AP® Computer Science A

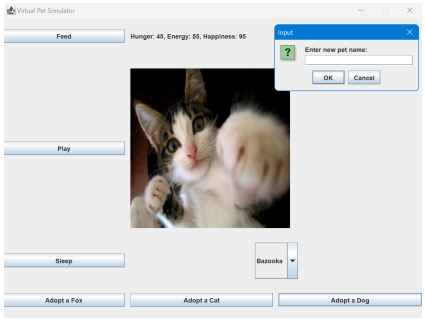
Virtual Pet Simulator Lab

Dr. Daniel Szelogowski

March 29, 2024

1 Introduction

This assignment will guide you through the process of developing a Virtual Pet Simulator using Java and Swing. You will practice object-oriented programming, event handling, and GUI design concepts.



2 Setup

Start by downloading the starter code and open it in IntelliJ IDEA. Familiarize yourself with the provided classes and their current functionalities.

3 Assignment Tasks

3.1 PetManager Class

1. Implement methods to add a pet, remove a pet, and get a list of all pets.
2. Ensure the pet list is updated when a new pet is adopted (see *Section 3.4*).

3.2 MainForm Modifications

1. Populate the `petSelectorComboBox` with the names of the pets by implementing the `updatePetList()` method. Use the `PetManager` class to retrieve the pet names. It may be helpful to make a `getSelectedPet()` method as well.
2. Implement the actions for the feed, play, and sleep buttons. These should affect the current selected pet's state and update the status label. Call the provided `waitButtons(seconds)` method to prevent UI spamming.
3. Update the pet status label whenever a pet's state changes or a new pet is selected.
4. Add functionality to the pet selector to update the UI when a different pet is selected. You'll need to update the status label and pet image.

3.3 Pet, Cat, and Dog Classes

1. Implement a method in `Pet` to return the pet's status as a string (i.e., its hunger, energy, and happiness levels). Override this method if needed in subclasses.

2. Override the `feed()`, `play()`, and `sleep()` methods in `Cat` and `Dog` classes to reflect their unique behaviors. You'll need to call the superclass `get` and `set` methods to update the values.
   - **Feed Method:**
     - Increase the pet's energy and happiness.
     - Decrease its hunger.
     - Ensure that none of the pet's state attributes exceed their maximum values or drop below 0.
   - **Play Method:**
     - Increase the pet's happiness.
     - Decrease its energy and increase hunger due to exertion.
     - Prevent the pet's happiness or hunger from exceeding their bounds.
   - **Sleep Method:**
     - Increase the pet's energy.
     - Slightly decrease its hunger and increase happiness.
     - Make sure the energy does not surpass its limit.
   - For `Cat` and `Dog` classes, implement these methods with specific behaviors:
     - Cats may have a higher increase in happiness when played with, but also get hungrier faster.
     - Dogs might gain more energy from sleeping but also require more food when fed.
   - Include conditional checks within each method to maintain the pet's state within realistic bounds. **All values should be between 0 and 100, inclusive**.
   - Update the pet's status in the GUI after each action to reflect the changes in its state. This requires invoking the status update method after executing feed, play, or sleep actions.

3.4 Adding New Pets

1. Add buttons or a dialog to the UI allowing users to adopt a new cat or dog (see *Section 3.4.1* if you aren't working in IntelliJ).

2. Prompt the user to enter a name for the new pet and add the pet to the pet manager. You can make a dialog input box using:
   `String name = JOptionPane.showInputDialog("Enter new pet name:  ");`
   Be sure NOT to add the pet if `name` is empty (`name.trim().isEmpty()`) or null.

3. Update the pet list to reflect the addition.

3.4.1 Adding Adoption Buttons without Swing Designer

To add buttons for adopting a cat or dog at the bottom of your form if you're using a different IDE, you'll need to adjust the row-count of your grid layout to accommodate these new components. Since your current layout has a row-count of 3, you'll want to increase this to 4 to add another row for the new buttons. Update the **row-count** value in line 3 so it has a value of 4, like so:

```
<grid id="27dc6" binding="mainPanel" layout-manager="GridLayoutManager" row-count="4"
column-count="3" same-size-horizontally="false" same-size-vertically="false" hgap="-1"
vgap="-1">
```

