

UNIVERSIDAD NACIONAL DE QUILMES

**Ingeniería en Automatización y
Control**

COMPUTADORES II

Proyecto Final:

**“POO en Matlab: Teoría y Ejemplo
de Aplicación”**

- **Docente: Lic. Pablo Barrientos**
- **Auxiliar a cargo: Leandro Silvestri**

Patricio Colmegna – Demian Garcia

Violini - Pablo Manchione

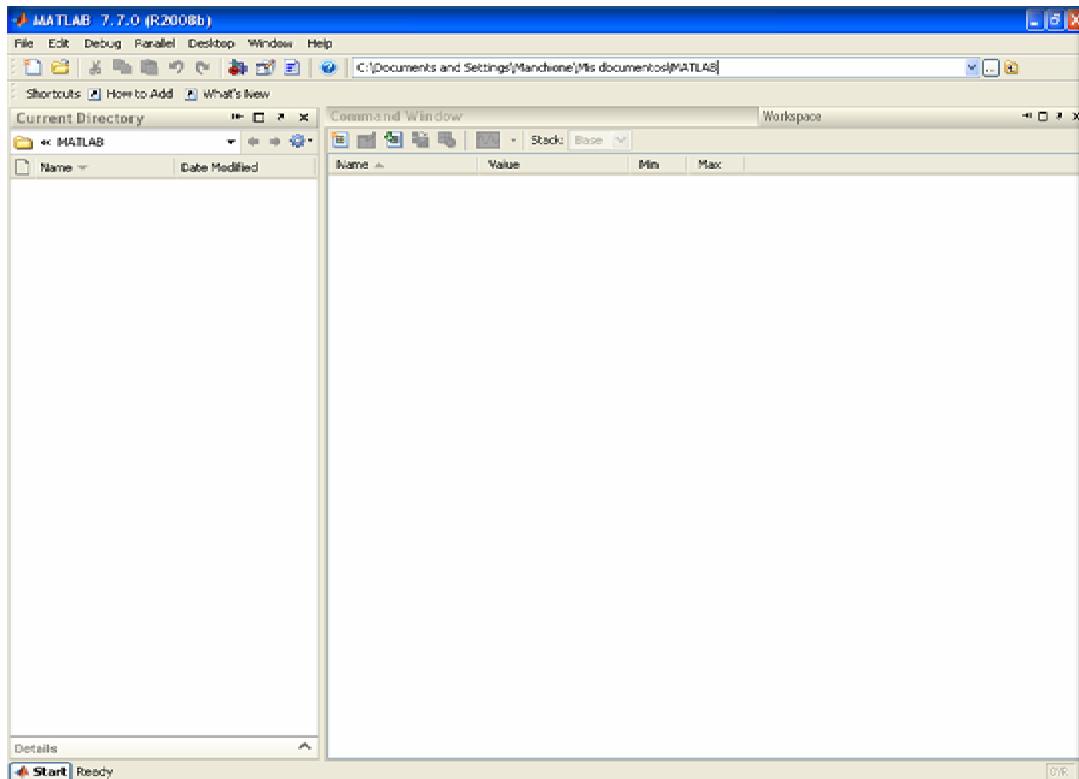
INDICE

0.	Introducción	2
0.1.	Un poco de Historia	2
0.2.	¿Qué es Matlab?	2
0.3.	Ejecución de Comandos y Funciones	3
0.4.	Variables.....	4
0.5.	La Ventana de Current Directory	5
0.6.	MAT-file	5
1.	Programación Orientada a Objetos en Matlab	6
1.1.	Introducción	6
1.2.	Clases en Matlab	7
1.3.	Atributos de Clase	8
1.4.	Organización de Clases en Directorios	9
1.5.	Propiedades	10
1.6.	Métodos.....	16
1.7.	Conceptos Generales de la POO en Matlab.....	23
2.	Diseño del Algoritmo	26
3.	Escalabilidad	42
3.1.	Escalabilidad de Espacio de Trabajo.....	42
3.2.	Escalabilidad del Tipo de Comunicación	46
4.	Conclusiones	49
4.1.	Sobre la Escalabilidad del Diseño	49
4.2.	Comparación entre POO y Programación Imperativa	49
4.3	Modelo del Diseño Implementado	49

0. Introducción a Matlab

0.1 Un poco de historia

La primera versión de Matlab data de los años 70, y fue diseñada como herramienta de apoyo para los cursos de Teoría de Matrices, Álgebra Lineal y Análisis Numérico. El nombre Matlab es un acrónimo: "MATrixLABoratory". Hoy en día, Matlab es un programa muy potente, con un entorno agradable, que incluye herramientas de cálculo científico, técnico y de visualización gráfica, así como un lenguaje de programación de alto nivel.



0.2 ¿Qué es Matlab?

Matlab es un software matemático que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M). Está disponible para las plataformas Unix, Windows y Apple Mac OS X.

Entre sus prestaciones básicas se hallan: la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuario (GUI) y la comunicación con programas en otros lenguajes y con otros dispositivos hardware. El paquete Matlab dispone de dos

herramientas adicionales que expanden sus prestaciones, a saber, Simulink (plataforma de simulación multidominio) y GUIDE (editor de interfaces de usuario - GUI). Además, se pueden ampliar las capacidades de Matlab con las cajas de herramientas (toolboxes) y las de Simulink con los paquetes de bloques (blocksets).

