

# Classes

## What is a class?

A class is a grouping of objects that share the same set of applicable actions but differ in personality. A class is always given a name, such as `Airplane`. The personality of an object is known as `local state` in Matlab parlance and the actions are called `methods`.

**Real world example 1: Airplane.** The personality of a particular object of class `Airplane` (also known as an airplane) includes: 1) the exterior paint scheme; 2) the pattern in which seats are arranged; 3) the number of seats; 4) many other such attributes which you can imagine (notice all the nouns). The actions associated with an airplane include: 1) take off; 2) land; 3) crash; 4) many other actions which you can imagine (notice all the verbs). The actions are common to all airplanes, but the local state of each airplane is different.

**Real world example 2: Person.** The personality of a particular object of class `Person` (also known as a person) includes: 1) gender; 2) race; 3) experience in life; 4) height; 5) weight; 6) many other such attributes which you can imagine (notice all the nouns again). The actions associated with a person include: 1) eat; 2) sleep; 3) party; 4) walk; 5) run; 6) many other actions which you can imagine - some more pleasurable than others (notice all the verbs). The actions are common to all people, but the local state of each person is different.

**Matlab example 1: double.** The personality of a particular object of class `double` (also known as a number) includes: 1) integer value; 2) fraction value. The actions associated with a number include: 1) multiply; 2) divide; 3) subtract; 4) add; 5) many other actions which you can imagine. The actions are common to all numbers, but the local state of each (that is, its value) is different.

**Matlab example 2: vector.** The personality of a particular object of class `vector` (also known as a vector or list) includes: 1) objects stored in the list; 2) length of the list; 3) maybe some other attributes. The actions associated with a vector include: 1) insertion of a given object into the vector at some location with respect to the first object in the vector; 2) finding an object in the vector that matches a specification; 3) removing a found object from the vector; 4) maybe other actions which you can imagine. The actions are common to all vectors, but the local state of each (that is, its contents) is different.

## Why do we need to use classes?

The world turns on classes (I just referenced another class, namely `planet`). Actually, we have been using classes all along but we did not tell you for fear that your brain would overheat and burn to a crisp. Matlab takes care of defining most important classes for us so we do not have to worry about building those classes. However, Matlab cannot anticipate every class we may need. So, Matlab gives us the tools to create our own classes to help us

solve the problems concerning us.

## How do we know what classes to build?

Consider defining a class for every type of object you can name that appears to be part of the problem at hand and for which a class is not already defined. Some will be obvious, some not. Try to imagine using the obvious ones. This should reveal requirements that make other necessary and/or useful classes obvious. Continue this refinement process to completion.

For lab 8/9 we can identify the following classes immediately: patient, doctor, receptionist. Then we think about how these will be used. We might start by imagining a doctor's office with people waiting in first-come-first-served order to be paired with a doctor. This suggests defining a queue class (surprisingly, Matlab does not offer this class) which can be used by the receptionist to keep track of waiting patients and available doctors. We continue by noticing that the situation in the doctor's office changes only at particular points in time and that there are only three kinds of events that could cause a change: 1) a patient enters the office; 2) a patient leaves the office because treatment is completed and this frees a doctor; 3) a doctor enters the office. This suggests defining an event class. An event object would then have a type and a receptionist would act on an event differently for each type. But how would the receptionist receive an event to work with? A little thought makes evident the possibility that there may be many pending events at any one time so this suggests an eventList class, an object of which may be used to hold events and from which a receptionist could pick a "next event." The action of the receptionist will be to take an event from the eventList, record statistics, and generate a new list of events that should be put on the eventList. This suggests some handler of the events which we may call an eventListManager. The receptionist could do the eventlist handling on its own but its functionality is complicated enough (or at least it will be seen as such) so it is much better to outsource the eventlist handling to an eventListManager.

*Recap:* For lab 8/9 we have identified the following classes to develop: `patient`, `doctor`, `receptionist`, `queue`, `event`, `eventList`, and `eventListManager`. Next comes the development of these classes.

## How do we build a class?

There are three things to worry about here: 1) Matlab syntax (what matlab requires you to tell it); 2) personality definition (that is, local state); 3) action definition (that is, methods). Each is described in turn in separate subsections.

