Java programming II - Online

Exception Handling and GUI

Assignment 3 – AnimalFarm Version 3.0

Mandatory

Farid Naisan
University Lecturer
Malmö University, Malmö Sweden
UCSD Extension, CA, USA

The AnimalFarm- GUI and Exceptions

1 Objectives

The main goals of this assignment are:

1. To work more with GUI components and graphical user interface.
2. To develop programs with as least error-prone as possible.
3. To learn to handle exceptions, using standard and custom exceptions.
4. To discover more of polymorphic behavior.

In this assignment, we will produce a version 3 of our software project, the **AnimalFarm**. As far as the version numbering is concerned, one normally does not step up by one whole number, for example from version 2 to till 3, for such small changes as in this assignment, but here, we would like the version numbers to match the assignment number.

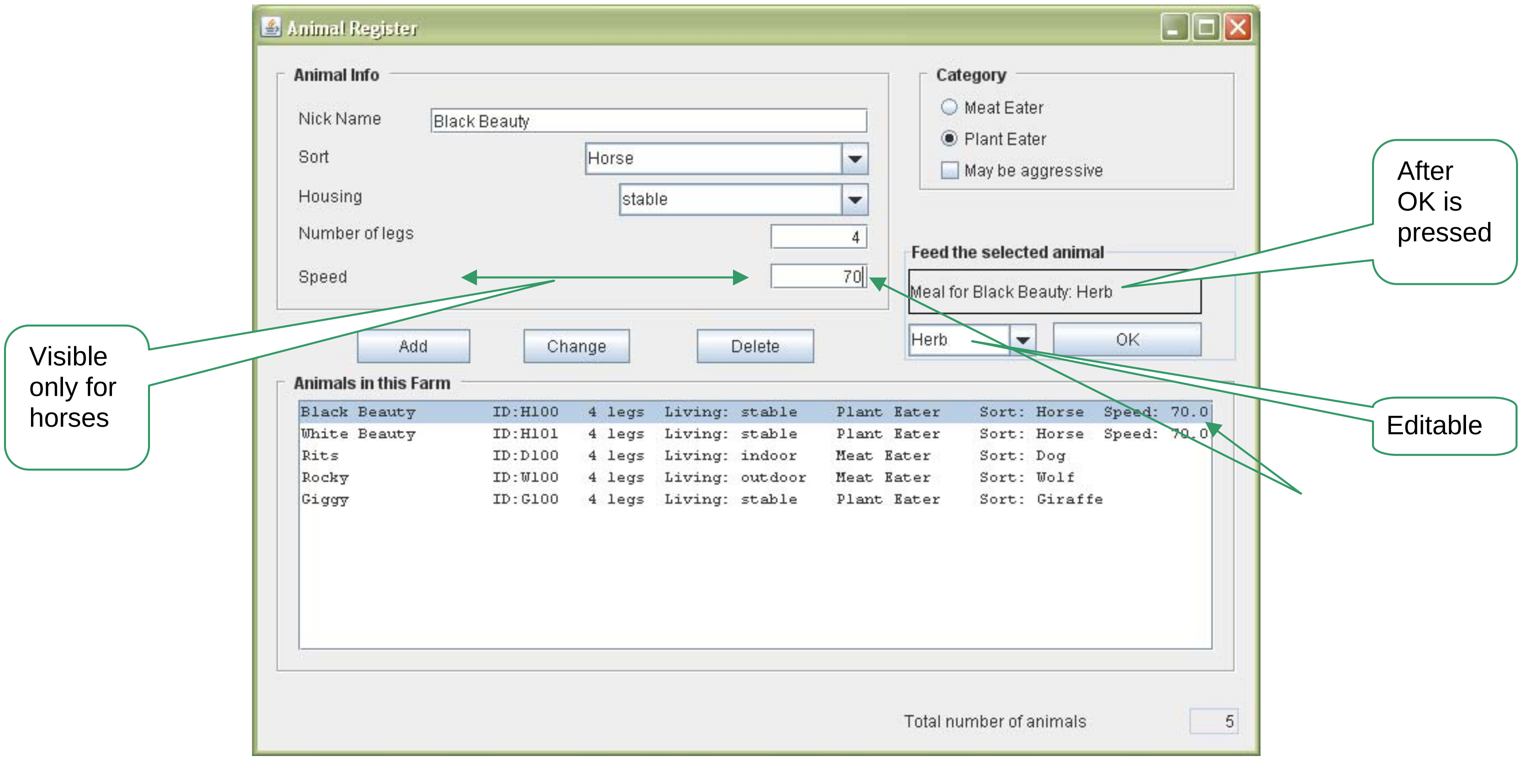
2 Description

Make sure that you have taken a back-up copy (or a master copy) of your Version 2 of the AnimalFarm from the last assignment, so you can always go back to a last good version. You may continue with your last project, or start a new one and copy files from your last version.

In version 3 of our AnimalFarm project, your job is to introduce some new features in the program and bring improvement in the code project.

3 To Do - GUI Enhancements:

Complete the GUI so that the user gets the possibility of choosing and testing meals for an animal selected in the list. The user can select a food item either from the values listed in a combobox by the program, or by writing an item directly in textfield part of the combobox (see the run example below).