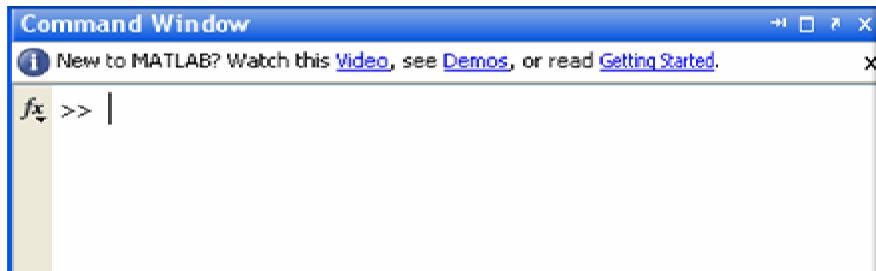
0.3 Ejecución de comandos y funciones

Existen dos modos de ejecución de los comandos y funciones:

- **Interactiva**
- **Por programa**

– **Interactiva:**

Matlab es un programa command-driven, es decir, que se introducen las órdenes escribiéndolas una a una a continuación del símbolo » (prompt) que aparece en la siguiente interfaz de usuario:

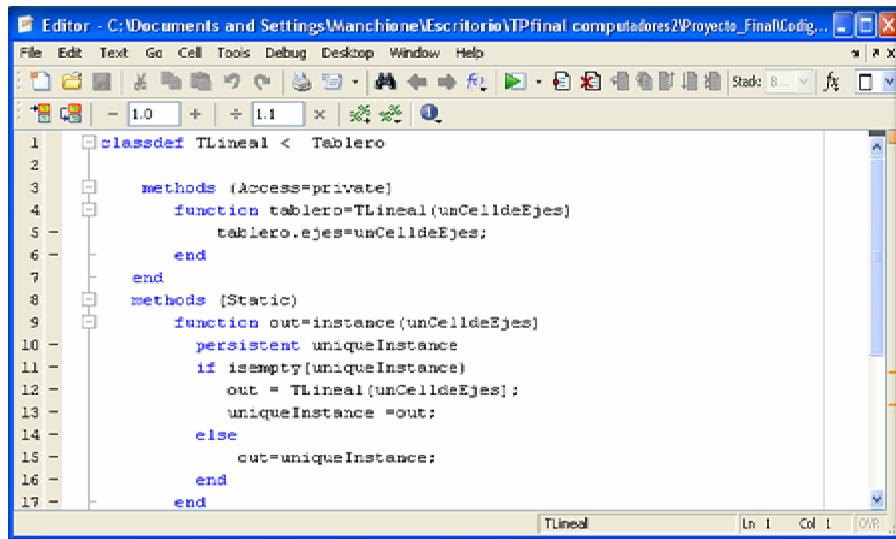


Con este modo se puede tipar directamente sobre el área de trabajo de Matlab el comando o la función a ejecutar, permitiendo una ejecución paso a paso, así como el seguimiento de las variables y el gráfico de los resultados intermedios.

– **Programa:**

Se tiene que crear un archivo con la secuencia adecuada de comandos y funciones que se requiera, este archivo se puede realizar desde cualquier editor o bien desde el editor de Matlab, asegurándose que el archivo sea llamado <nombre>.m (a estos archivos se les conoce como m-file). A continuación podemos ver en la siguiente figura, la interfaz gráfica del editor de Matlab.

Computadores II – Proyecto Final



```
Editor - C:\Documents and Settings\Manchione\Escritorio\TPfinal computadores2\Proyecto_FinalCodigo.m
File Edit Text Go Cell Tools Debug Desktop Window Help
+ 1.0 - 1.1 × * * ? 
1 classdef TLineal < Tablero
2
3     methods (Access=private)
4         function tablero=TLineal(unCelldeEjes)
5             tablero.ejes=unCelldeEjes;
6         end
7     end
8     methods (Static)
9         function out=instance(unCelldeEjes)
10            persistent uniqueInstance
11            if isempty(uniqueInstance)
12                out = TLineal(unCelldeEjes);
13                uniqueInstance =out;
14            else
15                out=uniqueInstance;
16            end
17        end
18    end
19
```

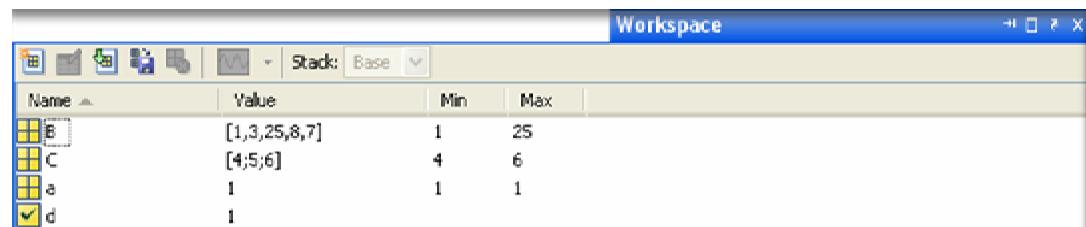
Las líneas que lo integran son líneas en las que están tecleados los comandos y funciones que ejecutará Matlab. Para la creación del archivo únicamente se tienen que seguir las reglas de sintaxis propias del lenguaje y guardarla con extensión ".m".

Por otro lado, para correr el programa hay que pasar al área de trabajo de Matlab y verificar que estamos en el directorio correcto o en su defecto cambiarlo por el que se encuentra el archivo que acabamos de crear. Por último, si la sintaxis es la correcta y no hay mensajes de error, obtendremos el resultado esperado.

0.4 Variables

Lo que hay que recalcar es que en Matlab, las variables, no necesitan declararse previamente y tampoco dimensionarse, como se hace en la programación tradicional.

La visualización de las variables se realiza a partir de la ventana del Workspace (área de trabajo):

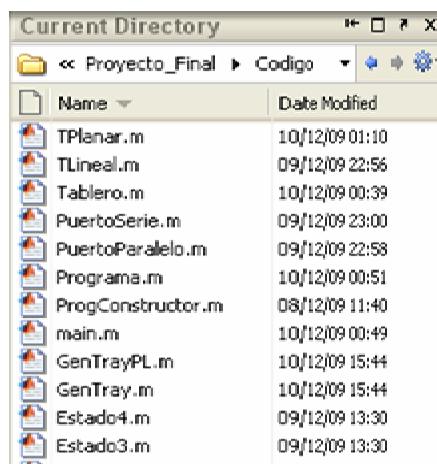


Name	Value	Min	Max
B	[1,3,25,8,7]	1	25
C	[4;5;6]	4	6
a	1	1	1
d	1		

Donde se pueden observar el nombre de la variable, lo que contiene cada una (Array, Cell, Integer, Logical, etc), el mínimo, el máximo y la dimensión de la misma.

