Coursework 1: Object Orientation

The coursework revolves around the implementation of a stock control system for a small shop. This system will have a list of stock items, their quantity in stock and methods for adding new stock, checking stock levels and selling stock. A half-finished version is provided, in StockControl.py. The aim of the coursework is to implement the unfinished methods, add new methods and extend the class to cater for perishable stock items. (Hint: The 'pass' commands are just for if a method is empty. Once you have coded a method you can delete 'pass').

To complete the coursework work through the following steps:
0. Type your name and student number into the spaces at the top of the code
1. Implement StockItem's toString method (Returns a string describing the stock item, its barcode and the quantity remaining)
2. Implement StockItem's needRestock method (Returns true if this item needs restocking (i.e. the quantity<a threshold)
3. Implement StockControl's listRestock method (Return a string listing the items that need restocking). Hint: use the '\n' character to add a newline after each item.
4. Make sure the listRestock method returns a message "All items stocked" if all items are stocked .
5. Implement StockItem's sell method (Process the sale of an item, raises an exception if an item is sold when its stock is zero).
6. Implement StockControl's addStockType method (Add an item to the stock list)
7. Implement StockControl's sellStock method (Process the sale of one item). This takes the barcode of the item sold, so we need to search for this item in the list of items. If it's not there we need to raise an exception.
8. Add a new method to StockControl which allows restocking, call it "restock" and let it take two parameters: The barcode of the item being restocked and the quantity. i.e. its definition will look like:

```
restock(self, barcode, quantity)
```

9. add a new method to the StockItem class to allow restocking , this would take just the quantity of new stock (and add this to the current quantity of stock). Call this method from the StockControl restock method. Again raise an exception if the product is not found.

Test it! (this is for your benefit!)

10. Extend the StockItem class with a new class: PerishableStockItem
This will be for items that can go off quickly (like milk)
for this type of product we'll assume that all of the current stock is of the same date, so we'll have a new instance variable sellbydate
11. For this class make a constructor with the last parameter the sellbydate:
def __init__(self, name, barcode, quantity, sellbydate):
Reminder you can call the parent constructor with the super method, e.g.
super(PerishableStockItem,self).__init__(name, barcode, quantity)

For testing there is a commented out line for milk you can use

12. Add a new method to this class: pastSellByDate
which checks if we're past the sellbydate (returns true if we are)
hint: the date type can be used in the following way. To test if today is AFTER the sellbydate, simply compare:
date.today()>self.sellbydate
(where date has been imported from the datetime module)
13. Override the needRestock method of the PerishableStockItem method to also check if it's past the sellbydate, and return true if either it's out of stock or out of date.
14. Finally override the toString method of the PerishableStockItem class to display the sell by date .
Hint 1: why not get the original string using the super method:

```
message = super(PerishableStockItem,self).toString()
```

hint 2: you can use the str() method to get the date as a string:

```
message += " " + str(self.sellbydate)
```

Submit your code by email to **msmith@cit.ac.ug**, with the subject line:
[Advanced Programming: Coursework 1]

The deadline for this coursework is: **Midnight on Friday 10th October**