## Matlab requirements:

1. A directory (folder) in which all user-defined classes will reside. For example, I made a directory called `Classes` from my “home” directory using `mkdir Classes`. Call this directory the `class root directory`.
2. Tell Matlab where this directory is using `addpath Classes`, for example, from your home directory.
3. For each class make a subdirectory of the `class root directory` with a name that begins with the character '@' and ends with the name of the class. For example, make the directory for the queue class from the `class root directory` using `mkdir @queue`. Call this a `class directory`.
4. In the `class directory` associated with the class you are building you will create `m` files for each action that can be applied to the objects of the class plus a constructor plus a display function. All these `m` files are functions. This means that they have a first line that looks something like this:

```
function ans = func1(arg1,arg2,arg3)
```

The constructor may have any number of arguments (including 0).

5. Build the constructor first. The constructor defines the personality of the class. If your class needs vectors, doubles, even objects of types you defined, it is also defined and initialized here. The first line of the constructor looks like this:

```
function object = classname(arg1,arg2,...)
```

where `classname` is the name of the class you are creating and `object` refers to an object of this class. The variable `object`, to which you can give any name you want, will be used in the body of the constructor to reference `fields` (data items) that are local to each and every object. For example, suppose you need a vector. Give it a name, say `lst`. Suppose you want to initialize it to an empty vector. Then you would include the line

```
object.lst = [];
```

in the constructor where `object` matches the name used to the left of = in the first (function) line of the constructor. The last line of a constructor looks like this:

```
object = class(object,'classname')
```

where `object` matches the name used to the left of = in the first (function) line of the constructor.

6. Build the methods. It is important to note two things. First, the argument list of the function you build to implement any method must have at least one argument and the first (leftmost) argument is *always* an object of the class the method is being built for. That object is the only means the method has of making changes to the fields of an object of the class. For example, in

```
function ans = classname(obj,arg2,arg3,...)
```

the variable `obj` is an object of class `classname`.

Second, the object that is passed into the function through the argument list is only a copy of the “calling” object and any changes made to it are not made to the original object unless one of two things are done. One way to change the original object is to have the copy returned by the function. For example like this:

```
function obj = classname(obj,arg2,arg3,...)
```

But this is often impractical because another value may need to be returned. In that case, and generally anyway, the copy can be sent back in using the following as the last line of the function (before `end`):

```
assignin('caller',inputname(1),obj)
```

where `obj` is the copy of the originally passed object.

## Defining personality:

1. All of the personality attributes of a class are defined in the constructor.
2. Attributes of a class are called `fields`. Each field is itself a member of a class. Each field has a name.
3. A field value is accessed or changed through a specified object and it is only that object whose field value is changed. The syntax required by Matlab is

```
object.field = value;
```

for changing the field value and

```
value = object.field;
```

for accessing the field.

4. The following is a constructor for the `event` class, annotated to illuminate what is going on.

```
function evt = event(arg1, arg2, arg3);
    evt.type = 0;           % There four fields: type and epoch
    evt.epoch = 0;         % are of type double, doc is of type
    evt.doc = doctor();    % doctor and pat is of type patient.
    evt.pat = patient();   % These are the default values.
    if nargin > 0 evt.type = nargin; end
    if nargin == 1         % Event is: doctor becomes available
        evt.doctor = arg1;
    elseif nargin == 2    % Event is: patient enters office
        evt.pat = arg1;
        evt.epoch = arg2; % time that patient will enter office
    elseif nargin == 3    % Event is: patient paired with doctor
        evt.doc = arg1;
        evt.pat = arg2;
        evt.epoch = arg3; % time that pairing will end
    end
    evt = class(evt,'event'); % this makes it a class!
end
```

## Defining methods (actions):

1. One method is built for each action that can be applied to an object of a class. The developer of a class must consider carefully what actions on objects of that class need to be implemented. An action is carried out through manipulation of the local state of an object.
2. For example, consider the `queue` class. The purpose of a queue object is to release items, upon demand, in the order they are stored. Thus, it is reasonable to suppose that the local state for this class is a vector. Obviously, important actions on queue objects would be the insertion and removal of items. When implementing the `remove` method one notices the need to check for an empty vector to head off a Matlab runtime error of trying to access a vector element that does not exist. This suggests implementing an `isempty` method and using it in `remove`. Further consideration leads to the implementation of a `peek` method which returns the next item that will be removed when `remove` is called.
3. It is important to develop a display method for each class. Here is a display method for the `event` class:

```
function display(evt)
    if evt.type == 1
        fprintf('Event: doctor %s becomes available',evt.doc);
    elseif evt.type == 2
        fprintf('Event: patient %s becomes sick at time %d',
            evt.pat,evt.time);
    elseif evt.type == 3
        fprintf('Event: doctor %s paired with %s until time %d',
            evt.doc,evt.pat,evt.time);
    end
end
```