0.5 La ventana del Current Directory

La ventana de Current Directory muestra los ficheros del directorio activo o actual. El directorio activo se puede cambiar desde la Command Window, a través de la ventana o desde la barra de herramientas debajo de la barra de menús con los métodos de navegación de directorios propios de Windows. Cliqueando dos veces sobre alguno de los ficheros *.m del directorio activo se abre el editor de ficheros de Matlab.



La ventana Current Directory permite explorar los directorios del ordenador en forma análoga a la del explorador u otras aplicaciones de Windows.

0.6 Mat-file

Este tipo de archivos son aquellos que permiten al usuario guardar el estado de las variables en un momento determinado. Para almacenar las variables en este tipo de archivos, se debe seleccionar "save as" en la barra de menú del espacio de trabajo.

1. Programación Orientada a Objetos en Matlab

1.1 Introducción

En Matlab, toda variable o valor es asignado a una clase. Por ejemplo, mediante un constructor con asignación creamos una variable de una clase determinada:

```
>> a = 7;  
>> b = 'some string';  
  
>> whos  
  
Name      Size            Bytes Class  
a          1x1              8 double  
b          1x11             22 char
```

Existe un tipo de variables que pueden contener objetos instancia de diversas clases, estos son los “cells”. Estos últimos son pueden ser vistos como colecciones y fueron incorporados a Matlab desde la versión 5.0 de este. Pueden ser n-dimensionales y como ya se dijo en cada una de sus posiciones pueden contener variables de diversos tipos. Veamos un ejemplo:

```
>> A=cell(3,2);  
  
>> whos  
  
Name      Size            Bytes Class  
a          3x2              24 cell
```

En este último ejemplo creamos un cell vacío de 3x2. Para referirnos a cada uno de sus elementos lo hacemos con el operador {} a diferencia de los “array” comunes en donde nos referimos con el operador (). Por ejemplo, si creamos un array de caracteres (string) nombre='Pablo Barrientos', la manera de referenciar al 4º elemento, por ejemplo, sería de la siguiente forma:

```
>> nombre(4)  
ans ="
```

Otra manera de construir un cell es mediante un constructor con asignación. Veamos el ejemplo del nombre={{'P'} {'a'} {'b'} {'l'} {'o'} {' '} {'B'} {'a'} {'r'} {'r'} {'i'} {'e'} {'n'} {'t'} {'o'} {'s'}}. De esta forma estamos creando un cell de cells inicializado. Por

este motivo si nosotros quisiéramos referenciar al 4º elemento y lo hiciéramos con el operador [], haríamos referencia al cuarto cell y no al carácter 'l'. La manera de hacer referencia adecuadamente al carácter 'l' es mediante el operador {}. Así obtenemos lo siguiente:

```
>> nombre{4}  
ans = "l"
```

1.2 Clases en Matlab

Clases de Matlab se definen en bloques de código, con sub-bloques para delimitar las definiciones de los distintos miembros de la clase. Los elementos que pueden definirse dentro de una clase de Matlab son:

- Propiedades (variables de instancia).
- Métodos.
- Eventos

1.2.1 Tipos de clases en Matlab

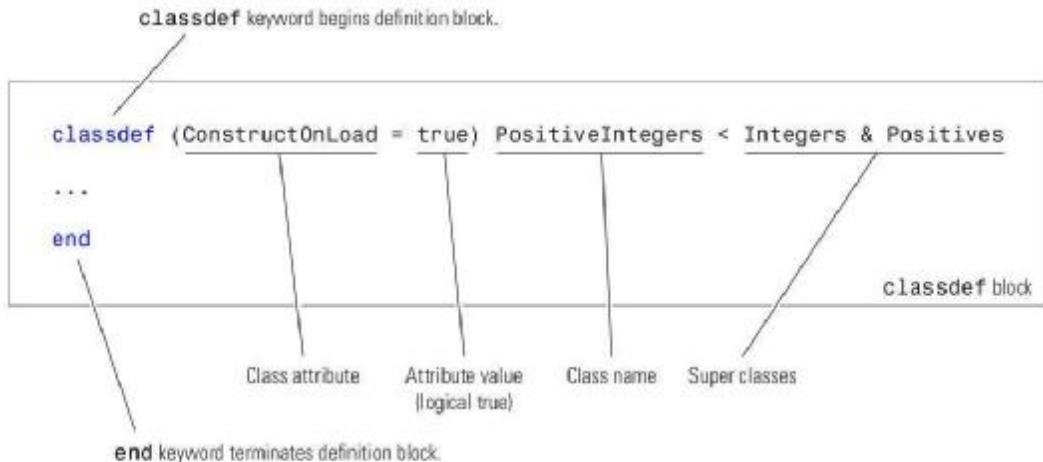
En Matlab existen dos tipos de clases, del tipo handle o del tipo value. Las clases del tipo handle instancia objetos que referencian al contenido y las copias de estos objetos también referencian a esos mismos datos. Las clases del tipo value, realizan copias del valor cuando el objeto se copia se pasa a una función. En Matlab las clases numéricas son del tipo value.

- **Clases Value:** cuando se copia un objeto value, Matlab también copia los datos contenidos en el objeto. El nuevo objeto es independiente de los cambios del objeto original.
- **Clases Handle:** los objetos instancia de una clase handle utilizan una referencia para referenciar los objetos de la clase. Esta referencia es justamente una variable que se encarga de identificar las distintas instancias de una clase. Cuando se copia un objeto handle Matlab copia la referencia pero no los datos almacenados dentro de las propiedades del objeto. La copia refiere a los mismos datos que el objeto original y los cambios en el objeto copiado se reflejan en el objeto original. Todas las clases handle son subclases de la clase abstracta Handle. Además de esto ser subclase de Handle da una serie de ventajas:
 - Heredar una serie de métodos útiles (Handle Methods).
 - Definición de Events y Listeners.

