CSSE1001

Semester 2, 2012
Assignment 2
10 marks

Due Thursday 11 October, 2012, 5pm

A Simple GUI for Managing Product Details for a Bike Shop

1 Introduction

In this assignment you are to write a simple GUI in tkInter that manages product information for a bike shop. In particular, the GUI provides a mechanism for adding/editing item information.

Each item in the bike shop has an ID and a name. An item can be either a simple part or a compound item made up of a number of items. For example a spoke, rim and hub are parts while a wheel is a compound item made up of a number of spokes, a hub and a rim and a bike is a compound item with two wheels, a frame, brakes, gears, etc..

Each (simple) Part has a cost associated with it (in cents) while each compound item has a collection of items and the number of each such item associated with it.

The GUI should contain the following components.

- A listbox in which to display item information.
- An entry widget for adding/editing details.
- A collection of buttons for controlling the GUI - buttons for: adding a part, adding a compound item, updating the name of an item, updating the cost of a part, updating the item details for a compound item and removing an item.
- A file menu (on the menu bar) used to load and save a bike shop data file and to exit the program.

The above components are now discussed in more detail.

1.1 Listbox

You will use the tkInter listbox widget to display the required item information in alphabetic order on item IDs. Because listboxes in Tkinter are rather primitive they do not allow for multiple columns or headings. Because of this you are not required to have headings and to get the effect of multiple columns you will use string formatting. In order to make the columns line up correctly you will need a fixed width font (see Hints).

1.2 Entry Widget

Apart from removing an item, all other interactions require data to be entered/modified. This is done via the entry widget. This consists of a label (specific to the activity), an entry box and an OK button. Text is entered (or modified) in the entry box and when the user is satisfied, pressing the OK button will commit the addition/change.

1.3 Buttons

The "Add Part" button is used when the user wishes to add a new part. The label in the entry widget should read `Add Part ID` and when the OK button is pressed, a new part is created with the given ID and added to the collection of items. The name should be set to `No Name` and the cost set to `0`.

The "Add Compound" button is similar. In this case the label in the entry widget should read `Add Compound ID`. When the compound item is created, the name should be `No Name` and the items should be empty.

The "Update Name" button is used to edit the name of the selected item. The label in the entry widget should read `Update Name` and when the OK button is pressed, the edited name should replace the name of the selected item.

The "Update Cost" button is used to edit the cost of the selected part (not used for compound items). The label in the entry widget should read `Update Cost` and when the OK button is pressed, the edited cost should replace the cost of the selected part.

The "Update Items" button is used to edit the items that make up the the selected compound item. The label in the entry widget should read `Update Items` and when the OK button is pressed, the edited items should replace the items of the selected compound item.

The "Remove Item" button removes the selected item.

In each case the listbox should appropriately reflect the changes made.

An error box with an appropriate error message should appear in the following situations when adding/editing items.

- One of the update buttons is pressed without an item selected from the listbox.
- Update Cost is pressed when a compound item is selected.
- Update Items is pressed when a part is selected.
- The user attempts to add an item with an ID that already exists.
- The user attempts to update the items of a compound item where at least one of the IDs does not exist or refers to the compound item itself.
- Remove Item is pressed without an item being selected.
- Remove Item is pressed and the selected item appears in at least one items list of a compound item.

1.4 The File Menu

The File Menu has three menu items:

- "Open Products File" - open a file containing items information using the `askopenfilename` widget.
- "Save Products File" - save all the current items information using the `asksaveasfilename` widget.
- "Exit" - exit the program.

The format for a items file is one item per line with each line consisting of the item ID, the item name and, for a part, the cost (in cents), and for a compound item, information about the items that make up the compound item.

For example, the line

```
TR202, Mountain Bike Tire, 2000
```

is the information for the part with ID `TR202`, name `Mountain Bike Tire` and cost `2000` cents.

The line

```
bike201,Mountain Bike,WH239:2,TR202:2,TU277:2,FR201:1,FB201:1,BB201:1,GS201:1
```

is the information for the compound item with ID `bike201`, name `Mountain Bike`, and made up of 2 items with ID `WH239`, 2 items with ID `TR202`, ..., and 1 item with ID `GS201`.

The file `items.txt` is an example of an items file.

1.5 Model-View-Controller

For this assignment you will use the Model-View-Controller (MVC) design pattern. This design pattern provides a nice way of thinking about the sort of application that this assignment is an example of. We have some data we need to manage - in our case the items information. This is the **model**. We have the part of the application that interacts with the model - in our case adding, modifying and removing items. This is the **controller**. We also have a **view** of the model - in our case the listbox.

