ASSIGNMENT 3 REPORT

Álvaro Castillo García

Pablo Ernesto Soëtard

Andrés Martín Manos

Universidad Autónoma de Madrid

Escuela Politécnica Superior

Ingeniería Informática bilingüe, grupo 2292

Software Analysis and Design

PART 1-5

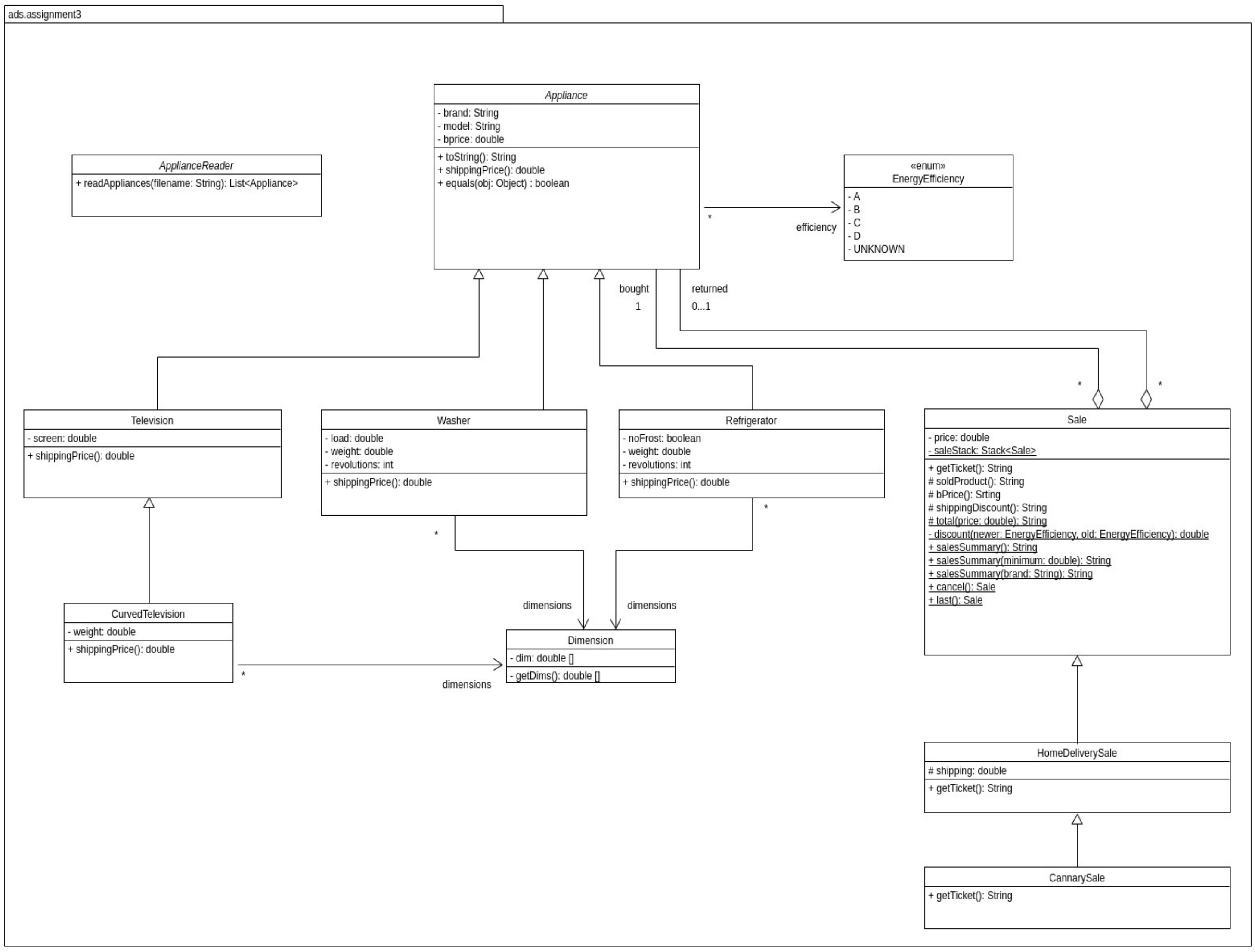
For the first part, we had to implement a lot of classes. We had to decide if the attributes were protected or private and if the functions were public/ private/ protected. At first, we implemented getTicktet() as a function that created and retuned the whole required string, but then when we read part 4 we had to change it.

In the second part we were tasked to develop the class *ApplianceReader* where whe had to add a method which read line by line the text file containing all the data. It was challenging as we hadn't done file reading in java before but we managed to succeed.

We thought the third part would be tricky, but it ended up being much simpler than we thought, as we had to modify part two in order to avoid duplicates when reading from the file. We implemented it based on the thought that if the read data is already in our *result* list, then it wouldn't be added to said list.

