You will receive points for this problem after the exam is over. Each solved subtask awards some of the points, even if you do not complete all subtasks.

In this problem, you will write the classes for modelling an inventory management system and cash register for a small grocery store. You are allowed, and in some cases required, to add fields and helper methods beyond what is mentioned by each subtask. Each completed subtask below will award points, so you do not need to solve all tasks. Each task corresponds to a test, and you may submit your solution as many times as you like without point deductions.

All fields must be private and all methods and constructors must be public. The return type of methods is void, if no return value is mentioned.

Subtasks:

1. Implement the **SaleItem** class to represent an item in stock at the store. A **SaleItem** has a name (a text string), a category (another String), a **priceInCents** (a whole number), and a stock (another whole number) indicating how many items are currently for sale in the store.
2. Implement the constructor to SaleItem such that it initializes the fields, with one parameter for each field in the order mentioned above.
3. Add a method **setPrice(price)** that updates the field price to the value given as parameter.
4. Implement the method **checkStock(count)** in SaleItem such that it returns true if there is at least count items left in stock, and false otherwise.
5. Implement the method **addToStock(extra)** that increases the stock by the extra amount.

The ShoppingBasket class should keep track of the items as single customer would like to buy. You are free to choose what fields the class has.

6. Add a method **addItem(item)** that takes a SaleItem and adds a single one of it to the shopping basket. If the customer wants to buy more than one, the same SaleItem object will be added multiple times. Then add a method **totalCost()** that returns the sum of all items in the shopping basket.
7. Implement the method **checkStock()** in ShoppingBasket such that it returns true or false depending on whether there are enough items in stock to purchase all the items in the shopping basket. The method must not change the current stock of any item.

Two exceptions, InvalidPurchase and **OutOfStock** have been defined in the code template.

8. Add a method **finalizePurchase()** that locks the order, and removes the items from stock (i.e., have their stock count reduced). If any item has too few items in stock, an **OutOfStock** exception must be thrown (you choose the message), and no items must be removed.
9. Modify **finalizePurchase** so that if it is called multiple times, only the first call must do something; you can only finalize once.
10. Modify **addItem(item)** so that if the purchase has already been finalized, it throws an **InvalidPurchase** exception (you choose the message) and does not add the item.

11. Create a method **printReceipt()** that calls **finalizePurchase** (propagating any exception) and prints a list of all the items in the basket, sorted primarily by category, secondly by name. Each line must contain the name of the item followed by the price in dollars. Finally, a total must be printed. Example:

```
Bread $1.29
Cheese $4.00
Milk $0.99
Milk $0.99
---
Total: $7.27
```