

ECS510 Algorithms and Data Structures in an Object-Oriented Framework

Exercise Sheet 7: Extended use of the **DrinksMachine** class

This is a set of exercises to support the “Inheritance” section.

- 1) Look at the code folder for this section. You will find the following files there:
Can.java, EmptyCanException.java DrinksMachine.java,
ExtDrinksMachine1.java, ExtDrinksMachine2.java, DrinksCompany.java,
DrinksMachineOps.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.
- 2) Write an extended version of class Can called PromotionCan. This represents cans produced in a promotion where certain cans are prize-winning cans. If you have a prize-winning can and you contact the company to inform them, they will send you a prize. All cans produced as part of this promotion have a special design on the outside, so you can tell by looking at it whether a can is a promotion can. But you only find out if a promotion can is also a prize-winning one by drinking from it and seeing if it has a special message printed inside it. Your class PromotionCan should add an extra method called isWinner to the class Can which tells you whether a particular PromotionCan object represents a prize-winning can. As it is necessary to drink the contents of a promotion can to see if it is a prize-winning can, calling isWinner on a PromotionCan object which does not represent an empty can should throw an exception.
- 3) Write a class CanFactory that describes an object which has a method makeCan that can be called to return a new PromotionCan object. The CanFactory class should be set up so that a proportion of cans that are produced are prize-winning cans, the proportion should be an argument to the CanFactory constructor. You could use Java's built-in class Random to make a random decision on whether a new PromotionCan object it returns is a prize winner.
- 4) Modify your answer to question 3) to give CanFactory a method which represents a switch – when you turn it one way, makeCan returns ordinary Can objects, when you turn it the other way, makeCan returns PromotionCan objects.
- 5) Write an extended version of the class DrinksMachine which differs from a normal one by using a CanFactory object as defined in question 4) to produce the Can objects added to the machine when it is created. There should also be variants of the methods loadCoke and loadFanta which cause a Can produced from a particular CanFactory to be loaded when the methods are called instead of the Can being an argument to the method. Write front-end code which uses this class to simulate buying a drink from a machine, seeing if it is a can from the promotion, and if it is, checking if a prize-winning can has been bought. The factory used to produce Can objects should be given by a field in the extended DrinksMachine class which is initialised in the constructor.

- 6) 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.
- 7) 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 6). If the `DrinksMachine` object is actually an `ExtDrinksMachine1` object, the method returns the result of inserting the money and pressing the “Sprite” and “change” buttons. Otherwise it returns the result of inserting the money and pressing the “Fanta” and “change” buttons.
- 8) 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. The robot has a command implemented by the instance 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).
- 9) Write an extended version of `CanBuyer` which has an additional method `buySprite` which will return with the change and a can of Sprite from the cheapest machine in the locality which has Sprite in stock.
- 10) 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 that can be called to return a new robot and a method which returns all the commissions collected by the robots it controls.