1.2.2 Definición de clases en MATLAB. Sintaxis del bloque classdef

Las definiciones de clases se llevan a cabo con mediante el bloque classdef. Estos llevan la palabra clave “classdef” (al principio) y la palabra clave “end”. Las

clases se definen en archivos .m y solamente pueden tener la definición de una clase. El siguiente esquema muestra la definición del bloque “classdef”. Sólo líneas en blanco y comentarios pueden preceder al bloque classdef.



1.3 Atributos de clase

<i>Nombre del Atributo</i>	<i>Clase</i>	<i>Descripción</i>
Hidden	Logical (default=false)	Si este atributo está en true, el nombre de la clase no saldrá como salida de los comandos de Matlab para enumerar clases.
InferiorClasses	Cell (default={})	Use este atributo para establecer la relación de precedencia entre las clases. Especifique un cell con el nombre de las clases (representados como objetos de metaclass), especificando que objetos son inferiores al presente. En la definición se debe utilizar el operador? Para poder definir objetos de metaclass ¹ .
ConstructOnLoad	logical (default=false)	Si este atributo está en true, Matlab llama al constructor del objeto cuando se carga un objeto de un MAT-file (archivo de variables). Por lo tanto se debe implementar un constructor que pueda

¹ Ver MATLAB 7. Object Oriented Programming, “Specifying Class Precedence” en la pagina 4-11.

		ser llamado sin argumentos y sin producir error. ²
Sealed	logical (default=false)	Si este parámetro esta en true, esta clase no puede ser subclase de otra.

1.3.1 Atributos dentro de classdef

Los atributos son especificados para cada uno de los bloques de definición de los miembros de classdef: propiedades, métodos y eventos. El valor del atributo se aplica a todos los miembros definidos dentro del bloque particular. Esto significa, por ejemplo, que pueden utilizarse múltiples bloques de definición de propiedades para así aplicar distintos atributos a distintas propiedades.

1.3.2 Atributos de superclases

Los atributos de las clases no son heredados, por lo tanto, los atributos de las superclases no afectan a las subclases.

1.3.3 Sintaxis de los atributos

Se especifican los atributos entre paréntesis separando cada par atributo/nombre-valor mediante comas. La lista de atributos siempre sigue al classdef o al miembro de clase como se muestra a continuación:

```
classdef (attribute-name = expression, ...) ClassName

    properties (attribute-name = expression, ...)
        ...
    end
    methods (attribute-name = expression, ...)
        ...
    end
    events (attribute-name = expression, ...)
        ...
    end
end
```

1.4 Organización de clases en directorios

Existen dos tipos de directorios que pueden contener definición de clases. Cada uno posee diferencias en múltiples aspectos.

- **@-directories:** El nombre del directorio comienza con @ y no se encuentra

² Ver MATLAB 7. Object Oriented Programming, "Calling Constructor When Loading" en la pagina 7-24

en el path de MATLAB pero su directorio raíz si se encuentra en el path de MATLAB. Este tipo de directorios se pueden utilizar cuando se utilizan varios archivos en una misma definición de clase. Puede ser solamente una clase con directorio y el nombre de la clase debe coincidir con el nombre del directorio sin el símbolo @.

- **Path directories:** El nombre del directorio no hace uso del símbolo @ y es en si mismo el path de MATLAB. Este tipo de directorios para definir varias clases en un mismo directorio.

1.5 Propiedades

Las propiedades permiten encapsular los datos que pertenecen a las instancias de las clases. Los datos contenidos pueden ser públicos, protegidos o privados. Estos datos pueden ser del tipo constantes, o pueden depender de otros valores y se calcula solamente cuando se pregunta por ellos. El comportamiento de las propiedades se establece mediante la asignación de atributos al miembro properties dentro de classdef y definiendo los métodos de acceso a dichas propiedades (getters y setters).

1.5.1 Flexibilidad de las propiedades de los objetos

En cierto modo, las propiedades son como los campos de un objeto “struct”. Sin embargo el almacenamiento de datos en una propiedad del objeto proporciona una mayor flexibilidad. Las propiedades pueden:

- Definir valores constantes que el usuario no pueda cambiar fuera de la definición de clase.
- Calcular su valor basándose en el valor de otra propiedad.
- Asignar una función para corroborar si un intento de asignar algún valor a esa propiedad verifica ciertos criterios.
- Desencadenar una notificación de eventos cuando se realiza algún intento para obtener o establecer su valor.
- Restringir el acceso de otro código para el valor de la propiedad. Ver los métodos SetAccess y GetAccess.
- Controlar si su valor se guarda un archivo MAT-file.

1.5.2 Tipos de propiedades

Existen dos tipos de propiedades:

- Propiedades almacenadas (stored properties). Usan memoria y son parte del objeto.
- Propiedades Dependientes (dependent properties). No están almacenadas en memoria y el getter calcula el valor cuando se lo llama.

Como ventajas de las propiedades almacenadas pueden mencionarse:

- Puede asignarse un valor inicial en la definición de clase.

- El valor de la propiedad se guarda cuando se guarda al objeto en un MAT-file.
- Se puede utilizar el setter para controlar los posibles valores que se almacenaran, pero no se esta obligado a utilizar tales métodos.

Como ventajas de las propiedades dependientes pueden mencionarse que estas permiten ahorrar memoria ya que estos valores dependen del resto y solo se calculan cuando se necesitan.

1.5.3 Propiedades constantes

Se pueden definir propiedades constantes a cuyos valores se puede acceder por su nombre. Para asignar a una clase con atributos constante se debe asignar el valor true al atributo Constant. De esta forma se establece el acceso para el seteo (setters) como privado. De modo que los valores no pueden cambiar desde fuera de la clase.

