4 Animal Keeper

4.1 General information

Your submission for all the parts of the exercise should be in Java. All the advanced exercises must be done in pairs.

4.2 Assignment description

You are thinking of accepting an invitation to become the new Animal Keeper of the local Zoo. However, as the Animal Keeper is supposed to handle many things at the same time, the idea of doing this without proper planning is daunting to you. Therefore, you decide that you will accept the job if you are able to write a program first that can help in keeping track of the current animal situation in the Zoo.

The Zoo can keep a number of different animals. Here is a numbered list:

1. Lion;
2. Tiger;
3. Leopard;
4. Zebra;
5. Antelope;
6. Giraffe;
7. Bear.

You need to be able to accommodate all these animals. For this, the Zoo has a fixed number of *open enclosures* and *cages*. Zebras, Antelopes and Giraffes can only live in open enclosures. The other animals can live in both.

Of course, herbivores (animals 4–6) cannot be kept together with carnivores (animals 1–3) and omnivores (animal 7). In addition, carnivores and omnivores do not like to live with other carnivore and omnivore species.

Tigers, Leopards and Bears are solitary creatures that do not want to live in a group. All other species can live in groups.

Each type of home has a limited capacity: in a cage, you can keep up to two animals, while in an open enclosure, you can keep six animals.

The animals must be fed. There is:

1. hay;
2. corn;
3. grain;
4. carrots;
5. chicken;
6. beef.

The first four types of food are for herbivores, but carrots cannot be given to Antelopes, for dietary reasons. For the carnivores, there is chicken and beef. Bears, being omnivores, can eat beef, chicken and carrots. You need to buy food before you can use it.

Finally, as at times, animals are sold to another Zoo or die, the system must be able to remove animals, and it should be possible to move animals within the Zoo, from one home to another.

4.3 Exercise

Design and write a program that helps you to manage the Zoo. The program should be object-oriented, and robust against all possible violations of the Zoo restrictions, and report whenever a violation occurs. In case a given command leads to a violation, the command should not be executed at all. No violation should lead to the program terminating.

4.4 Input

As input, the program should accept a (space-separated) sequence of commands, each in one of the following formats:

1. **0 t "name" h**: add (command '0') an animal of type **t** (see the list of animals given before) with name **"name"** to the home with number **h** (see the restrictions given later for the home IDs);
2. **1 "name" h**: move the animal with name **"name"** from its current home to the home with number **h**;
3. **2 "name"**: remove the animal with name **"name"** from the Zoo;
4. **3 f x**: buy of food type **f** (see the list of food given above) the amount **x**;
5. **4 f x h**: feed of food type **f** the amount **x** to home **h**.

Any given command with an ID higher than 4 should lead to the program terminating without a violation. Any subsequent commands given can be ignored.

4.5 Output

The output must consist of a (space-separated) sequence of responses, each reflecting the result of executing the corresponding command in the input sequence. A response is either of the form **c**, with **c** being the number of the corresponding input command (see the list given before), in case the command could be executed without any issues, or **c!** if the command could, for whatever reason, not be executed. You should think about what can possibly go wrong, and prevent that your program executes a command under any of those circumstances.

A given command with an ID higher than 4 (indicating termination) should not lead to any output.

4.6 Specifications

1. There are 10 cages in the Zoo, numbered 0 to 9, and 5 enclosures, numbered 10 to 14;

2. Each animal has a unique name. Accommodation of more than one animal with the same name is not allowed;

3. You have a single storage for each type of food (one for hay, one for corn, etc.), each with a maximum capacity of 100. Each storage initially contains no food.

4. As your program is going to be automatically checked by Momotor, please, avoid in the output any (except an empty line in the end) extra strings or any extra characters, including leading or trailing spaces.

5. Your submission should consist of a number of Java files, with one called `MyZoo.java` (for the *MyZoo* class) and one called `AnimalKeeper.java`, containing the `main` method.

4.7 Restrictions and recommendations

This assignment is about object-oriented programming. There should be a class called *MyZoo* in your program, which is to be instantiated in a client program. The `main` method of the client program should be in a file called `AnimalKeeper.java`. It should be possible to use the methods of the *MyZoo* class to process the commands given in the input. All other entities in the Zoo (animals, food, homes) should be classes as well. Think carefully about the fields and methods that each class should have. You will be assessed on the quality of your class design.

4.8 Examples

Input	**Output**	**Explanation**
0 1 Simba 0 0 1 Musafa 5 0 7 Paddington 0 5	0 0 0!	The third command violates the fact that Lions and Bears do not like to live with carnivores, and that Bears are solitary..
0 1 Simba 0 0 4 Marty 11 0 7 Paddington 3 0 5 Spot 11 3 2 10 3 4 20 4 4 10 11 6	0 0 0 0 3 3 4!	The wrong food is offered to at least one animal.
0 2 Bagheera 10 0 2 Bagheera 2 8	0 0!	An animal name must be unique.
0 6 Marty 12 3 2 110 4 2 10 12 7	0 3! 4!	Too much food was bought to be stored, and because of that, no food is available to feed the Giraffe.

4.9 Submitting the assignment

Upload to Canvas all your Java files of the project in a zip file.