    M = class(M, 'bike', Veh);
end
```

@wheelchair class
Simple constructor wheelchair class, inheriting vehicle

```matlab
function M = wheelchair(RW, Lw, Rw, xy, v, theta, wh)
    Veh = vehicle(xy, v, theta, wh);
    M.RearWheelRadius = RW;
    M.LeftWheelRate = Lw;
    M.RightWheelRate = Rw;
    M = class(M, 'wheelchair', Veh);
end
```

Front wheels are very small and light, so perhaps...

```matlab
function A = FrontWheelSlipAngle(W)
    A = 0;
end
```

Folder layout
Each class has a constructor and methods.

```
vehicle.m
vehicle.m
vehicle.m
 vehicle.m
   ismoving.m

@vehicle

bike.m
incrementFrontChainRing.m
decrementFrontChainRing.m
@bike

wheelchair.m
FrontWheelSlipAngle.m
wheelchair.m
@wheelchair
```

@ReferAssignGetSet
Constructor (see actual code) just keeps track of the options to handle decisions for each of the cases described.

```matlab
function A = ReferAssignGetSet(InputArgsSpecifiers)
    A.PropertyRef = 'followGet' ; % or 'disallow'
    A.PropertyAssign = 'followSet' ; % 'disallow'
    A.ParenRef = 'ProgSupplied' ; % 'builtin', 'disallow'
    A.ParenAssign = 'ProgSupplied' ; % 'builtin', 'disallow'
    A.BraceRef = 'ProgSupplied' ; % 'builtin', 'disallow'
    A.BraceAssign = 'ProgSupplied' ; % 'builtin', 'disallow'
    A.AllowSetAssignin = 0 ; % 1
    A = class(A, 'ReferAssignGetSet');
end
```

Questions/Options for a given class
- Is property reference allowed, as in `Object.Property`?
- Is property assignment allowed, as in `Object.Property = Value`?
- Is () reference allowed, as in `Object(Idx1, Idx2)`?
- Is () assignment allowed, as in `Object(Idx1, Idx2) = Value`?
- Does set use `assignin` when called with no output arguments?

```matlab
set(A, Prop, Value) % uses inputname, overwrites A
A = set(A, Prop, Value) % overwrites A, directly
B = set(A, Prop, Value) % A unchanged
```
@NewClass

```matlab
function A = NewClass(itsdata)
A.Field = itsdata;
A = class(A,'NewClass',ReferAssignGetSet('disallow'...));
```

Folder layout

```
@ReferAssignGetSet
get and set call GetSetPropertyInfo

function [GPN,SPN,SPD] = GetSetPropertyInfo(A)
% Return a N-by-1 cell array of GET Prop Names
% Return a M-by-1 cell array of SET Prop Names
% Return a M-by-1 cell array of SET Property
% Descriptions
get and set also call PrivateGet and PrivateSet

function Value = PrivateGet(Object,GPropName)

function B = PrivateSet(Object,SPropName,Value)
```

Pseudocode for get.m

```matlab
function V = get(A,Property)
[GPN,SPN,SPD] = GetSetPropertyInfo(A); % Get Prop Info
if nargin==2
    FullPropName = findinlist(Property,GPN);
    V = PrivateGet(A,FullPropName);
else
    % Create structure V1 with fieldnames equal % to the property names in GPN. Then fill % the values of the struct with the values % calling PrivateGet.
    if nargout==1; V = V1;
    else; disp(V1)
    end
end
```

Pseudocode for PrivateGet.m

```matlab
function V = PrivateGet(A,FullPropName)
switch FullPropName
    case PropertyName1
        V = % extract Property1 from fields of A
    case PropertyName2
        V = % extract Property2 from fields of A
    case ...
    case PropertyNameK
        V = % extract PropertyK from fields of A
    Otherwise
        error('Unknown Property');
end
```

Similar for Assignment/Reference

```
General purpose subsref, subsasgn
    % reside in @GetSetReferAssign
    % they call the object’s specific reference and assignment methods
    % ParenthesisReferenece % ParenthesisAssign
    % BraceReference % BraceAssign

In this way, one version of subsref and subsasgn address the issues
    % Proper relation to get/set % Deep references/assignments

Similarly, there should be tight integration with horzcat and vertcat, but at this point, we leave that to the programmer.
```
Look at example: @point

```matlab
function A = point(X,Y,varargin)
    S.X = X;  S.Y = Y; % point in 2-dim Euclidean space
    PropRef = 'followGet';  % or 'disallow'
    PropAssign = 'followSet';  % or 'disallow'
    ParenRef = 'builtin'; % or 'builtin' or 'disallow'
    ParenAssign = 'builtin'; % or 'builtin' or 'disallow'
    BraceRef = 'disallow'; % or 'ProgSupplied' or 'builtin'
    BraceAssign = 'disallow'; %or 'ProgSupplied', 'builtin'
    ASA = 1; %  or 0, ASA-> AllowSetAssignin
    B = ReferAssignGetSet(PropRef,PropAssign,...
                               ParenRef,ParenAssign,BraceRef,BraceAssign,ASA);
    A = class(S,'point',B);
    lvin = length(varargin);
    if lvin>0 && ceil(lvin/2)==floor(lvin/2)
        A = set(A,varargin{:});
    end
```

Examples using @point (try these)

```matlab
data = point(2,4);
n = get(data)
n = get(data,'Radius')
n = p.Angle
n = p.An
n = get(data,'Angle')
>> p.XCoord = 4;
>> p.Angle*180/pi   % convert to degrees
>> set(p)
>> p2 = point(-5,10);
>> M = [p p2;point(1,1) point(3,-3)];
>> M(2,1).XCoord
>> M(2,1).XCo = 3;
>> M(2,1).An
```

Example output:

```
>> p = point(2,4)
data = point(2,4)
>> get(p)
data.X = 2
   data.Y = 4
   data.A = 0
   data.R = 2.8284
   data.T = 0
   data.F = 1
   data.S = 0
   data.P = 0
>> get(p,'Radius')
data.R = 2.8284
>> p.Angle
   ans = 0
>> p.An
   ans = 0
>> get(p,'Angle')
data.T = 0
>> p.XCoord = 4;
data.X = 4
>> p.Angle*180/pi   % convert to degrees
   ans = 0
>> set(p)
>> p2 = point(-5,10);
>> M = [p p2;point(1,1) point(3,-3)];
>> M(2,1).XCoord
   ans = -5
>> M(2,1).XCo = 3;
>> M(2,1).An
   ans = 0
```

Why do these work?