Exercise 7: Extended use of the `DrinksMachine` class

1) Make sure you are up to date with the notes on the course web pages for "ODL127 Algorithms" up to section 17 of the course notes.

Other sets of notes you may find useful include:

`http://www.ibiblio.org/javafaq/course/` (up to chapter 4)

`http://java.sun.com/docs/books/tutorial/java/index.html`

`http://www.dcs.qmul.ac.uk/~mmh/TIJ3/TIJ3.htm` (up to chapter 10)

`http://sepwww.stanford.edu/sep/josman/oop/oop1.htm`

`http://www.cs101.org/ipij/` (up to chapter 14)

2) Look at the directory available through the web:

`http://www.dcs.qmul.ac.uk/~mmh/DCS128/notes/code/objects/`

You will find the following files there: `Can.java`, `EmptyCanException.java` `DrinksMachine.java`, `ExtDrinksMachine1.java`, `ExtDrinksMachine2.java`, `DrinksCompany.java`, `UseDrinksMachines5.java` and `UseDrinksMachines6.java`. Download these files, compile them, and make sure you can run the programs whose `main` methods are in `UseDrinksMachines5.java` and `UseDrinksMachines6.java`.

3) Write a class `CanAndCash`. This will define a single object which stores a `Can` object and an `int` value representing a sum of money. Use this as the return type for a static method `buyCoke` which takes a `DrinksMachine` object and an integer representing a sum of money, and returns a `CanAndCash` object representing the result of putting the money into the machine, pressing the "Coke" button and then pressing the "change" button.

4) Write a static method `buySpriteOrFanta`. This method should take as its argument a `DrinksMachine` object and an integer representing a sum of money, and return a `CanAndCash` object as mentioned in question 3). If the `DrinksMachine` object is actually an `ExtDrinksMachine1` object, the method returns the result of inserting the money and pressing the "Sprite" and "change" button. Otherwise it returns the result of inserting the money and pressing the "Fanta" and "change" button. Make sure you know how to use the `instanceof` operator and type casting, which you will need for this question.

5) Write a static method `cheapest` which takes an arrayList of `DrinksMachine` objects, and returns the one which is cheapest.

Then write a static method `cheapestCoke` which takes an arrayList of `DrinksMachine` objects, and returns the one which is cheapest but ignoring any which have run out of Coke cans (as given by the `cokesEmpty` method).

6) Write a class `CanBuyer`. A `CanBuyer` represents a robot who you send off to buy cans of drink for you. It will have an arrayList representing the drinks machines in the locality. It has a command implemented by the non-static method `buyCoke` to take some money from you and return with the change and a can of Coke (so return type `CanAndCash`), and a command `buyFanta` to do the same and return with the change and a can of Fanta. It will buy the can from whichever machine in the locality is the cheapest one which has the type of drink you want in stock. But it will also charge you a commission to do the job (the commission rate might be set when the `CanBuyer` object is created).

7) Write an extended version of `CanBuyer` which represents robots operated by a company represented by an object of type `RobotCompany`. The commission charged by the robot is passed on to the company. The `RobotCompany` object should have a method which returns all the commissions collected by the robots it controls.

8) Read the paper "Why a Duck?" by John Brewer which you can find on the link `http://www.jera.com/techinfo/duck.html`. Consider how the ideas in this paper could be applied to the robots and drinks machines examples covered here.

For example, write an `interface` with two commands, `buyCoke` and `buyFanta`. Your class `CanBuyer` should implement this interface. But you could also write an Adapter, to enable `DrinksMachine` objects to implement the interface. You could write a Factory Method which produces `CanBuyer` robots in the `RobotCompany` class. You could write a Decorator which keeps count of the number of cans a robot or machine has sold. You could write a Composite where a robot boss keeps a whole array of robots.