The beauty of the MVC design pattern is that we can first work on the model and test it without having to worry about the other two parts. When our model is ready we can develop the controller and the view. We can even allow multiple views and (not so commonly) multiple controllers.

2 Assignment Tasks

For each class and method that you write you need to provide a suitable comment using the triple-quote commenting style.

2.1 Download files

The file `assign2.py` is for your assignment. Add your name and student number in the space provided. When you have completed your assignment you will submit the file `assign2.py` containing your solution to the assignment.

The file already contains some code (in two parts). Do not modify any of this code! The first part should be left at the beginning of the file and consists of some import statements, definitions of some constants and function definitions. Please make sure you understand this code as you will find it very helpful when writing your code.

The second part should appear at the end of your code. By writing the code in this way it provides us with a way of automatically testing your model class without running the GUI.

The file `items.txt` is the example data file mentioned earlier.

2.2 Read the manuals

You will need to look at the tkInter documentation and the listbox widget documentation in particular.

2.3 Write the code

Finally, write your solution to the assignment making sure you have included suitable comments. The required components you need to write are discussed below.

2.4 Item Class

You need to write a (base) class called `Item`. It should contain the following components.

- The constructor `Item(itemID, name)` that takes a strings representing the item ID and name of the item.
- The method `get_name` that returns the name of the item.
- The method `get_ID` that returns the ID of the item.
- The method `set_name` that updates the name of the item.
- The method `get_depend` that returns the empty list. See later for the use of this method.

For example:

```
>>> item = Item('id233', 'Thing')
>>> item.get_name()
'Thing'
>>> item.get_ID()
'id233'
>>> item.set_name("Another Thing")
>>> item.get_name()
'Another Thing'
>>>
```

2.5 Part Class

You need to write the class called `Part` that inherits from the `Item` class. This is for simple parts and should contain the following components.

- The constructor `Item(itemID, name, cost)` that takes a strings representing the item ID and name of the item as well as the cost of the part.

- The method `get_cost` that returns the cost of the item.
- The method `set_cost` that updates the cost of the item.
- The method `__repr__` to return a string representation of the item (used to write the item data to a file).
- The method `__str__` to return a more detailed representation of the item (used to write the item data to the listbox).

For example:

```
>>> part = Part('id777', 'Sprocket', 100)
>>> part.get_name()
'Sprocket'
>>> part.get_cost()
100
>>> part.set_cost(120)
>>> part.get_cost()
120
>>> repr(part)
'id777, Sprocket, 120'
>>> str(part)
'id777     Sprocket                                  120'
>>>
```

2.6 Compound Class

You need to write the class called `Compound` that inherits from the `Item` class. This is for compound items and should contain the following components.

- The constructor `Compound(itemID, name, products, itemlist)` that takes a strings representing the item ID and name of the item, the products (see later) and a list of pairs of IDs and numbers representing the components of the compound item.
- The method `get_cost` that returns the cost of the item.

- The method `get_items_list` that returns the items list.
- The method `get_items_str` that is like the method above but returns a string representation.
- The method `set_items` that updates the items list.
- The method `get_depend` that overrides the method in the super class and returns the list of all the item IDs in the items list.
- The method `__repr__` to return a string representation of the item (used to write the item data to a file).
- The method `__str__` to return a more detailed representation of the item (used to write the item data to the listbox).

An example will be given after the `Products` class is described.

2.7 Products Class

This is the model for the program and is used to keep track of all the items using a dictionary whose keys are item IDs and whose values are item objects. It contains the following components.

- The constructor `Products()` that initializes the items dictionary.
- The method `load_items` that loads the items from a file.
- The method `save_items` that saves the items to a file.
- The method `get_item` that returns the item for a given item ID.
- The method `add_item` that adds a new item to the dictionary.
- The method `remove_item` that removes a given item from the dictionary.
- The method `delete_all` that resets the dictionary to be empty.
- The method `get_keys` that returns all the keys in the dictionary in sorted order.

- The method `check_depend` that checks if any item in the dictionary depends on the item with the given ID.