Asignación de propiedades constantes

La asignación se debe realizar como se muestra en el siguiente ejemplo:

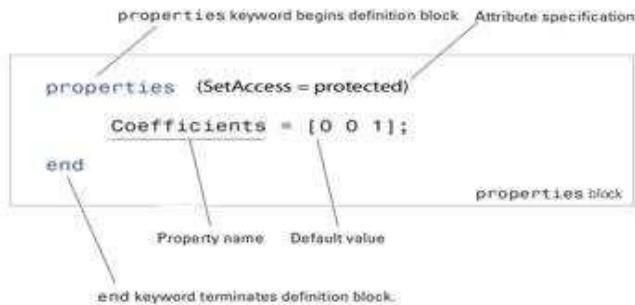
```
classdef NamedConst
    properties (Constant)
        R = pi/180;
        D = 1/RadDeg.R;
        AccCode = '0145968740001110202NPQ';
        RN = rand(5);
    end
end
```

Matlab evalúa las expresiones cuando se carga la clase (cuando se hace la primera referencia a una propiedad constante de esa clase). Esto significa que los valores asignados a RN son el resultado de única llamada a la función rand y no cambian cada vez que se pide su valor. Para poder cambiar este valor es necesario borrar las clases con “clear classes”.

1.5.4 Definición general de propiedades

La siguiente ilustración muestra una especificación de propiedades típica. Las palabras clave property y end delinean el bloque de código para definir propiedades para un conjunto de estas con los mismos atributos.

Computadores II – Proyecto Final



El ejemplo anterior muestra que la propiedad denominada Coeficientes posee un valor pude defecto igual al vector [0 0 1].

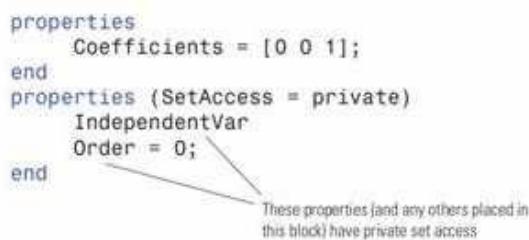
Se pueden inicializar valores mediante expresiones de Matlab, sin embargo estas expresiones no pueden referirse a la clase que se esta definiendo salvando los métodos de clase. Matlab ejecuta las expresiones que inicializan en algún valor a las propiedades solo cuando la inicialización de la clase se produce justo antes del primer uso de la clase.

1.5.5 Herencia de las propiedades

Cuando una clase hereda de otra, esta hereda las propiedades de la superclase. En general, las subclases solo definen las propiedades que son de uso exclusivo de esa clase en particular. La superclase define aquellas propiedades que son de uso común por todas las subclases.

1.5.6 Especificación de los atributos de las propiedades

La asignación de atributos se realiza como se muestra en la siguiente figura:



1.5.7 Propiedades de los atributos

La totalidad de atributos que puede poseer una propiedad determinada se enumeran en la siguiente tabla.

Computadores II – Proyecto Final

Nombre del Atributo	Clase	Descripción
AbortSet	logical default=false	Si este atributo esta en true y se trata de una clase tipo handle, entonces Matlab no setea el valor de esta propiedad si el nuevo valor es idéntico al anterior.
Abstract	logical default=false	Si este atributo se encuentra en true, la propiedad no tiene implementación, pero una subclase concreta debe redefinir esta propiedad sin el atributo Abstract seteado en true. <ul style="list-style-type: none"> • Las propiedades abstractas no pueden definir getters y setters. • No se les puede asignar valores iniciales.
Access	char default=public	<ul style="list-style-type: none"> • public: acceso ir-restringido. • protected: acceso solo desde la clase o desde sus derivadas. • private: acceso solo desde la clase.
Constant	logical default=false	Con este atributo en true se obtiene un solo valor para esta propiedad en todas las instancias de esta clase. <ul style="list-style-type: none"> • Las subclases heredaran las propiedades constantes y tampoco podrán modificarlas. • Las propiedades constantes no pueden ser dependientes. • Se ignora el setter para estas propiedades.
Dependent	logical default=false	Si es false el valor de la propiedad se almacena en el objeto. Si es true, el valor no es almacenado en el objeto3.
GetAccess	enumeration	<ul style="list-style-type: none"> • public: acceso ir-restringido.

3 Ver MATLAB 7. Object Oriented Programming, “Dependent Properties” en la página 2-27, “Property Get Methods” en la página 10-14 y “Avoiding Property Initialization Order Dependency” en la página 7-21

	default=public	<ul style="list-style-type: none"> protected: acceso solo desde la clase o desde sus derivadas.
GetObservable	logical default=false	Si este valor es true y se trata de una propiedad de una clase tipo handle se pueden crear listener para acceder a esta propiedad. Los listeners son llamados cuando se consultan las variables.
Hidden	logical default=false	Determina si la propiedad se mostrara en los listados de propiedades que se realicen sobre la propiedad en cuestión.
SetAcces	enumeration default=public	<ul style="list-style-type: none"> public: acceso ir-restringido. protected: acceso solo desde la clase o desde sus derivadas. private: acceso solo desde la clase.
SetObservable	logical default=false	Si este valor es true y se trata de una propiedad de una clase tipo handle se pueden crear listener para acceder a esta propiedad. Los listeners son llamados cuando se modifican las variables.
Transient	logical default=false	Si esta variable esta en true la variable no se almacena cuando un objeto se guarda en un MAT-file.

1.5.8. Métodos de Acceso a las Propiedades.

Los métodos de acceso a la propiedad le permiten ejecutar código cada vez que las propiedades de los valores se referencian o se le asigna un nuevo valor. Estos métodos permiten:

- Ejecutar código antes de realizar una asignación para, por ejemplo, establecer un rango de valores posibles, chequear dimensiones, etc.
- Ejecutar código antes de devolver el valor de una propiedad determinada para:
 - Calcular el valor de la propiedad para no almacenarla.
 - Cambiar el valor de la propiedad.
 - Eventos de trigger y así sucesivamente.

Matlab no posee métodos de acceso por defecto. Por lo tanto, si no se definen métodos de acceso a las propiedades, Matlab no invoca ningún método antes de asignar o devolver el valor de la propiedad.

Definición De Los Métodos de Acceso.

Los métodos de acceso tienen una sintaxis especial, en donde incluyen el nombre de la propiedad. Por tanto, `get.NombrePropiedad` será el getter y `set.NombrePropiedad`.

