Animal Shelter Project

By Anastassia Tarassova (1277712)

and

Dang Huynh Minh (1670214)

Presentation Structure

Why the Project

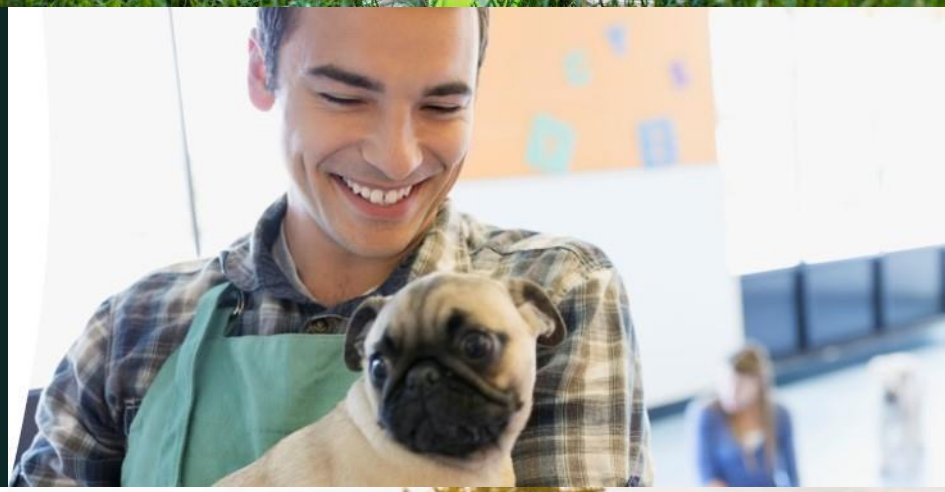
First Iteration

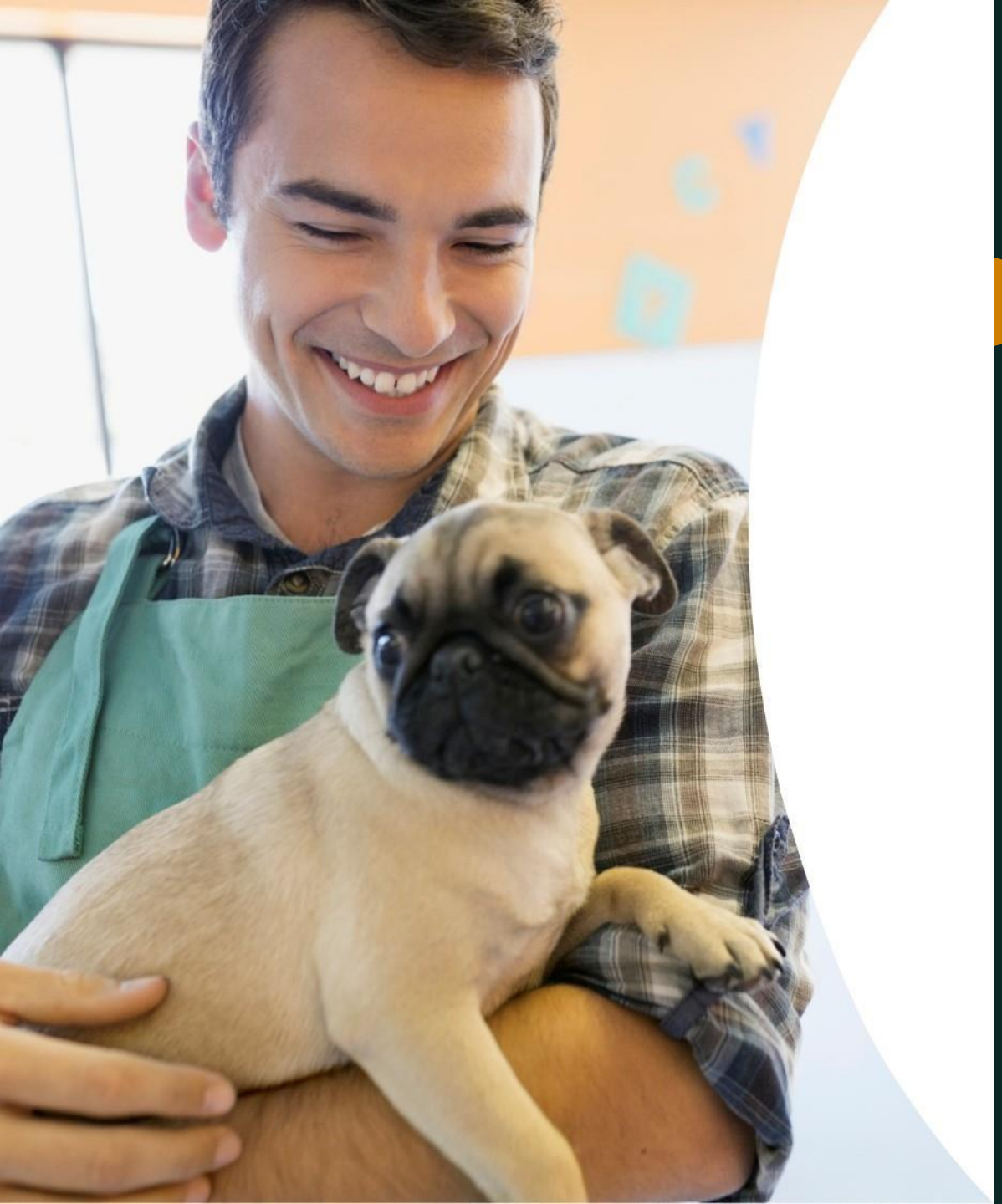
Current Iteration

SOLID, Design Patterns, MVC, Three-Tier

Unit Testing

Potential Improvements





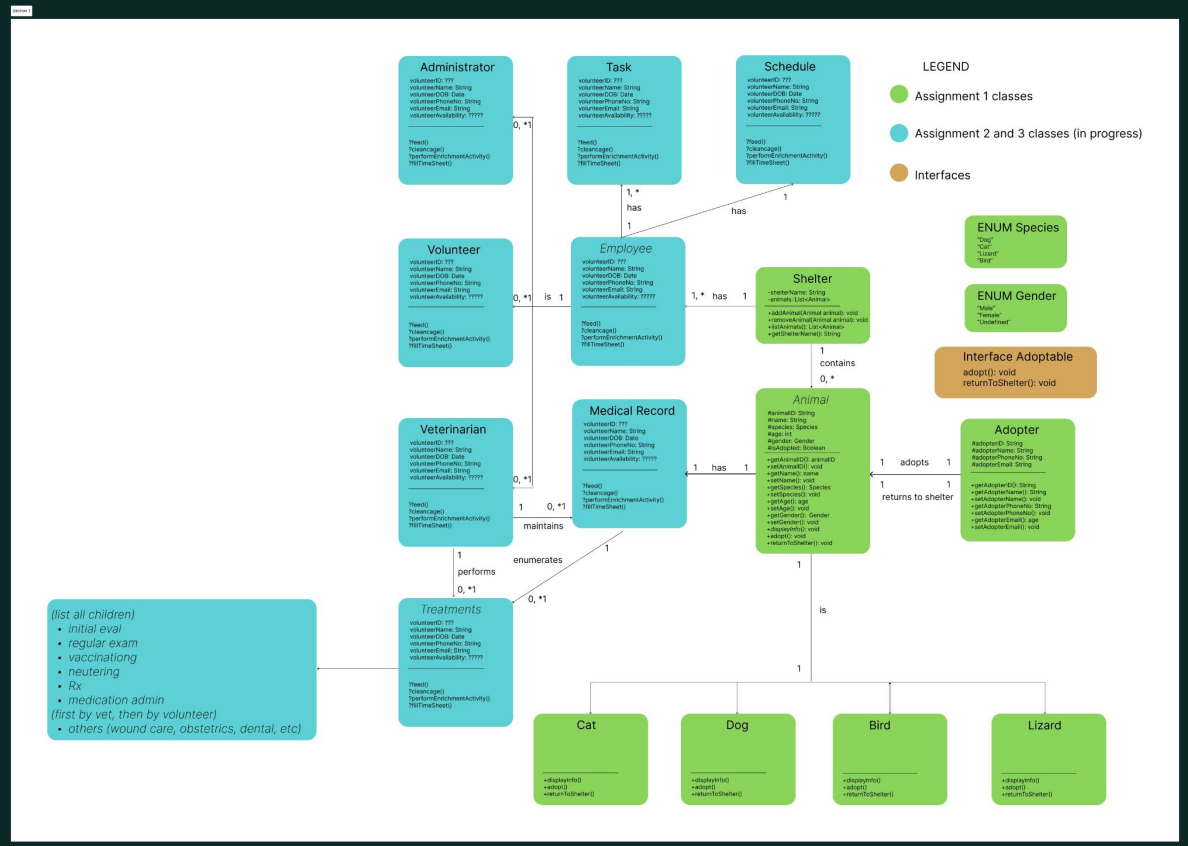
1. Why Animal Shelter?