Then, you can paste the XML components for the adoption buttons into your **MainForm.form** XML inside the `<children>` tag at the end (in between lines 56 and 57) to add them to the new fourth row:

```
<!-- Adopt a Cat Button -->
<component id="4a8k8" class="javax.swing.JButton" binding="adoptCatButton">
  <constraints>
    <grid row="3" column="1" row-span="1" col-span="1" vsize-policy="1" hsize-policy="3"
    anchor="0" fill="1" indent="0" use-parent-layout="false"/>
  </constraints>
  <properties>
    <text value="Adopt a Cat"/>
  </properties>
</component>
<!-- Adopt a Dog Button -->
<component id="5b9l9" class="javax.swing.JButton" binding="adoptDogButton">
  <constraints>
    <grid row="3" column="2" row-span="1" col-span="1" vsize-policy="1" hsize-policy="3"
    anchor="0" fill="1" indent="0" use-parent-layout="false"/>
  </constraints>
  <properties>
    <text value="Adopt a Dog"/>
  </properties>
</component>
```

Afterwards, you'll need to manually add these buttons to the private fields of the **MainForm.java** class, like so:

```
public class MainForm extends JFrame {
    ...
    private JButton adoptCatButton;
    private JButton adoptDogButton;
    ...
```

Lastly, be sure to add an Action Listener for each button inside the constructor so you can program their functionality.

3.5 Extra Credit: Pet Fox

If you finish the project early, your last step will be to add a new child class `Fox` that inherits from `Pet` and overrides the interaction methods similar to the `Cat` and `Dog` classes. You can also switch this out with a different animal provided you can find a JSON API that returns a random image of that animal.

First, you should add a new method to the `ImageHelper` class to grab a random fox picture:

```
public static String getRandomFoxImg() {
    return getImgFromJSON("https://randomfox.ca/floof/?ref=apilist.fun", "\"image\":\"", "\",\"link\"");
}
```

Next, you'll need to call that method inside the `Fox` class constructor:

```
public Fox(String name) {
    super(name);
    super.setImage(ImageHelper.getRandomFoxImg());
}
```

Then, override the `feed()`, `play()`, and `sleep()` methods in the class using the same rules as the other two animals (see *Section 3.3*), but with the following differences in behaviors:

- **Feed:** Foxes are omnivores with a diet that can vary widely. When feeding a fox, you could increase its happiness slightly along with reducing hunger, as finding food is a significant part of a wild fox's day that affects its mood. However, feeding a fox doesn't increase its energy as much as a dog, reflecting the cautious nature of foxes around food.
- **Play:** Foxes are known for their playful behavior, showing a lot of energy and curiosity. Playing with a fox could significantly increase its energy and happiness more than it does for cats and dogs, but also make it hungrier, reflecting the physical exertion and metabolic needs.
- **Sleep:** Foxes in the wild are opportunistic and can sleep both day and night, adjusting their schedule as needed. When a fox sleeps, it could gain a moderate amount of energy but also a slight increase in happiness, reflecting the satisfaction of resting in a safe, comfortable place. However, their metabolism slows down while sleeping, so hunger is not as affected.

Finally, be sure to add an **Adopt a Fox** button to the form and give it the relevant functionality, similar to the adoption buttons for cats and dogs.

4 Free Response Questions

Name:

1. Explain how inheritance is used in the Virtual Pet Simulator.

   Inheritance is used so that the three main pet classes are able to use certain methods within the pet class, so that the amount of work is heavily reduced. As one does not have to copy the methods into each class. All the three main pet classes inheriting a pet class means that they all can be called in as a "pet" and thus can easily be put into an ArrayList of that class.

2. Discuss the benefits of polymorphism in the context of this project.

   The three main pet classes being able to be called in as a pet means that they can all be added to a single array list rather than having them all be in separate lists.

3. How does encapsulation improve the design of the Virtual Pet Simulator?

   Rather than each individual pet class having their own methods, having the "Pet" class encapsulate everything allows for a much simpler system that allows each pet class to inherit

4. Describe how event handling is implemented in the Swing GUI components.

When I click a button, it runs multiple functions.

5. Consider the method for adding a new pet to the simulation. How might this process be improved to handle different types of pets beyond cats and dogs?

The pets could be sorted into their pet types rather than all be clumped together as "Pets." This could allow for easier access of each type rather than having to sort through and array list of all pet types. It could also make finding an individual pet easier if that pet's type is known

6. Consider the addition of a `PetShop` class to the simulator, where users can 'purchase' pets with virtual currency earned through taking care of their existing pets. Describe how you would design the `PetShop` class, considering the principles of encapsulation and modularity. What methods and properties would it have? How would it interact with the existing `PetManager` and pet classes (Cat, Dog, Fox)?

The PetShop would be an independent class that contains data storing Cat, Dog, and Fox. It would have a return method for each purchasable pet that would return a new pet named by the user as well as reducing the user's virtual currency by the pet's cost.