Many ideas of
get, set, subsref, subsasgn
will to carry over to all classes we write. For example,
– Reference of M.Value is the same as get(M,'Value')
– Multi-depth subsref has specific recursive meaning
– Assignment of M.Value is the same as set(M,'Value',B)
– Multi-depth subsasgn has specific recursive meaning
– In some cases, () reference and assignment should be handled by the
  built-in methods for Matlab arrays
– set should work with or without output arguments
– Special Matlab-like conventions for get and set should be followed
  • Calls with 1-input argument
  • Property names can be partially specified
  • Multiple args in set(A,P1,V1,P2,V2)
This is a good motivation to write a class that handles all of this
functionality, and all other classes will simply inherit this.

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)
  • if so, are builtin methods used, or does the class programmer supply the reference
    method?
– Is () assignment allowed, as in Object.(Idx1,Idx2) = Value
  • if so, are builtin methods used, or does the class programmer supply the
    assignment method?
– Is {} reference allowed, as in Object{Idx1,Idx2}
  • if so, are builtin methods used, or does the class programmer supply the reference
    method?
– Is {} assignment allowed, as in Object{Idx1,Idx2} = Value
  • if so, are builtin methods used, or does the class programmer supply the
    assignment method?
– Does set use assignin when called with no output arguments?

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

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 = 'disallow'; % or 'ProgSupplied' or 'builtin'
A.BraceAssign = 'disallow'; % or 'ProgSupplied' or 'builtin'
A.AllowSetAssignin = 0;  % or 1
A = class(A,'ReferAssignGetSet');

Also need general purpose versions of
get, set, subsref, subsasgn

Example child class: @point

function A = point(X,Y,varargin)
    S.X = X;  S.Y = Y; % point in 2-dim Euclidean space
    PropRef = 'followGet';  % or 'disallow'
    PropAssign = 'followSet';  % '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
@ReferAssignGetSet

get and set call the child’s GetSetPropertyInfo

\[
\begin{align*}
\text{function } & [\text{GPN,SPN,SPD}] = \text{GetSetPropertyInfo}(A) \\
& \% \text{Return a } N \times 1 \text{ cell array of GET Prop Names} \\
& \% \text{Return a } M \times 1 \text{ cell array of SET Prop Names} \\
& \% \text{Return a } M \times 1 \text{ cell array of SET Property Descriptions} \\
\end{align*}
\]

get & set also call child’s PrivateGet and PrivateSet

function Value = PrivateGet(Object,GPropName)

GPN = [CommonNames;OnlyGet];  % gettable properties
SPN = [CommonNames;OnlySet];  % settable properties
SPD = {'XCoordinate of Point';'YCoordinate of Point'};

Pseudocode for get.m

function V = get(A,Property)
[GPN,SPN,SPD] = GetSetPropertyInfo(A);
if nargin==2
    FullPropName = smartpropmatch(Property,GPN);
else  % nargin==1
    % 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

function V = PrivateGet(A,FullPropName)
switch FullPropName
    case 'XCoord'
        V = A.X;
    case 'YCoord'
        V = A.Y;
    case 'Radius'
        V = norm([A.X A.Y]);
    case 'Angle'
        V = atan2(A.Y,A.X);
    otherwise
        error(['Unknown Property. ']);
end

@point/PrivateGet.m

Function V = PrivateGet(A,FullPropName)
switch FullPropName
    case 'XCoord'
        V = A.X;
    case 'YCoord'
        V = A.Y;
    case 'Radius'
        V = norm([A.X A.Y]);
    case 'Angle'
        V = atan2(A.Y,A.X);
    otherwise
        error(['Unknown Property. ']);
end

Pseudocode for PrivateSet.m

function A = PrivateSet(A,FullPropName,Value)
switch FullPropName
    case 'XCoord'
        if isa(Value,'double') && isscalar(Value)
            A.X = Value;
        else
            error('XCoordinate should be real scalar');
        end
    case 'YCoord'
        if isa(Value,'double') && isscalar(Value)
            A.Y = Value;
        else
            error('YCoordinate should be real scalar');
        end
    otherwise
        error(['Unknown Property. ']);
end

@point/PrivateSet.m

Function A = PrivateSet(A,FullPropName,Value)
switch FullPropName
    case 'XCoord'
        if isa(Value,'double') && isscalar(Value)
            A.X = Value;
        else
            error('XCoordinate should be real scalar');
        end
    case 'YCoord'
        if isa(Value,'double') && isscalar(Value)
            A.Y = Value;
        else
            error('YCoordinate should be real scalar');
        end
    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
  - ParenthesisAssign
  - 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...

```matlab
@ReferAssignGetSet/subsref
function B = subsref(A,L)
    ReAsGeSe = A(1);
    switch L(1).type
    case '.'
        if isequal(ReAsGeSe.PropertyRef,'followGet')
            B = get(A,L(1).subs);
        else
            error('dot(.) reference not allowed. Use GET.');
        end
    case '()' case '{}'
    end
    if length(L)>1
        B = subsref(B,L(2:end));
    end
end
```

```matlab
@ReferAssignGetSet/subsasgn
function B = subsasgn(A,L,RHS)
    ReAsGeSe = A(1);
    switch L(1).type
    case '.'
        if isequal(ReAsGeSe.PropertyAssign,'followSet')
            if length(L)==1
                B = set(A,L(1).subs,RHS);
            else
                tmp = get(A,L(1));
                tmp = subsasgn(tmp,L(2:end),RHS);
                B = set(A,L(1).subs,tmp);
            end
        else
            error('dot(.) assignment not allowed. Use SET.');
        end
    case '()' case '{}'
    end
    if length(L)>1
        B = subsref(B,L(2:end));
    end
end
```
Examples using `@point` (try these)

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

Why do these work?

New Challenge: After you check these commands out, make Radius and Angle settable. Rule would be that when you change Radius, the Angle is unchanged, and similar for changing Angle. In each case, the underlying "raw" data (X and Y) have to get correctly modified.