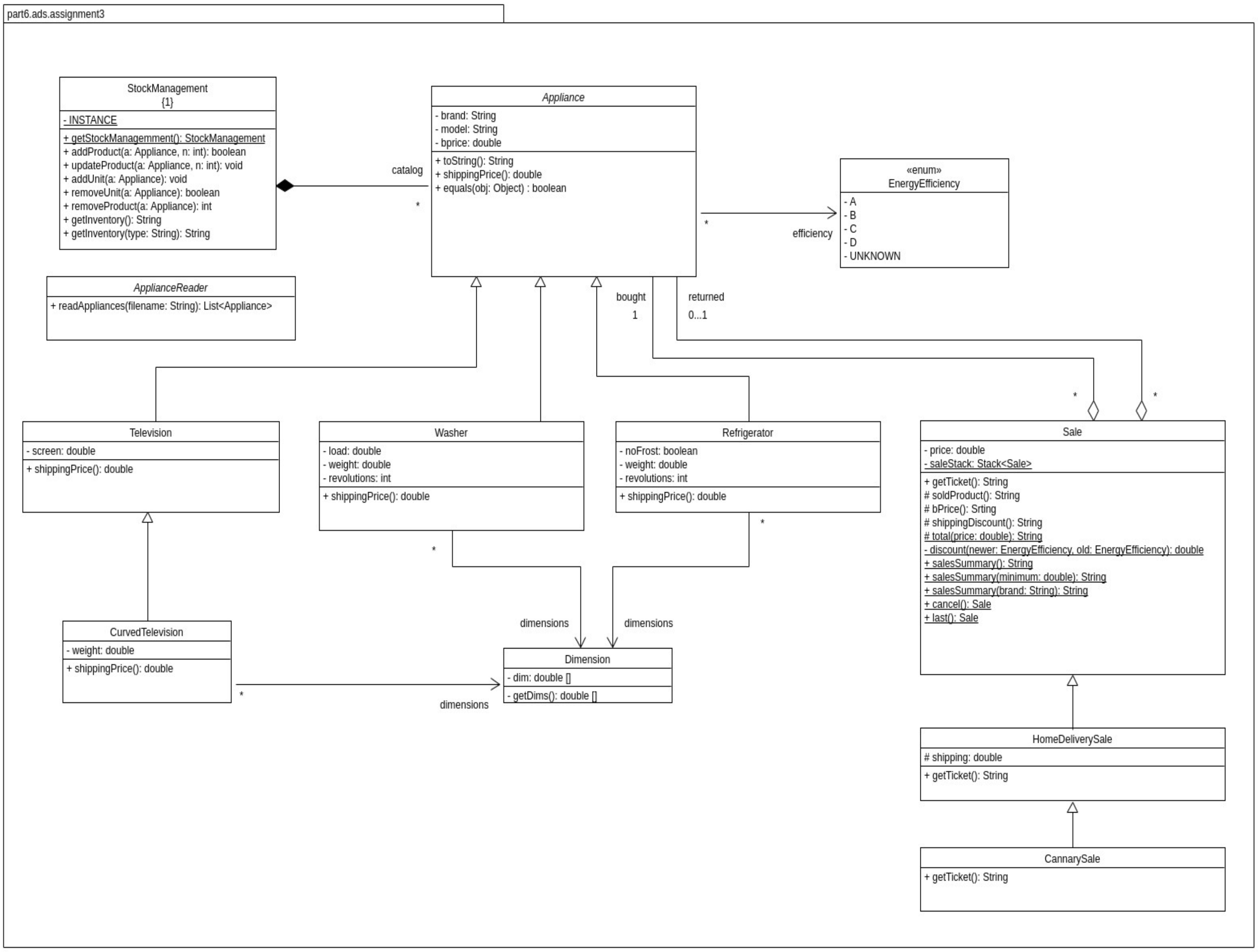
In part 4, we created a satic stack, to keep all the sales that were created. As it was static, we had to change getTicket() and split the code into different functions declarating them as static too. This was a bit messy, but not difficult.

The 5th part was very easy, as we just had to create two more classes (CannarySale and CurvedTelevision) that inherited from HomeDeliverySale and Television respectively. In addition, we had to overload the constructor of HomeDeliverySale by creating another one just for the Cannary Sales.



This class diagram corresponds to that of part 5. We can observe that there is an abstract class called *Appliance* which is the one that will give the basic parameters to other classes such as the *Television, Washer* and *Refrigerator,* and which will be read in the forward code by the class *ApplianceReader*. Adding to this we can also see there is an *Energy Efficiency* enumeration, with the values it can take. Meanwhile, the *Washer*, *Refrigerator* and *CurvedTelevison* classes, this last one having its screen size defined in the *Television* class, all have a dimension parameter, which is implemented in the *Dimension* class and which is comprised of an array of three different dimension values. Finally, we have the *Sale* class, which is the biggest one yet, and is the one which has all the methods to interact with the prices of the other appliances by creating and deleting sales. *HomeDeliverySale*, that inherits all the attributes and functions form the *Sale* class, is used when a home delivery is done. It will calculate the delivery fee which will be imposed depending on the sizes of the appliance. The *CannarySale*, which calculates the delivery fee for Cannary Islands-bound packages, is a subclass of HomeDeliverySale.

PART 6



This class diagram represents the code used in part 6.
In addition to everything else in the previous diagram, we can observe that there is a new class. The StockManagement class is declared as a singleton and that is why we have an attribute INSTANCE. Furthermore the function getStockManagement() will return the INSTANCE if it exists. Otherwise it will call the private constructor and it will return the new object.
This class includes a Map of appliances and its number of units. It includes also all functions for deleting, adding and getting information from the map.

For this part we have also modified the files Appliance.java and Sale.java in order to make the stock work. The constructor of Appliance.java will call addProduct and the functions for removing and creating a sale will call addUnit and removeUnit() respectively.
Finally we have programmed a little Tester for this part, that creates some appliances, add some units to the stock, prints the stock and remove some items and units.

We had decided to create just one map, that contains all the appliances, but we could have modeled it as three maps, one for each type (Washer, Refrigerator, Television). In addition, we have decided that the number of units of a product should be added from the main program.

There is one thing left to be done. When we create a sale (in the constructor), we first check that there are appliances in the stock, but if there are none, we do not know yet how to not create the sale. So we print a string saying: *Sale to be done when we recieve appliances in the stock.*