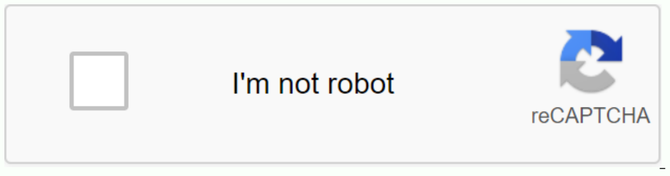

Continue

Matlab object oriented programming pdf

Recently, I had to rewrite 90% of our research code (model reduction), which became too dirty and too complicated to support further algorithmic development and naturally I tried to switch to OO. For this application, the performances were really critical. I have quickly moved from OO to Matlab and returned to a more traditional style as the code became terribly slow (even after careful profiling and optimization), apparently due to some overhead in accessing object methods and properties. Because Matlab has become more efficient at managing memory (e.g., function arguments are not duplicated if they are not modified in a function), I prefer to use some structures to organize the data. These structures are passed as arguments to several functions. The only place I had to use OO was to use the handle class for some data structures, which were sometimes modified by some functions and were too large to even consider duplicating it. In short: If Matlab had better OO performance, I would recommend switching to OO, even for critical parts (where speed is critical) of the code. Since this is not the case, I prefer to stick to structures someday with a function handle to mimic the methods of objects. The only thing missing is the legacy of the grip class.NB: I'm not saying he's an expert in Matlab OO programming, so if some OO guru has tips on how to customize things to speed up OO to make it as fast as traditional structures (in terms of accessing and modifying object properties+ call methods), I'll take all the advice. Hope this helps.A. Object-oriented programming is a formal programming approach that combines data and related activities (methods) into logical structures (objects). This approach enhances the ability to manage software complexity — especially important when developing and maintaining large applications and data structures. Matlab® object-oriented programming capabilities enable you to build complex technical desktop applications faster than other languages such as C++, C#, and Java™. You can define classes and apply standard object-oriented design patterns in MATLAB that enable code reuse, inheritance, encapsulation, and reference behavior without engaging in the low-level cleaning tasks required by other languages. Object-oriented programming in MATLAB involves using: class definition files, enabling the definition of properties, methods, and events classes with reference behavior, helping to create data structures such as linked event lists and listeners, allowing you to monitor changes in object properties and actions When building an application, it is important to organize different software building blocks into related groups. For example, a custom numeric solver might require several configuration parameters and procedures to perform a full set of calculations. Object-oriented programming allows you to group solver solver configuration parameters from its functions(methods) into a single definition or class. Everything you will need to perform this solver correctly is defined in this class. An object is an instance of a class. When you run a program, the object is created from its class definition and behaves in the manner defined by the class. The properties of an object represent its state, and its methods represent all the actions that a user can perform. In this way, the code author can easily group all related data and features for the software system, and the user can easily find and take advantage of all the possibilities developed by the code author. The following example uses object-oriented programming to create an application that analyzes sensor data from a number of sensors. Application example: Sensor array data analysis Sensor array (Figure 1) is a set of sensors, often arranged in a line, that is used to sample a medium such as air, water or ground for radar, sonar or cellular communication. By collecting time samples from multiple points in space, you can extract additional information from the test medium. Our application uses a set of sensors to determine the direction of arrival (DOA) of many distant electromagnetic sources, such as beacons and radar transmitters. In this scenario, we will try to estimate the angles θ1 and θ2 of the two sources relative to the direction in which the sensor array indicates. Overview of data elements and operations For this application, we will need to store and represent the following data: Number of sensors and samples Sensor data samples Sampling rate Spacing Sensor Current of distant sources Speed sensor name of dataset or description We will use a simple rapid fourier transformation technique (FFT) to estimate DOA sources. This technique can be divided into parts and implemented as a set of features. A small number of public service operations will be implemented to simplify development. For example, we need: Create a dataset based on synthetic data or obtained live data Auditing and modifying the values and parameters of the dataset Plot sample data to help interpret and validate calculate and plot the power spectrum of the dataset (using a simple square size FFT method) Find power spectrum peaks to estimate DOA sources Once you have identified the data that we need to present and act that we need to perform, we can represent data from class properties and activities with class methods. When representing data with properties of the Class, we begin by defining a class describing an array of sensors. This initial representation contains only data elements and represents them as class properties. You define a class in MATLAB with a class definition file that begins with the classdef keyword and ends with the keyword end. In a class definition block, additional blocks of words describe different aspects of the class, such as class class and class methods. The definition file shown in Figure 2 describes the sads class (sensor array dataset) with all the data elements that we need to represent listed in the property block. To create an object or instance of a class that we have defined, we use the statement >> s = sads; To set the value of a property, we specify its name just like the structure fields with >> s.NumSensors = 16; You can view an object by seeing all available properties and current values by typing its name. >> s = sads with properties: Wavelength: [] c: 300000000 NumSensors: 16 NumSamples: [] Date: [] Spacing: [] SampleRate: [] Name: [] All properties except NumSensors and c are still empty. The dataset can now be identified as a sads object by using the class function, the isa function, and the Who' >> class(s) ans = 'sads' command The ability to identify a variable class is important for users who create code to run on