Los métodos de acceso deben estar bloques de métodos que no posean atributos. El usuario no podrá llamar a estos métodos, Matlab lo hará cada vez que el código intente acceder a las propiedades. Por lo tanto, los métodos de acceso a las propiedades no figuran en las listas de métodos y no están incluidos en los métodos de propiedades de los objetos de metaclasses.

- **Definición del Setter.**

La declaración del setter tiene la siguiente sintaxis, en donde `NombrePropiedad` es el nombre de la propiedad.

```
methods % No method attributes
    function obj = set.PropertyName(obj,value) % Value class
end
```

En el ejemplo `obj` es el objeto cuya propiedad se le está asignando un valor “`value`”. En las clases del tipo `value` los setter deben retornar un objeto con el nuevo valor ya asignado a la propiedad. Para el caso de las clases del tipo `handle` no es necesario retornar algún objeto.

```
methods % No method attributes
    function set.PropertyName(obj,value) % Handle class
end
```

El setter puede contener alguna estructura de control con el fin de chequear el valor a asignar, si verificar alguna condición.

Comportamiento del Setter.

Matlab llama al setter siempre que la propiedad y el método existan. Sin embargo no se llama al setter en los siguientes casos:

- Una llamada al setter dentro del método de seteo para evitar asignaciones recursivas.
- Asignación del valor por defecto de una propiedad.
- En la copia de un objeto del tipo `value`.

- **Definición del Getter.**

Matlab llama al getter siempre que se consulte el valor de la variable. La sintaxis es la misma que para el setter, en donde NombrePropiedad es el nombre de la propiedad. El getter siempre debe devolver un valor:

```
methods % No method attributes
    function value = get.PropertyName(obj)
end
```

Propiedades Dependientes.

Una de las aplicaciones de los getter es determinar el valor de una propiedad solo cuando esta se necesita. Para poder utilizar esta aplicación es necesario determinar a la propiedad como dependiente seteando su atributo como Dependent en true.

```
properties (Dependent = true)
   PropertyName
end
```

Ahora el getter de PropertyName determinara el valor de la propiedad y asignara este al objeto.

Acceso sin Getters y Setters.

La no existencia de métodos de acceso no imposibilita al usuario el acceso a las variables de instancia. Esto quiere decir que accediendo a las propiedades sin la definición de setter y getter puede visualizarse y modificarse los valores de las propiedades. De esta forma se accede a la estructura interna del objeto pero se viola el doble encapsulamiento, elemento fundamental del paradigma de la POO.

La forma de acceder a la estructura interna es con el siguiente esquema de sintáctico NombredelObjeto.NombrePropiedad, en donde NombredelObjeto es el nombre del objeto y NombrePropiedad es el nombre de la propiedad.

1.6. Métodos.

Los métodos son funciones que implementan las operaciones realizadas sobre los objetos de una clase. Los métodos junto con otros métodos de clase apoyan el concepto de encapsulación.

1.6.1. Tipos de Métodos.

Hay varios tipos de métodos especializados que llevan a cabo ciertas funciones o se comportan de manera particular:

- Métodos Ordinarios son funciones que actúan en uno o más objetos y retornan un nuevo objeto o algún valor calculado. Estos métodos son similares a las funciones ordinarias de MATLAB que no pueden modificar los parámetros de entrada. Este tipo de métodos permiten a la clase implementar operaciones aritméticas, o funciones de cómputo.

- Métodos Constructores son métodos especializados en la creación de objetos de una clase particular. Un método constructor debe llamarse del mismo modo que la clase y usualmente inicializa al objeto con los valores que se le pasan al método como argumento. Este tipo de métodos deben devolver el objeto instanciado.
- Métodos Destructores son llamados automáticamente cuando el objeto se destruye, por ejemplo si se llama a delete(obj) o si no existe referencia al objeto en cuestión.
- Métodos de acceso a las propiedades permiten a la clase ejecutar código específico siempre que se requiera modificar o visualizar el valor de una propiedad.
- Métodos de Clase son métodos asociados a las clases pero no necesariamente están asociados a los objetos instancia de esa clase. Este tipo de métodos no necesitan una instancia de la clase para la invocación del método.
- Métodos de Conversión estos métodos sobre cargan constructores de otras clases y permiten a la clase convertir sus propios objetos a la clase de constructor sobrecargado. Por ejemplo si se está trabajando con una clase determinada, y esta implementa un método de double entonces se llama un método de conversión y así en lugar de llamar al constructor de la clase double para convertir al objeto en cuestión en un objeto de la clase double.
- Métodos Abstractos sirven para definir una clase la cual no puede ser instanciada pero que si sirve para determinar un protocolo común para el conjunto de subclases.

1.6.2. Atributos de los Métodos.

Todos los métodos soportan el conjunto de atributos que se enumeran en la siguiente tabla. Estos sirven para modificar el comportamiento de los métodos. Por ejemplo, se podría prevenir el acceso a un método desde afuera de la clase o permitir que el método sea llamado sin necesidad de la existencia de una instancia de la clase.

Atributo	Clase	Descripción
Abstract	logical default=false	Si este atributo es true, significa que el método no tiene implementación. El método se escribe mediante una sintaxis que puede contener argumentos que las subclases utilizarán en la implementación del método: las subclases pueden no definir la

		misma cantidad de parámetros de entrada o/y salida. Sin embargo es frecuente que cuenten con la misma cantidad de parámetros. El método puede contar con comentarios después de la declaración. La declaración no llevan las palabras claves “function” y “end”.
Access	enumeration default=public	public: acceso ir-restringido. protected: acceso solo desde la clase o desde sus derivadas. private: acceso solo desde la clase.
Hidden	logical default=false	Cuando este atributo es false entonces el método se muestra en el listado de métodos. Caso contrario, no aparece.
Sealed	logical default=false	Si este parámetro se encuentra en true el método no puede ser redefinido en las subclases. Intentar redefinir el método producirá un mensaje de error.
Static	logical default=false	Setear este parámetro para definir métodos de clase, es decir, que no dependan de una instancia particular de la clase. Para referirse a métodos de clase la sintaxis debe ser: NombredelaClase.NombredelMetodo(Parametros).