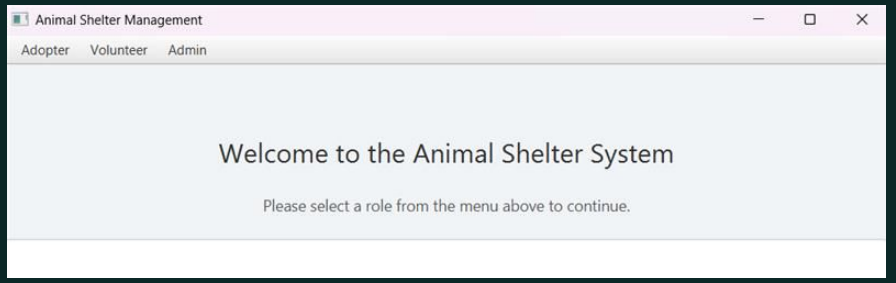
- Continuity of past course
- Real-life application
- Multitude of designs
- Cute!

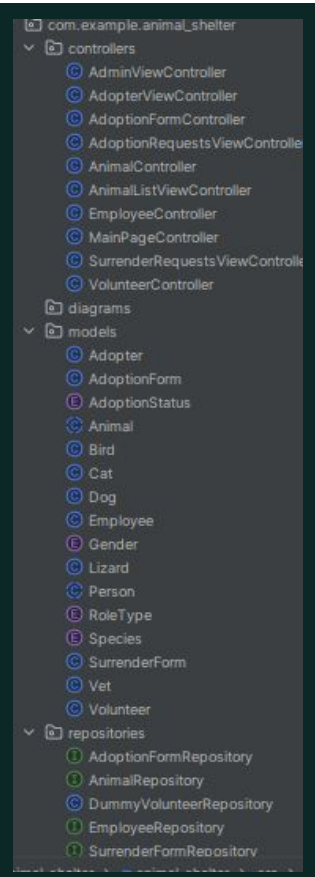
2. First Iteration (Java III course)

- extensive UML
- multitude of classes
- three user roles planned (Admin, Volunteer, Vet)

First Iteration



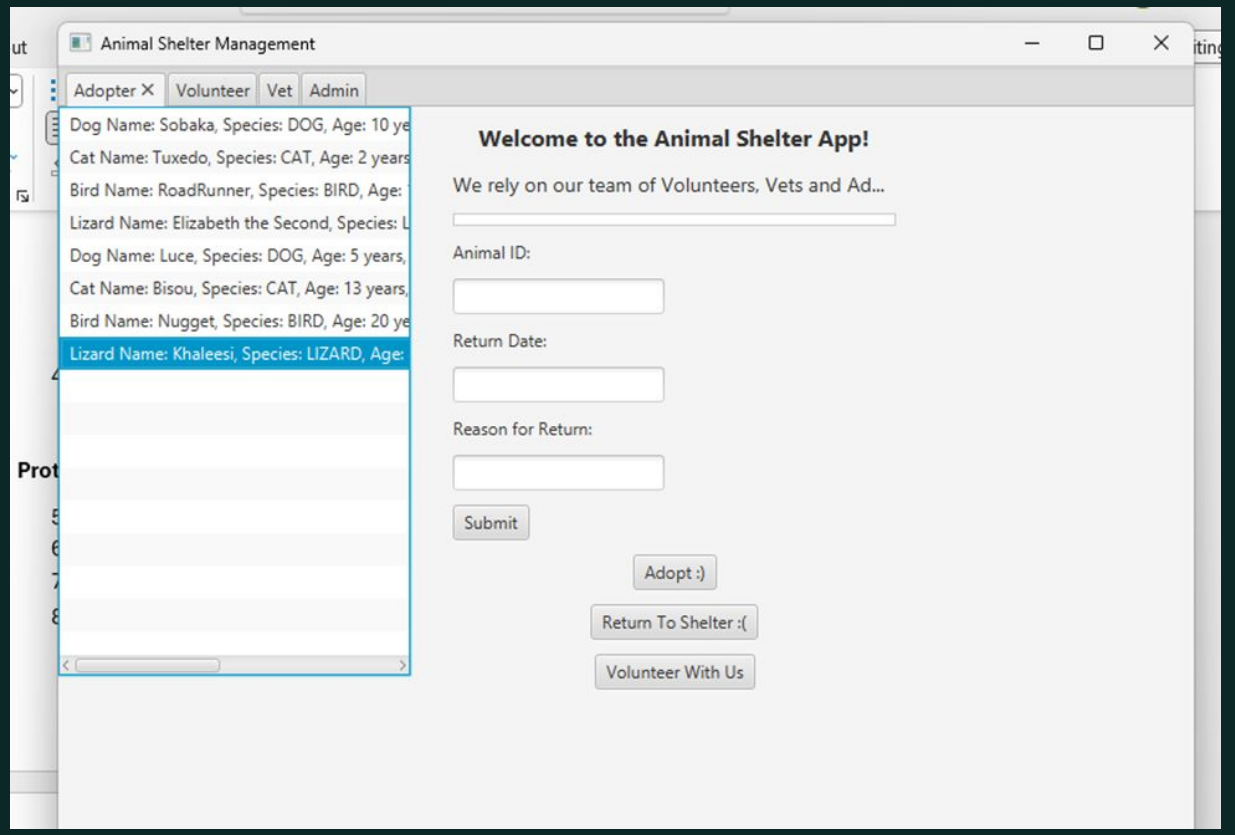




First Iteration

Pros:

- Ambitious
- Abstract classes galore
- Realistic user stories



Cons:

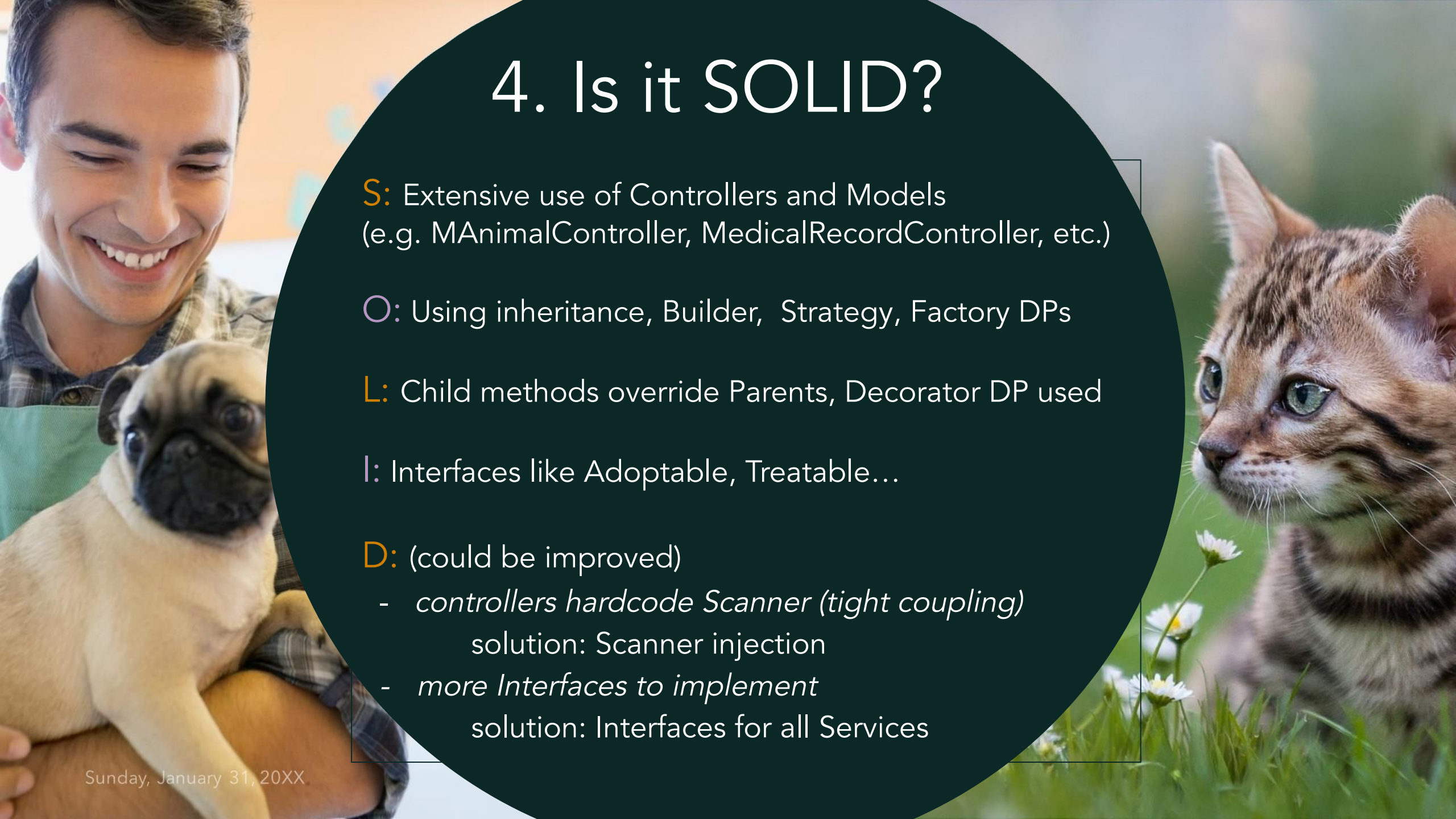
- Structural issues (not SOLID)
- Repetition (not quite DRY)
- Incomplete logic (very little design patterns)
- No automated testers

3.Current Iteration

with

Data Structures and Algorithms

- extensive UML
- multitude of classes
- CLI and JavaFX friendly architecture



4. Is it SOLID?

S: Extensive use of Controllers and Models
(e.g. MAnimalController, MedicalRecordController, etc.)

O: Using inheritance, Builder, Strategy, Factory DPs

L: Child methods override Parents, Decorator DP used

I: Interfaces like Adoptable, Treatable…

D: (could be improved)

- *controllers hardcode Scanner (tight coupling)*
  solution: Scanner injection
- *more Interfaces to implement*
  solution: Interfaces for all Services

Did you use data structures?

ArrayList <Animal>:

- good random access and iteration

Queue <Animal> (Shelter Queue)

- FIFO adoption, task queues…

HashMap <String, Animal>

- looking up animals by String id

List <String> for MedicalRecord

- listing check-ups, vaccines, treatments

AnimalRegistry.java

```java
/**
 * Returns all animals currently registered in the shelter.
 *
 * @return a list of all registered animals
 */
public List<Animal> getAllAnimals() { return animalList; }

/**
 * Finds an animal by its exact ID using binary search.
 * The animal list is first sorted by ID before performing the search.
 *
 * @param id the ID to search for
 * @return the animal if found; null otherwise
 */
public Animal findById(String id) {  9 usages  ▲ Ana Tara
    // if no animals in a shelter
    if (id == null || id.isEmpty()) return null;

    // sorting the animal list with stream before binary search
    List<Animal> sorted = animalList.stream()
            .sorted(Comparator.comparing(Animal::getId))
            .collect(Collectors.toList());

    // attribute an index to each sorted element
    int index = binarySearchById(sorted, id);
    // if no animals, nothing to sort, no indexes
    return (index >= 0) ? sorted.get(index) : null;
}
```

What about algorithms?

search by name:

linear scan over list (O(n)) + binary

search by id:

currently linear (also O(n), could be improved with a Map)

sorting:

Java’s built-in sort() with custom comparators for name and age

Binary Search by Id

AnimalRegistry.java (cont.)

```java
private int binarySearchById(List<Animal> sortedList, String id) {  1 usage  Ana Tara
    int low = 0;
    int high = sortedList.size() - 1;

    while (low <= high) {
        // start with middle
        int middle = (low + high) >>> 1;
        Animal midAnimal = sortedList.get(middle);
        int cmp = midAnimal.getId().compareTo(id);
        if (cmp == 0) return middle;
        if (cmp < 0) low = middle + 1;
        else high = middle - 1;
    }

    return -1;
}

/**
 * Checks whether the registry is empty.
 *
 * @return true if the registry has no animals; false otherwise
 */
public boolean isEmpty() { return animalList.isEmpty(); }

/**
 * Returns the maximum capacity of the shelter.
 *
 * @return the maximum number of animals that can be registered
 */
public int getMaxCapacity() { return maxCapacity; }
```

Best case:

can be O(1)

if we land directly on searched Animal

Worst case:

can be O(logn)

because of call stack from recursive search until Animal found

Algorithms (cont.)

FIFO:

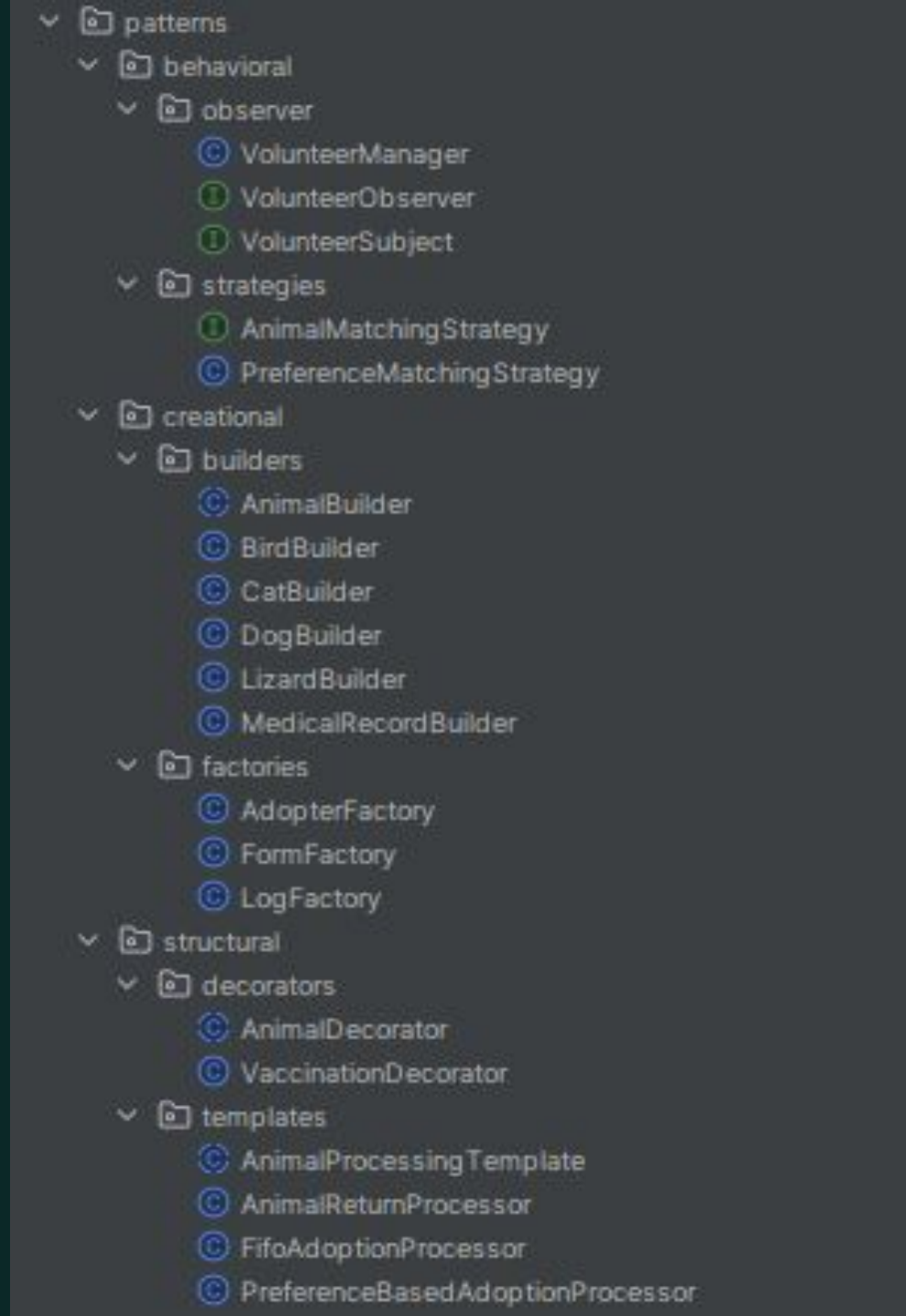
enqueue/dequeue

prompt validators:

basic loops for checking booleans/numeric inputs (in Animal Controller)

builders:

not quite an algorithm, but contributes to their efficiency

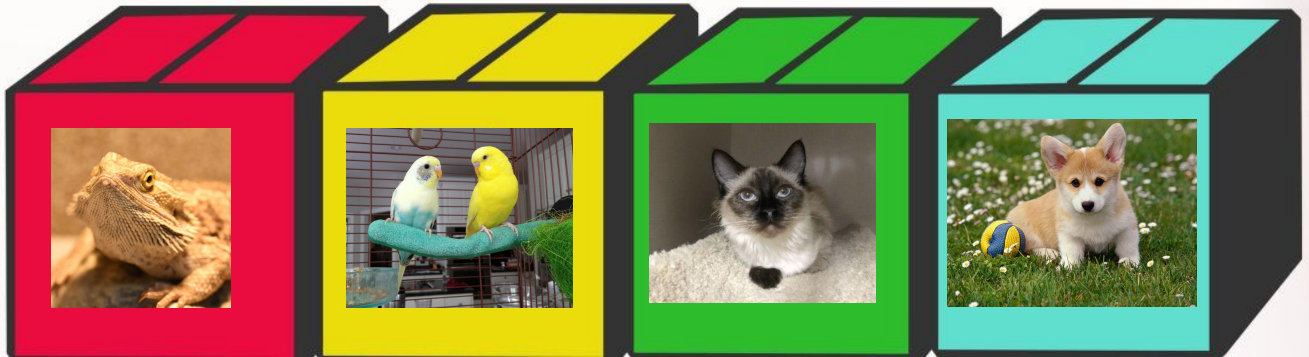


FIFO

First In, First Out

Marketed as

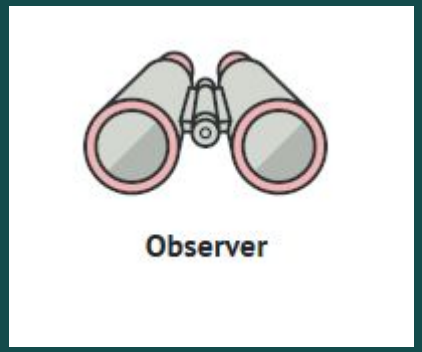
Animal of the Month

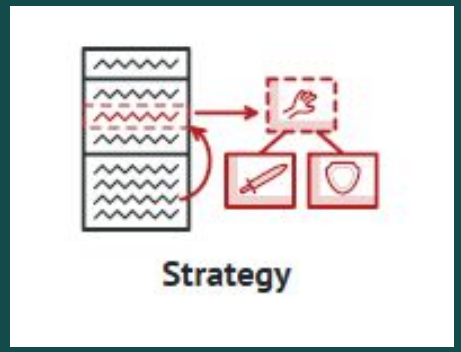


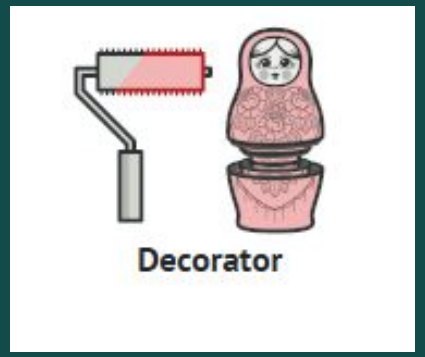
Design Patterns Used

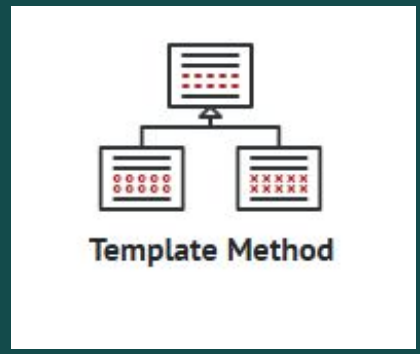
Behavioral

Structural









Observer

New arrivals signaled to:

- Volunteers
- Vets
- Admins

…

Strategy

Adoption strategies for:

- Animal of the Month
- Preferred Animal

…

Decorator

- Animal decorator
- Vaccine decorator

Template

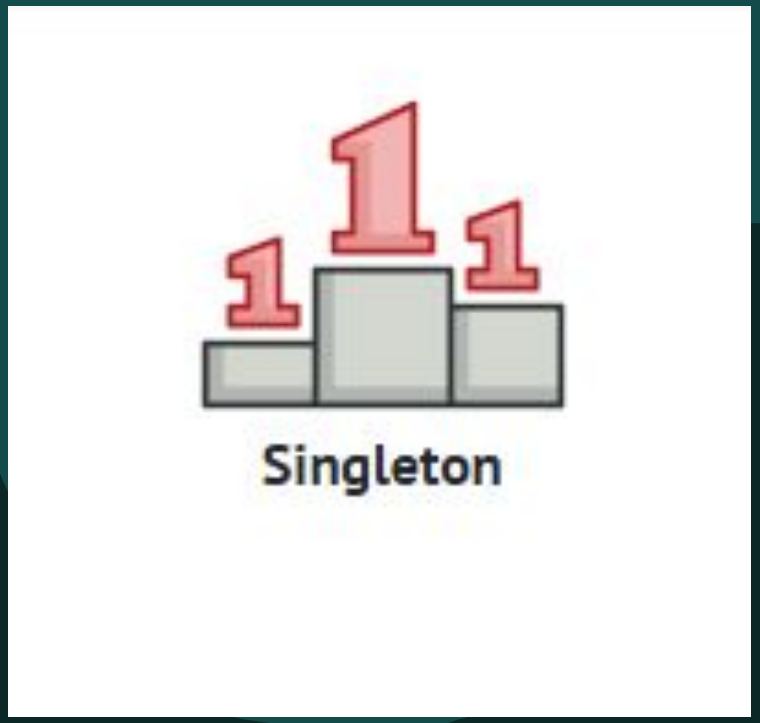
- FIFO adoption
- Preferential adoption

…

Creational

Singleton

Main class



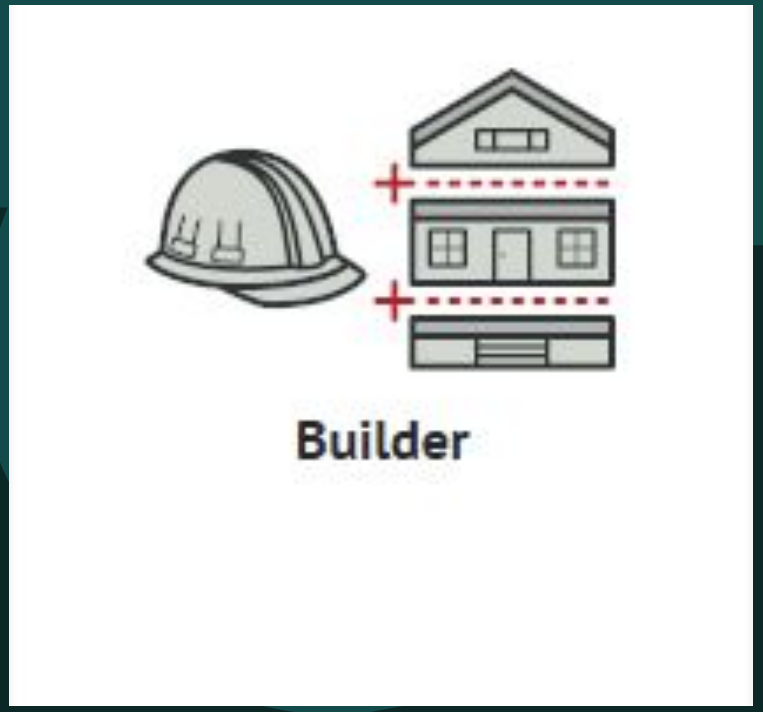
Builder

- Animal

- animal subclasses

-MedicalRecords

…



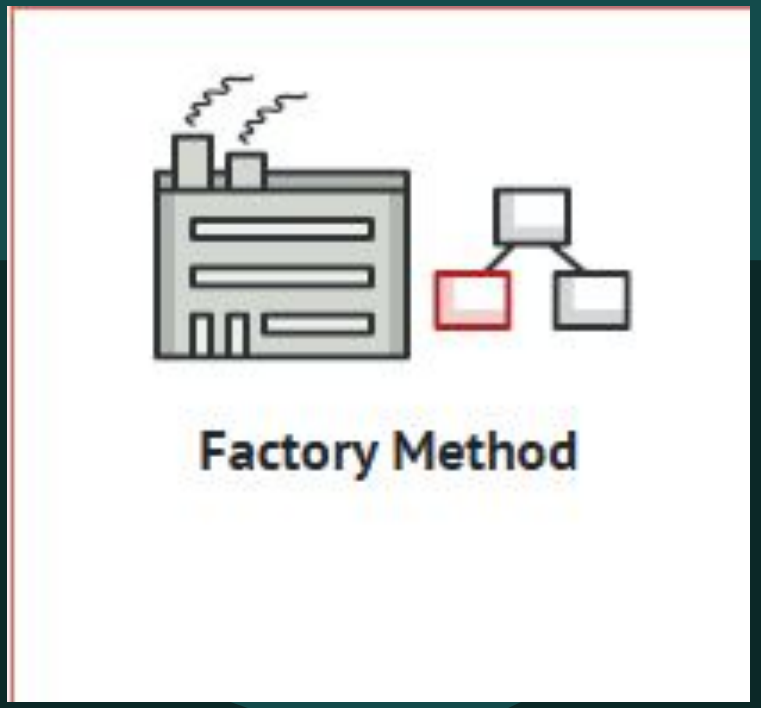
Factory

-Adopters

- Forms

- Logs

…



Animal Builder (abstract)

```java
/**
 * Abstract base builder for constructing {@link Animal} objects using the Builder pattern.
 * <p>
 * Uses the Curiously Recurring Generic Pattern (CRGP) to allow method chaining
 * with concrete subclasses returning their own type.
 * <p>
 * Provides common properties for all animals, such as name, age, species, breed, and medical record.
 *
 * @param <T> the type of {@link Animal} this builder constructs
 * @param <B> the concrete builder type extending this class (used for fluent method chaining)
 */
public abstract class AnimalBuilder<T extends Animal, B extends AnimalBuilder<T, B>> {  8 usages  4 inheritors  ▲ An
    private static final Logger logger = LogFactory.getLogger(AnimalBuilder.class);  5 usages

    protected String name;
    protected int age;
    protected MedicalRecord medicalRecord;  5 usages
    protected String species;
    protected String breed;

    /**
     * Returns the concrete builder instance.
     * <p>
     * Used internally to enable fluent method chaining in subclasses.
     *
     * @return the builder instance of type B
     */
    protected abstract B self();  5 usages  4 implementations  ▲ Ana Tara
```

[...]

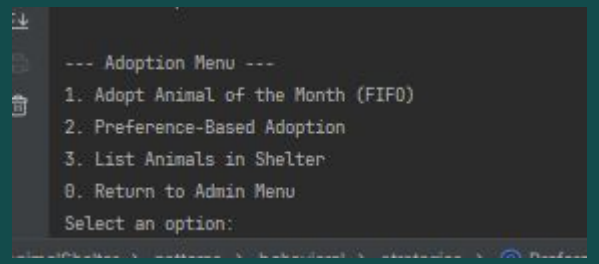
```java
    /**
     * Sets the species of the animal.
     *
     * @param species the species name (e.g., "Canine", "Feline")
     * @return this builder instance for chaining
     */
    public B setSpecies(String species) {  ▲ Ana Tara
        this.species = species;
        logger.fine( msg: "Set species: " + species);
        return self();
    }

    /**
     * Sets the breed of the animal.
     *
     * @param breed the breed name (e.g., "Labrador", "Siamese")
     * @return this builder instance for chaining
     */
    public B setBreed(String breed) {  1 override  ▲ Ana Tara
        this.breed = breed;
        logger.fine( msg: "Set breed: " + breed);
        return self();
    }

    /**
     * Builds and returns the constructed animal instance.
     * <p>
     * Concrete subclasses must implement this method.
     *
     * @return a new instance of T representing the built animal
     */
    public abstract T build();  4 implementations  ▲ Ana Tara
}
```

e.g. Bird builder (concrete)

```java
package patterns.creational.builders;

import models.animals.Bird;

/**
 * Builder for creating {@link Bird} instances.
 * <p>
 * Extends the generic {@link AnimalBuilder} with bird-specific properties such as
 * breed and ability to fly.
 */
public class BirdBuilder extends AnimalBuilder<Bird, BirdBuilder> {  7 usages  Ana Tara
    private String breed;  2 usages
    private boolean canFly;  2 usages          Tara, 2025-05-20 9:41 a.m. • Bird and Lizard classes impleme

    @Override  5 usages  Ana Tara
    protected BirdBuilder self() { return this; }

    /**
     * Sets the breed of the bird.
     *
     * @param breed the breed name
     * @return this builder instance for chaining
     */
    public BirdBuilder setBreed(String breed) {  Ana Tara
        this.breed = breed;
        return this;
    }
```

```java
    /**
     * Sets whether the bird can fly.
     *
     * @param canFly true if the bird can fly, false otherwise
     * @return this builder instance for chaining
     */
    public BirdBuilder setCanFly(boolean canFly) {  2 usages  Ana Tara
        this.canFly = canFly;
        return this;
    }

    /**
     * Builds a new {@link Bird} instance with the configured properties.
     *
     * @return a new {@link Bird} object
     */
    @Override  Ana Tara
    public Bird build() { return new Bird(name, age, breed, canFly, medicalRecord); }
}
```

e.g. adding a Lizard




```
0
Allowed: [dog, cat, bird, lizard]
Enter type (Dog/Cat/Bird/Lizard): lizard
Enter name: bob
Enter age (0-30): 1
Enter breed (or 'Unknown'): unknown
Is the lizard venomous? (true/false): true
Enter vaccinations (type 'done' to finish):
done
Enter treatments (type 'done' to finish):
done
Enter checkups (type 'done' to finish):
done
May 29, 2025 11:49:17 A.M. patterns.creational.builders.MedicalRecordBuilder build
INFO: Building MedicalRecord with 0 vaccinations, 0 treatments, 0 checkups.
May 29, 2025 11:49:17 A.M. patterns.behavioral.observer.VolunteerManager notifyVolunteers
INFO: Notifying 0 volunteer(s) of event:
New Lizard arrived: bob
Lizard added. Total: 1/20
[Background Thread] Logging new animal: bob
```

Preference-based Adoption Strategy

(Dijkstra-inspired)

AnimalMatchingStrategy.java
(Interface in strategies)

```java
package patterns.behavioral.strategies;

import models.Adopter;
import models.AdopterPrefereoces;
import models.AnimalRegistry;
import models.ShelterQueue;
import models.animals.Animal;

import java.util.List;

public interface AnimalMatchingStrategy {  11 usages  2 implementations  Ana Tara
    Animal selectAnimal(Adopter adopter, AnimalRegistry registry, ShelterQueue queue);  2 usages  2 implementa
}
```

```java
public class AdopterPreferences {  8 usages  Ana Tara
    private String species;  4 usages
    private String breed;  4 usages
    private Integer maxAge;  4 usages

    public AdopterPreferences() {}  3 usages  Ana Tara

    public AdopterPreferences(String species, String breed, Integer maxAge) {  2 usa
        this.species = species;
        this.breed = breed;
        this.maxAge = maxAge;
    }

    public String getSpecies() { return species; }

    public void setSpecies(String species) { this.species = species; }

    public String getBreed() { return breed; }

    public void setBreed(String breed) { this.breed = breed; }

    public Integer getMaxAge() { return maxAge; }

    public void setMaxAge(Integer maxAge) { this.maxAge = maxAge; }

    private final Map<String, String> preferences = new HashMap<>();  2 usages

    public void setPreference(String key, String value) {  1 usage  Ana Tara
        preferences.put(key.toLowerCase(), value.toLowerCase());
    }

    public String getPreference(String key) {  no usages  Ana Tara
        return preferences.get(key.toLowerCase());
    }
```