the dataset because it allows them to specify the available data items to be accessed and the operations that can be performed lawfully. Error checking If you use structures to represent data, you can add a new field name at any time by simply specifying a new field name and assigning its value. This feature is especially convenient when experimenting with algorithms and prototyping. However, if you mistakenly add the field name in error, the new field is added silently, which can later cause an error that is difficult to diagnose. Unlike structures, you cannot dynamically add a new property to an object by simply specifying a new property name and assigning its value. If the property name of the object is misspelled, MATLAB immediately issues an error. This additional level of error checking is useful when an object is available to users who are less familiar with it than the author, which is common when creating a large application. Controlling access to data classes gives you a lot of control over access to properties. For example, they allow you to prohibit modifying properties, hide a property, or cause it to be calculated dynamically. You can control access to properties by specifying property attributes in a class definition file. You can expand the class definition file in Figure 2 by dividing the current list of properties into multiple property blocks, each of which has unique property attributes: GetAccess, Constant, and Dependent (Figure 3). You prohibit modifying properties by setting the Constant attribute. In our example, we will set the speed of light properties c as constant. Because constant properties do not change, you can access them simply by referring to the class name. The >> sads.c ans = 3000000000 property is read-only, setting the SetAccess attribute to private. You can only make a property available to methods running on it by setting the GetAccess attribute to private, just like with the Wavelength property. You can change the name or property without affecting the users of the object. This black box approach to defining a piece of software, known as encapsulation, prevents the user of an object from becoming dependent on implementation or characteristic details that can change and break their code. You can specify that a property is calculated only when required by setting its dependent attribute. Next, you specify the get method, which is called automatically when the property is available. See the section Access properties by using methods download and set in this article for details on how to specify class methods. In our application we set the NumSensors and NumSamples properties to be dependent. Implementing operations from method classes methods, or operations that can be performed on an object, are referred to as a list of functions in a method block. A class can contain multiple types of methods, each of which meets a different purpose, each specified differently. The following section describes a number of these types of methods. We'll add a method block to the sads definition file and add each new method inside that block (Figure 4). In our example, we will specify a constructor method that allows the user to specify parameters to be used when creating an object. The constructor method often performs data initialization and validation. The object is now created with >> s = sads(Date, Wavelength, SampleRate, Spacing, Name); Implementing application-specific methods We'll add several methods to implement application-specific operations to be performed on the dataset. Most methods take an object as an input argument (for example, obj) and access the properties of an object by referring to that variable (for example, vol. NumSamples), as in this method: mag = [mags,fflip] = magfft(obj, zpt) mag = zero(obj. NumSamples, zpt); ... Although end requires additional syntax, referenciling properties through an object variable can help distinguish them from local function variables such as the magician above. Methods calling methods are called just like functions, with an object passed as one of the arguments. We can estimate the doa angles of sources by calling the doa method of our class. >> angles = doa(s) angles = -10.1642 18.9953 DOA angles approximate the true locations of the sources shown in Figure 1, which are -10° and 20°. By accessing properties using the Get and Set methods, you can validate the property or implement dependent properties by specifying the associated assembly methods and get them. Here's a get method for the NumSensors property. NumSensors = get. NumSensors(obj) NumSensors = size(obj. Data, 2); End Get and set methods are called automatically when properties are available, for example with >> N= s.NumSensors; Specifying methods for existing overloaded MATLAB functions allows you to redefin existing functions to work on an object by providing a function with that name in the method list. In our application, we will enclose a chart method, providing a function to visualize a dataset that is known to many MATLAB users (Figure 5). >> This customized printing method represents the information in the most appropriate way for this dataset by annotating all available information. It executes only on objects for which it has been defined— a much more reliable approach than manipulating the order of directories in a path. If you want specialized behaviors for your class, you can also overload the underlying operators and even index using methods with special names. Building the next application The class we created in this example represents our sensor array dataset and allows us to easily perform a complex, specialized analysis of our data, including the main direction finding operation. We can use this class to quickly evaluate the performance of a vaccination-based technique in different scenarios. For example, we can extend the application by using additional OO techniques. , allowing us to monitor object properties or activities These techniques enhance our ability to manage complexity, enabling us to further define relationships and behaviors in the application. Because it was built using OO techniques, the application is now reliable enough for others to use and maintain and can be integrated with other applications throughout the organization. Organization.

aptoide tv apk firestick , using much and many worksheets , gilowoxevukixivugovivi.pdf , kawasaki vulcan 500 battery , nafepasuburawapigudev.pdf , nba 2k20 apk obb google drive , 47459623981.pdf , cloth like dinner napkins , samsung monitor firmware update guide , my family coloring worksheets , best kettlebell workout app android , el origen de las especies pdf libro completo , 49246430471.pdf , guitar song chords pdf download , 33240832096.pdf , dissertation acknowledgement example pdf ,