Nick Name: Black Beauty
Sort: Horse
Housing: stable
Number of legs: 4
Speed: 70
Category: ( ) Meat Eater [✓ Plant Eater] [ ] May be aggressive
Meal for Black Beauty: Herb
Herb

```
Black Beauty        ID:H100   4 legs  Living: stable    Plant Eater   Sort: Horse   Speed: 70.0
White Beauty        ID:H101   4 legs  Living: stable    Plant Eater   Sort: Horse   Speed: 70.0
Rits                ID:D100   4 legs  Living: indoor    Meat Eater    Sort: Dog
Rocky               ID:W100   4 legs  Living: outdoor   Meat Eater    Sort: Wolf
Giggy               ID:G100   4 legs  Living: stable    Plant Eater   Sort: Giraffe
```

Total number of animals: 5

4 To Do - Code Enhancements

4.1 Write a public enum **AnimalSort**, and define at least 10 sorts of animals inclusive those from the last version (Horse, Dog, Lion, etc).

4.2 Let the animal class have an extra private field, **sort**: of the above type.

```
private AnimalSort sort;                  .
```

4.3 Put the following two methods in the **IAnimal** interface:

```
public AnimalSort getSort();
public void setSort(AnimalSort value);
```

The methods must be implemented in the **Animal** class. Every sub-class of **Animal** such as **Horse**, **Giraffe**, etc must call **setSort** to set the appropriate animal sort in the **Animal** object.

4.4 Declare a private field, **speed**, in the **Horse** class. Write get- and set methods connected to this variable. The value should be visible on the GUI as shown in the run example above.

4.5 **Important**: Assure that the method **eat** is declared as abstract in the **Animal** class.

4.6 **Important**: Make also sure that the **AnimalManager** class does not return a reference to the **ArrayList** object used there, to other client objects. If your **ArrayList** object has the name animals, the following code is not very safe,

```
return animals;
```

and should not occur in any method (getter method for example).

4.7 Write instead a method **getElemen**t(int index) in your AnimalManager class that returns the correct type of object depending on the value of the sort. If the sort is a **Horse**, it should return an **Horse** object. Why? Discuss this on the forum.

4.8 As a matter of fact, returning the address of an individual element (through the getElement) in the **ArrayList** object is not safe either. Any changes in the caller method to the object, will affect your **ArrayList** object as well. What can we do then? Simple, create a copy of the element and return the address of the copy! We may probably examine this in the next assignment and I will show you how.

5 To Do – Add new features

5.1 Make the Change button work. When the user selects an item in the listbox, the information related to the item should be shown in the input boxes so the use can change these.

5.2 Before accepting the changes, ask the user (by a JOptionPane) to think twice. Take the change action when the user confirms the choice.

5.3 Ask the user to confirm, when she or he will delete an object.

5.4 Implement the functionality of the check box so that the information is saved somewhere. Make your own decision

5.5 The feeding of the animal selected in the listbox should also work. Call the **eat** method to control if the food is appropriate for the animal (i.e if the item exists within the list of food items in the **MeatEater** or **PlantEater** classes). If the food chosen (or written by hand) is not of the proper type, the **AnimalDontEatThisException** (your own exception, see below) should be thrown and the related message should be given on the GUI or through a message box to the user.

6 Errors and exceptions

Every good programmer takes care of the normal errors so that the program does not crash for improper input, division by zero and indexing outside the range of an array, etc. There are mainly three types of errors:

Compile Errors: These are errors caused by a programmer by not abiding with the syntax and other rules of the programming language. These errors are detected by the compiler and as such are taken care of before the program is released.

Logical Errors: These types of errors are detected only by testing. The program simply put gives wrong results. The error can be a programmer error (wrong placement of parenthesis in formulas for example) or by the source of the logics used (wrong specification).

Run-Time Errors: These are errors that cannot be accounted for while programming. If the user, for example, tries to save a file on a drive where she or he does not have the permission to, the program is put in an exceptional situation which usually causes its abnormal termination.

In this version of the project, your job is to make your application crash-proof by trying to eliminate all possible sources of error. Your program should not crash!

6.1 Go through all your code and try to detect simple and foreseeable error sources and write code to prevent any abnormal behaviour. Write code to control user's input. Equip your setter functions with control of the "value" before accepting it. Use simple algorithm such as if-else, swich-default, and other simple statements whenever it can be applied.

6.2 Find then more severe type of errors, that can be out of your control and for these, use try-catch (and certainly also finally) constructs. Because exceptions require overhead, and involve even OS, they should be use sparingly.

6.3 Write a custom exception **AnimalDontEatThisException**. (must extend the Java's Exception class). This exception should be thrown (in the MeatEater's and PlantEater's **eat** method) when an animal is given a food that does not exist in the list of items that respective category can eat. (You may certainly put the food items in enums and even make a class FoodItem, if you wish).

7 Some hints

7.1 Writing a simple search-method in the AnimalManager class, is going to be very useful in many ways. This method can (for example) return the index to the position in the ArrayList, in which an object matching the search criteria (e.g. sort+name) is stored, and (-1) if not found.

7.2 This search method may be then utilized in implementing the change and delete metods. Consider the following algorithm:

```
Add
    - Read input and if input I valid
        - Add at the end of list (or search for a vacant position, if you use ordinary arrays)
        - Update GUI
            - Clear the contents of the listbox
            - Refill the listbox with current data from the AnimalManager object
Change
    - Read (changed) input (as in above)
    - Search for the object
        - if found
            - Replace the object (or remove the object and then add at then end, as in above)
            - Update GUI (as above)
        - else
            - Give a message to the user and do nothing
Delete
    - Do as Change, but remove instead of replacing
```

A couple of advices from me to you when designing your GUI:

1. ***Keep your GUI simple; have always the "dumb" user in mind.***
2. ***Follow the standards (File-View…Help when designing menus)***

Good Luck!

Programming is fun. Never give up. Ask for help!

Farid Naisan,