1.6.3. Métodos Ordinarios.

Se pueden definir de dos formas:

- Dentro de el bloque de definición de clase.
- En una archivo separado en un @-directory.

Métodos dentro de un bloque de definición de clase.

El siguiente ejemplo muestra como se realiza una definición típica con los bloques clasdeff y methods:

Computadores II – Proyecto Final

```
classdef ClassName
    methods (AttributeName = value,...)
        function x = compute(obj,inc)
            x = obj.y + inc;
        end % compute method
    ...
    end % methods block
...
end % classdef
```

Una vez definido el método puede invocarse de dos formas:

- NombredelObjeto.NombredelMetodo(Parametros). Para el ejemplo anterior seria: obj.compute(inc) (notación de punto).
- NombredelMetodo(objeto,Parametros). Para el ejemplo seria compute(obj,inc) (notación de función).

Los atributos del método se aplican solamente para los métodos definidos dentro del bloque de métodos particular. Se pueden definir entonces varios bloques de métodos según atributos sean necesarios.

Métodos en Archivos Separados.

Se pueden definir métodos en archivos “.m” separados usando un @-directory. En este caso se debe crear la función en un archivo el cual debe poseer el mismo nombre que la función. Se debe declarar la asignación del método con un bloque de método dentro de un bloque de clase. A continuación se muestra un ejemplo con asignación de atributos.

```
classdef myClass
    methods (AttributeName = value,...)
        tdata = testdata(obj,arg1,arg2)
    ...
    end % methods
...
end % classdef
```

Mientras que el M-file simplemente se debe definir el método como una función. El ejemplo a continuación muestra la sintaxis de esto:

```
function tdata = testdata(myClass_object,argument2,argument3)
    ...
end
```

Limitaciones de los Métodos Definidos en @-directories.

- El método constructor no puede ser definido en un archivo separado.
- Los métodos de no pueden estar definidos en archivos separados.

Argumento Dominante.

El argumento dominante es aquel argumento que determinara a cual de todas las definiciones del método se llamará. La dominación se determina por la precedencia relativa de las clases de los argumentos. En general las clases definidas por el usuario tienen privilegios por las clases propias de Matlab.

Supongamos que ClassA especifica a ClassB como inferior y supongamos que ambas clases definen el método combine. Llamando el método con un objeto de la clase A classA y otro de la clase B classB: `combine(classA,classB)`. Entonces se llamará a la definición en la clase A por que esta clase es la de argumento dominante.

Notación Dot vs Notación de Función.

Si bien las llamadas son equivalentes en algunos casos particulares no resulta así y resultados pueden diferir de acuerdo a como MATLAB despacha los trabajos⁴.

Llamando Métodos de SuperClases Desde Las SubClases.

En ocasiones las subclases sobrescriben la implementación de métodos definidos en la superclases. Sin embargo a veces es necesario ejecutar el código declarado para el método en particular pero definido en la superclase. Para llevar esta tarea a cabo MATLAB tiene una sintaxis especial reservada para invocar métodos de la superclase para la implementación en la subclase del método con el mismo nombre en la superclase.

La sintaxis para llevar esta tarea a cabo es mediante el operador `@`. El esquema general sería el siguiente:

`NombredelMetodo@NombredelaSuperClase(Parametros).`

Restricciones en el Llamado a Métodos de La SuperClase.

En el llamado a métodos de la superclase se ven ciertas restricciones. Se puede realizar esta operación siempre que:

- El método que posee el mismo nombre que el método en la superclase.
- La clase debe subclase de la superclase a la cual se le está pidiendo el método.

1.6.4. Métodos Constructores.

Reglas De Los Métodos Constructores.

Un Constructor es un método especial que devuelve una instancia de la clase. Típicamente el Constructor acepta parámetros de entrada para asignar valores a las propiedades del objeto.

- El Constructor siempre lleva el nombre de la clase.

⁴ Este concepto incorpora conceptos de la metacategoría del lenguaje, se recomienda ver: MATLAB 7. Object Oriented Programming, “Dot Notation vs. Function Notation” pagina 11.9.

- El único argumento de salida es una instancia de la clase en cuestión.
- Si la clase que está siendo creada es subclase, MATLAB llama al constructor de cada superclase para inicializar el objeto. Las llamadas implícitas al constructor de la superclase se hacen sin argumentos. Si el constructor de la superclase requiere argumentos deben pedirse desde el constructor de la subclase explícitamente.
- Si el constructor realiza una llamada explícita al constructor de la superclase, esta llamada debe ocurrir antes de cualquier otra referencia al objeto construido.
- Una clase no necesita definir un constructor a menos que sea una subclase de una superclase cuyo constructor requiera argumentos. En este caso el usuario deberá explicitar la llamada al constructor de la superclase con los argumentos que se requiera.
- Si una clase no posee un constructor definido, MATLAB suministra un constructor que no tiene argumentos un objeto cuyas propiedades están vacías o los valores seteados como por defecto en las propiedades.
- Un constructor siempre debe retornar un objeto de su propia clase. El constructor de la superclase no puede retornar objetos de las subclases.
- Los constructores de las superclases no pueden ser ubicados dentro de estructuras de control (if, while, for, switch, etc).
- Se puede restringir el acceso a los constructores mediante atributos.

Inicializando un Objeto Con Su Constructor.

Como ya se mencionó los constructores deben devolver un solo objeto inicializado. El valor de salida es creado cuando el constructor se ejecuta, antes de ejecutar la primera línea de código.

Por ejemplo, constructor que se muestra a continuación puede asignar el valor a la propiedad del objeto “A” en la primera sentencia por que el objeto “obj” ya fue asignado a una instancia de la clase “myClass”.

```
function obj = myClass(a,b,c)
    obj.A = a;
    ...
end
```

Ahora pueden llamarse otros métodos de la clase por que el objeto ya fue inicializado.