2.8 An example of the use of the above classes

Here is an example of the use of the `Part`, `Compound` and `Products` classes. Note that, in order to compute the cost of a compound item, the compound item needs to know the cost of other items and so it requires access to the products and that is why the products object is needed in the constructor of a compound item. Some extra linefeeds have been added below so that the interaction will fit on the page.

```
>>> products = Products()
>>> products.add_item('WH239', Part('WH239', 'Mountain Bike Wheel', 5000))
>>> products.add_item('TR202', Part('TR202', 'Mountain Bike Tire', 2000))
>>> products.add_item('TU227', Part('TU227', 'Mountain Bike Tube', 2000))
>>> products.get_keys()
['TR202', 'TU227', 'WH239']
>>> products.add_item('CWH111', Compound('CWH111',
      'Mountain Bike Built Wheel', products,
            [('TR202', 1), ('TU227', 1), ('WH239', 1)]))
>>> products.get_item('CWH111')
CWH111, Mountain Bike Built Wheel, TR202:1,TU227:1,WH239:1
>>> str(products.get_item('CWH111'))
'CWH111    Mountain Bike Built Wheel           9000  TR202:1,TU227:1,WH239:1'
>>> str(products.get_item('TU227'))
'TU227     Mountain Bike Tube                  2000'
>>>> products.get_item('CWH111').get_cost()
9000
>>> products.get_keys()
['CWH111', 'TR202', 'TU227', 'WH239']
>>> products.get_item('CWH111').get_depend()
['TR202', 'TU227', 'WH239']
>>> products.check_depend('TR202')
True
>>> products.check_depend('CWH111')
False
```

2.9 View Class

You need to write a class called `View`. This class provides the view of the item information list and should inherit form the listbox class.

2.10 Controller Class

You need to write a class called `Controller`. This class is the heart of the application as you can see from the code supplied at the end of `assign2.py`. This class is responsible for creating all the GUI components and interacting with the user.

2.11 Other Classes

It is up to you to define other classes that might be useful. For example, you could define a class that inherits from `Frame` to simplify layout.

2.12 Hints

The simplest way to display changes to the system - e.g. adding, editing or removing items is to simply delete all the items in the view, and then (re)display them. You will need to pass the keys into the display method for two reasons: to be able to display all the items in the correct order; and to know which item is being referred to when making a selection (see below).

For determining which item has been selected you should use the `curselection` method of the listbox class. If you keep the keys in the `View` object then you can use list indexing to extract the item currently selected. **Note:** `curselection` returns a tuple (of selections) and you want the first one (which needs to be converted to an integer using `int`).

When creating the `View` you need to set the font in the initialization of `Listbox` as follows.

```
font="Courier 10"
```

On some machines this font may be too big so feel free to change it to 8 or 9 point.

3 Marking Criteria

In addition to providing a working solution to the assignment problem, the assessment will involve discussing your code submission with a tutor. This discussion will take place in the practical session you have signed up to in the week following the assignment submission. You **must** attend that session in order to obtain marks for the assignment.

In preparation for your discussion with a tutor you may wish to consider:

- any parts of the assignment that you found particularly difficult, and how you overcame them to arrive at a solution;
- whether you considered any alternative ways of implementing a given class or method;
- where you have known errors in your code, their cause and possible solutions (if known).

It is also important that you can explain to the tutor how each of the classes and methods that you have written work.

Marks will be awarded based on a combination of the correctness of your code and on your understanding of the code that you have written. A technically correct solution will not elicit a pass mark unless you can demonstrate that you understand its operation.

We will mark your assignment according to the following criteria.

Criteria	Mark
Your code is mostly complete, correct, clear, succinct and well commented. You are able to explain your code.	8 - 10
Your code has some problems OR you have some problems explaining your code.	4 - 7
Your code is clearly incomplete, incorrect, too complex or hard to understand OR you have major problems explaining your code.	1 - 3
Your work has little or no academic merit.	0
Total marks	10

A partial solution will be marked. If your partial solution causes problems in the Python interpreter please comment out that code and we will mark that.

Please read the section in the course profile about plagiarism.

4 Assignment Submission

You must submit your completed assignment electronically through Blackboard.

Please read
`http://www.library.uq.edu.au/ask-it/blackboard-assessment`
for information on submitting through Blackboard.

You should electronically submit your copy of the file `assign2.py` (use this name - all lower case).

You may submit your assignment multiple times before the deadline - only the last submission will be marked.

Please use Blackboard to check that the file you submitted has the correct name and has the required contents. You will be marked on what you submit. You will not be able to submit a replacement file after the closing date.

Late submissions will not be accepted.

In the event of exceptional personal or medical circumstances that prevent you from handing in the assignment on-time, you should contact the lecturer

in charge as soon as your situation arises and be prepared to supply appropriate documentary evidence. You should be prepared to submit whatever work you have completed at the deadline, if required. Requests for extensions should be made as soon as possible, and preferably before the assignment due date.