AdopterPreferences.java
(model)

Preference-based Adoption Strategy (cont.)

preferenceMatchingStrategy.java

```java
import models.Adopter;
import models.AdopterPreferences;
import models.AnimalRegistry;
import models.ShelterQueue;
import models.animals.Animal;

import java.util.List;
import java.util.Comparator;

public class PreferenceMatchingStrategy implements AnimalMatchingStrategy {  2 usages  Ana Tara *
    @Override  2 usages  Ana Tara
    public Animal selectAnimal(Adopter adopter, AnimalRegistry registry, ShelterQueue queue) {
        List<Animal> animals = registry.getAllAnimals();
        return findBestMatch(adopter, animals);
    }

    private Animal findBestMatch(Adopter adopter, List<Animal> availableAnimals) {  1 usage  Ana Tara *
        AdopterPreferences preferences = adopter.getPreferences();
        if (preferences == null || availableAnimals == null || availableAnimals.isEmpty()) {
            return null;
        }

        // Stream to find animal with max score, parallel stream used because animal list could be very large
                return availableAnimals.parallelStream()  Stream<Animal>
                .max(Comparator.comparingInt(animal -> getScore(animal, preferences)))  Optional<Animal>
                .orElse( other: null);
    }

    private int getScore(Animal animal, AdopterPreferences preferences) {  1 usage  Ana Tara
        int score = 0;
Tara, 2025-05-26 3:12 p.m. • Logger implemented. Logic for suggesting animals via user preferences in place, mo
        if (preferences.getSpecies() != null && animal.getSpecies() != null &&
                preferences.getSpecies().equalsIgnoreCase(animal.getSpecies().name())) {
            score++;
        }
```

[...]

Usage of Streams and Threads

- Mostly used in Controllers and Models, for the scope of this project

e.g. Adopter model

```java
/**
 * Searches for adopted animals by partial name match (case-insensitive).
 *
 * @param name part or full name of the animal
 * @return list of matching adopted animals
 */
public List<Animal> searchAdoptedAnimalsByName(String name) {  no usages  ▲ Ana Tara
    if (name == null || name.isEmpty()) return new ArrayList<>();

    return adoptedAnimals.stream()
            .filter(a -> a.getName().toLowerCase().contains(name.toLowerCase()))
            .collect(Collectors.toList());
}

/**
 * Returns the number of animals this adopter has adopted.
 *
 * @return count of adopted animals
 */
public int getAdoptedAnimalCount() {  no usages  ▲ Ana Tara
    return adoptedAnimals.size();
}
```

- Better code readability
- Improves performance times
- Better architecture

e.g VolunteerManager for Observer (parallel streams)

```java
    /**
     * Notifies all registered volunteers of the specified event.
     *
     * @param event a string describing the event to notify about
     */
    @Override  2 usages  ▲ Ana Tara +1 *
    public void notifyVolunteers(String event) {
        logger.info( msg: "Notifying " + volunteers.size() + " volunteer(s) of event: " + even
        volunteers.parallelStream().forEach(v -> v.update(event));
    }
}
Dang-Huynh, 2025-05-23 12:04 p.m. • Medical record and volunteer related changes
```

Walkthrough: adding a new animal to the shelter

AddAnimalController.java

```java
@FXML  ▲ Ana Tara +1
public void initialize() {
    dogFields.setVisible(false);
    catFields.setVisible(false);
    birdFields.setVisible(false);
    lizardFields.setVisible(false);

    typeChoiceBox.getSelectionModel().selectedItemProperty().addListener((obs, oldVal, newVal) -> {
        dogFields.setVisible(false);
        catFields.setVisible(false);
        birdFields.setVisible(false);
        lizardFields.setVisible(false);

        if ("Dog".equalsIgnoreCase(newVal)) dogFields.setVisible(true);
        else if ("Cat".equalsIgnoreCase(newVal)) catFields.setVisible(true);
        else if ("Bird".equalsIgnoreCase(newVal)) birdFields.setVisible(true);
        else if ("Lizard".equalsIgnoreCase(newVal)) lizardFields.setVisible(true);
    });
}
```

```java
private void handleAdd() {
    String name = nameField.getText().trim();
    String ageText = ageField.getText().trim();
    String type = typeChoiceBox.getValue();

    if (name.isEmpty() || ageText.isEmpty() || type == null) {
        showAlert( title: "Missing Fields",  msg: "Please complete all required fields.");
        return;
    }

    int age;
    try {
        age = Integer.parseInt(ageText);
        if (age < 0 || age > 30) {
            showAlert( title: "Invalid Age",  msg: "Please enter an age between 0 and 30.");
            return;
        }
    } catch (NumberFormatException e) {
        showAlert( title: "Invalid Age",  msg: "Age must be a number.");
        return;
    }

    Map<String, String> extras = switch (type.toLowerCase()) {
        case "dog" -> Map.of(
                k1: "breed", breedField.getText().trim(),
                k2: "trained", String.valueOf(trainedCheckBox.isSelected())
        );
        case "cat" -> Map.of(
                k1: "furLength", furLengthField.getText().trim(),
                k2: "indoor", String.valueOf(indoorCheckBox.isSelected())
        );
        case "bird" -> Map.of( k1: "canFly", String.valueOf(canFlyCheckBox.isSelected()));
        case "lizard" -> Map.of( k1: "venomous", String.valueOf(venomousCheckBox.isSelected()));
        default -> Map.of();
    };
```

AddAnimalController.java (cont.)

```java
    private void clearForm() {  1 usage   Dang-Huynh +1
        nameField.clear();
        ageField.clear();
        breedField.clear();
        trainedCheckBox.setSelected(false);
        furLengthField.clear();
        indoorCheckBox.setSelected(false);
        canFlyCheckBox.setSelected(false);
        venomousCheckBox.setSelected(false);
        typeChoiceBox.getSelectionModel().clearSelection();

        dogFields.setVisible(false);
        catFields.setVisible(false);
        birdFields.setVisible(false);
        lizardFields.setVisible(false);
    }

    private void showAlert(String title, String msg) {  5 usages   Dang-Huynh
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle(title);
        alert.setContentText(msg);
        alert.setHeaderText(null);
        alert.showAndWait();
    }

    public void shutdown() {  no usages   Ana Tara
        executor.shutdown();
    }
}
```