El constructor también puede crear objetos cuyas propiedades tengan valores por defecto, vacías o con valores por defecto definidos en los bloques de propiedades. El siguiente ejemplo de constructor llama al método de clase CalculateValue para asignar el valor de la propiedad.

```
function obj = myClass(a,b,c)
    obj.Value = obj.CalculateValue(a,b);
    ...
end
```

Referenciando al Objeto en el Constructor.

Cuando se inicializa un objeto, por ejemplo, por asignación de valores a las propiedades, se debe usar el nombre del argumento de salida para referirse al objeto en construcción. Por ejemplo:

```
% obj is the object being constructed
function obj = myClass(arg)
    obj.property1 = arg*10;
    obj.method1;
    ...
end
```

Casos en los que no se pueden pasar Argumentos de Entrada al Constructor.

Cuando se cargan objetos al área de trabajo desde un MAT-file. Si el atributo de clase ConstructOnLoad esta activado la función de carga llamará al constructor el cual no puede pedir parámetros de entrada.

Construyendo Subclases.

El constructor de la subclase debe explicitar la llamada al constructor de la superclase si esta requiere argumentos de entrada. La subclase debe especificar estos argumentos en la llamada al constructor de la superclase usando el objeto de salida del constructor. A continuación vemos la sintaxis:

```
classdef MyClass < SuperClass

    function obj = MyClass(arg)
        obj = obj@SuperClass(ArgumentList)
        ...
    end
end
```

1.6.5. Métodos de Clase.

Este tipo de métodos son útiles cuando se desea crear una instancia del objeto después de haber ejecutado algún código.

Definición de Métodos de Clase.

Se muestra a continuación un ejemplo de definición:

```
classdef myClass
    ...
    methods(Static)
        function p = pi(tol)
            [n d] = rat(pi,tol);
            p = n/d;
        end
    end
end
```

Llamadas a Métodos de Clase.

La forma de invocar métodos de clase es la que se muestra a continuación:
NombreClase.MetododeClase(Parametros)

Herencia de Métodos de Clase.

Las subclases pueden redefinir los métodos de clase a menos que la super clase posea el atributo Sealed seteado.

1.7. Conceptos Generales de la PPO en Matlab.

1.7.1. Herencia Múltiple.

Cuando se definen dos o más superclases para una misma clase, la sub clase heredara las propiedades, métodos y eventos definidos por todas las superclases. Si una propiedad, un método o un evento es definido por más de una superclase no puede existir ambigüedades en la resolución. No se podrá derivar una clase de varias que posean incompatibilidades entre sus miembros. Por ejemplo, no se podría crear una subclase cuyas superclases son una del tipo handle y otra del tipo value.

1.7.2. Destructores.

La función por defecto para destruir un objeto es “delete”. Más aun, Matlab llama a esta función cuando desea o necesita destruir un objeto.

1.7.3. Paquetes.

Los paquetes son directorios especiales que pueden contener directorios de clases, funciones, y otros paquetes. Los paquetes pueden definir un ámbito de aplicación (a veces llamado un espacio de nombres) para el contenido del directorio del paquete. Esto significa que la función y los nombres de clase deben ser únicos dentro del paquete. El uso de paquetes proporciona un medio para organizar las clases y funciones y para seleccionar los nombres de estos componentes que pueden ser reutilizados en otros paquetes.

Directorios del paquete siempre comienzan con el carácter +. Por ejemplo:

```
+ mypack  
+ mypack / pkfcn.m; una función de paquete  
mypack + / @ class MiClase; clase en un paquete.
```

El directorio maestro del paquete siempre debe ubicarse dentro del path de MATLAB.

1.7.4. Variables De Clase.

No están soportadas en Matlab. Como alternativa se proponen las variables del tipo “persistent”. Por ejemplo, persistent X Y Z define X, Y y Z como variables que son locales a la función en la que se declaran sin embargo, sus valores se conservan en la memoria entre las llamadas a la función. Este tipo de variables son similares a las variables globales porque Matlab crea un almacenamiento permanente para ambas. Se diferencian de las variables globales en que las variables persistentes son conocidas sólo por la función en la que han sido declarados. Esto evita que las variables persistentes sean cambiadas por otras funciones o desde la línea de comandos de Matlab.

Ejemplo de esto podría ser la implementación de un patrón “Singleton”. Para esto definimos el método constructor como privado para que solo pueda llamarse desde dentro de la definición de la clase. Luego implementamos un método de clase “Instanciación” en donde definimos una variable persistente “UnicaInstancia”. Luego para instanciar la clase llamamos al método de clase creado y desde ahí, en la primera instancia llamamos al constructor y luego guardamos en “UnicaInstancia” lo que devuelve el constructor lo cual también devolvemos como parámetro de salida del método de clase. En las sucesivas llamadas al método de clase observamos que la variable persistente este vacía, si esta llena devolvemos su contenido.

1.7.5. Clases Abstractas.

Las clases abstractas son útiles para describir la funcionalidad que es común a un grupo de clases, pero requieren implementaciones únicas dentro de cada clase. Este enfoque se denomina a menudo una interfaz porque la clase abstracta define la interfaz de cada subclase, sin especificar la aplicación real.

Para definir una clase abstracta, es preciso setear el atributo Abstract para uno o más métodos de una clase. No se utiliza la estructura de “function” ... “end”.

```
classdef group
    % Both methods must be implemented so that
    % the operations are commutative
    methods (Abstract)
        rsum = add(numeric, polynom)
        rprod = times(numeric, polynom)
    end
end
```

1.7.6. Garbage Collection.

Tiempo de Vida de Un Objeto.

Matlab invoca en runtime el método delete solo cuando el tiempo de vida del objeto a concluido. Este tiempo de vida queda determinado cuando:

- Ya no se hace referencia al objeto en ningún lugar.
- Cuando se hace un borrado explícito.

Dentro de una función.

El tiempo de vida de un objeto que es local a la función, sea por variable local o parámetro de función, va desde que se asigna hasta que se reasigna, o no hay referencia hacia el objeto.