AnimalService.java

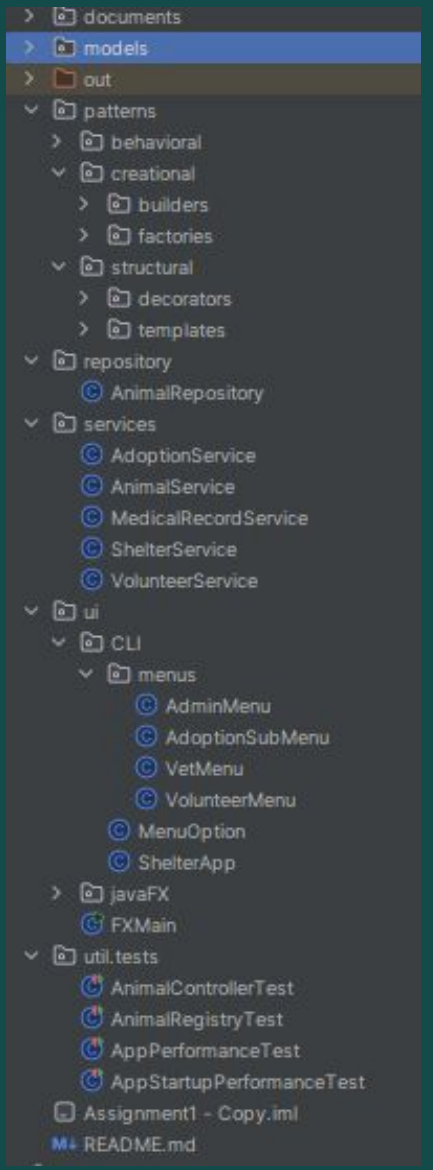
```java
public void addAnimalFromUI(String name, int age, String type, Map<String, String> extras) {  1 usage   Ana Tara
    Animal animal = switch (type.toLowerCase()) {
        case "dog" -> new DogBuilder()
                .setName(name)
                .setAge(age)
                .setBreed(extras.getOrDefault( key: "breed",  defaultValue: "Unknown"))
                .setTrained(Boolean.parseBoolean(extras.getOrDefault( key: "trained",  defaultValue: "false")))
                .build();
        case "cat" -> new CatBuilder()
                .setName(name)
                .setAge(age)
                .setBreed(extras.getOrDefault( key: "breed",  defaultValue: "Unknown"))
                .setFurLength(extras.getOrDefault( key: "furLength",  defaultValue: "Short"))
                .setIndoor(Boolean.parseBoolean(extras.getOrDefault( key: "indoor",  defaultValue: "false")))
                .build();
        case "bird" -> new BirdBuilder()
                .setName(name)
                .setAge(age)
                .setBreed(extras.getOrDefault( key: "breed",  defaultValue: "Unknown"))
                .setCanFly(Boolean.parseBoolean(extras.getOrDefault( key: "canFly",  defaultValue: "true")))
                .build();
        case "lizard" -> new LizardBuilder()
                .setName(name)
                .setAge(age)
                .setBreed(extras.getOrDefault( key: "breed",  defaultValue: "Unknown"))
                .setVenomous(Boolean.parseBoolean(extras.getOrDefault( key: "venomous",  defaultValue: "false")))
                .build();
        default -> throw new IllegalArgumentException("Unsupported animal type: " + type);
    };

    addAnimal(animal);
}
```

Three-Tier Architecture

- Presentation layer (UI)
  - FXML: JavaFX UI layout
  - CLI: test-based interactions
  - Menus: Admin, Vet,Volunteer

- Business Logic Layer
  - Controllers: for views, for objects, for processes
  - Services: vaccination, animal registration
  - Builders: controlled Object creation
  - Observer pattern: event handling (to be extended)

- Database layer
  - DAO: basic CRUD (to be extended)
  - Repositories: for DB (to be extended)



5. Testing

- Performance time test for startup time

```java
public class AppStartupPerformanceTest {  Dang-Huynh

    @Test  Dang-Huynh
    public void testAppStartupTime() {
        long startTime = System.currentTimeMillis();

        ShelterApp app = ShelterApp.getInstance();
        app.startSilently(); // Simulated non-interactive startup

        long endTime = System.currentTimeMillis();
        long duration = endTime - startTime;

        System.out.println("App startup took " + duration + " ms");
        assertTrue( condition: duration < 2000,  message: "App startup took too long!");
    }
}
```

- JUnit tests for logic heavy controller

```java
@Test  Ana Tara
public void testAddAnimal() {
    AnimalRegistry registry = new AnimalRegistry();
    Animal dog = new Dog( name: "Rex",  age: 5,  breed: "Pug",  isTrained: false);

    registry.addAnimal(dog);

    assertEquals( expected: 1, registry.getAllAnimals().size());
    assertEquals( expected: "Rex", registry.getAllAnimals().get(0).getName());
}

@Test  Ana Tara
public void testSearchByName() {
    AnimalRegistry registry = new AnimalRegistry();
    registry.addAnimal(new Cat( name: "Mittens",  age: 2,  breed: "short",  furLength: "Siamese",  isIndoor: true));
    registry.addAnimal(new Cat( name: "Whiskers",  age: 3,  breed: "long",  furLength: "Persian",  isIndoor: true));


    var results = registry.searchByName("mitt");
    assertFalse(results.isEmpty());
    assertEquals( expected: "Mittens", results.get(0).getName());
}

@Test  Ana Tara
public void testRemoveAnimalById() {
    AnimalRegistry registry = new AnimalRegistry();
    Animal cat = new Cat( name: "Mimi",  age: 2,  breed: "Siamese",  furLength: "short",  isIndoor: true);
    registry.addAnimal(cat);

    boolean removed = registry.removeAnimalById(cat.getId());
```

Timeline

Rebuilding Basic Classes

- project started from scratch
- reused UML

Choosing Design Patterns

- team discussions
- online resources
- tutorials

Establishing MVC structure

- accent on modularity
- threads
- streams

Elaborating CLI/JavaFX UI

- three branches: master, CLI, JavaFX GUI

Testing

- JUnit tests
- Mockito

6. Future Improvements

- more threads once DB is connected (parallelism)
- better use of design patterns (Bridge!)
- elaborate Volunteer metho (track tasks, mark as done, delegate)
- implement Scheduling
- implement sessions (Admin, Vet, Volunteer)
- refine DAO (PowerBI attempt)
- finalize logging
- finalize JavaFX UI

Summary

- Animal Shelter app for employee use
  - Supports multiple user roles
- Streamlines animal adoption, returns, care
  - Considers adopter preferences for optimal pairing
  - Animal of the Month feature
- Extensible, organized, user-friendly

Thank you for listening!