**SDAT AND DEVOPS COMBINED QAP 1**

**Dog Daycare Documentation**

**Date: Sept 30/2025**

**By: Justin Greenslade**

1. **Project Overview:**

The Dog Daycare project is a simple Java application designed to manage the check-in and check-out of dogs. It uses object oriented principles and consists of the following main components:

* **Dog** – represents each dog, including name, breed, age, vaccination status, and owner.
* **Owner** – represents a dog’s owner, including name and phone number.
* **DaycareService** – manages the check-in/check-out of dogs.

The Dog Daycare system ensures that:

* Only vaccinated dogs can be checked in.
* The daycare cannot exceed 30 dogs at once.
* Attendance lists are updated dynamically.
* Duplicate check-ins are prevented.

**2. Clean Code Practices**

Throughout the project, I used implemented clean code practices. Here are some examples from my project:

**Meaningful names for classes, methods, and variables**

Below we see meaningful naming like:

* checkInDog()



* isEligibleForCheckIn()



* getCheckedInDogs()



**Single purpose methods**

Below we see single purpose methods like:

* checkInDog() handles all logic for checking a dog in and returns a boolean.
* checkOutDog() handles all checkout logic separately.

A screenshot of a computer

AI-generated content may be incorrect.

**Clear structure and formatting**

Below we proper structure and formatting:

* consistent indentation
* comments explaining key logic
* separation of model and classes

A screenshot of a computer

AI-generated content may be incorrect.A computer screen shot of a program

AI-generated content may be incorrect.

**3. Unit Testing Documentation**

Unit tests are written using **JUnit 5**, covering both **positive** and **negative** scenarios.

**DogTest**

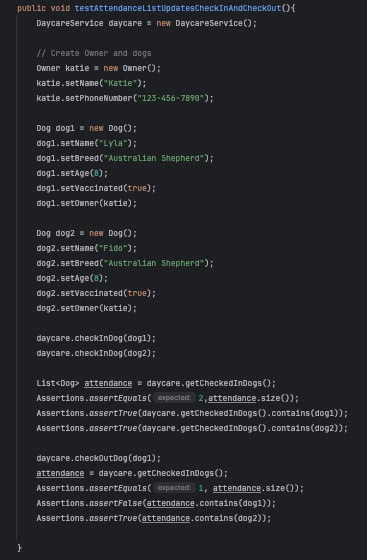
* **Positive tests:** Ensure correct getters/setters work, dog is linked to the right owner, and eligible dogs can be checked in.
* **Negative tests:** Ensure unvaccinated dogs are not eligible, and changing properties updates correctly.

**OwnerTest**

* **Positive tests:** Ensure Owner getters/setters work, and owner is linked to dogs correctly.
* **Negative tests:** Ensure wrong owner is not linked to dog.

**DaycareServiceTest**

* **Positive tests:**
  + testCheckDogInSuccessfully() – a vaccinated dog is checked in correctly.
  + testCheckDogOutSuccessfully() – a dog is checked out correctly.
  + testAttendanceListUpdatesCheckInAndCheckOut() – verifies attendance updates dynamically.
* **Negative tests:**
  + testCannotCheckInDogAtMaxCapacity() – cannot check in when at max capacity (30 dogs).
  + testCannotCheckInUnvaccinatedDog() – prevents unvaccinated dogs from checking in.



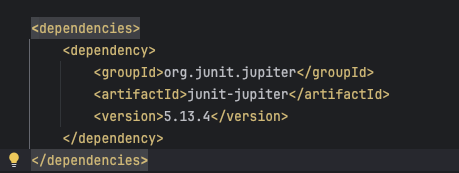
**4. Dependency List**

The project uses Maven to manage our dependencies. Dependencies are external libraries or tools that out project relies on to add functionality without us having to write everything from scratch. Specifically in this case we required Junit. Junit 5 is a testing framework for java which offers:

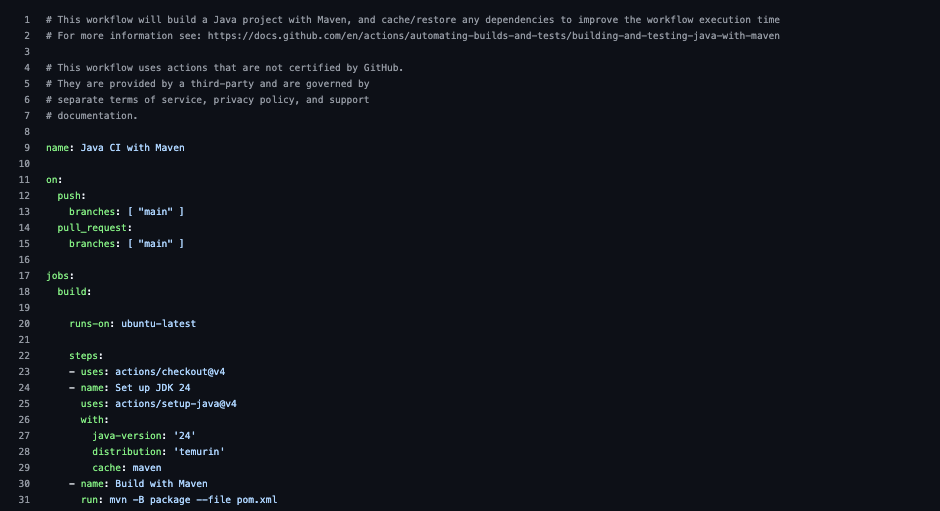
* Annotations like @Test to mark our test methods
* Assertions like Assertions.assertTrue, Assertions.assertFalse, Assertions.assertEquals so we can check for expected vs actual outcomes.
* It also has compatibility with GitHub actions so our tests run automatically whenever code is pushed or merged.

We needed the Junit dependency for this project to properly be able to test out project and verify expected outcomes. As well using Junit allowed the automated tests to run via GitHub actions, ensuring code functionality of my project behaved as intended.

Junit 5 Version 5.13.4 and was uptrained from: <https://central.sonatype.com/artifact/org.junit.jupiter/junit-jupiter/5.13.4>



**5. GitHub Actions**

For this project I set up a GitHub Actions to automatically run tests and build the project whenever code is pushed or a pull request is made. Ensuring that code cannot be merged unless tests pass  
A screenshot of a computer

AI-generated content may be incorrect.

**6.. Evidence of Dev/Trunk Workflow**

* Branching was implemented to preform work on.
* Pull requests (PRs) are merged into main after code review.
* Regular commits with meaningful messages demonstrate ongoing development.

A screenshot of a phone

AI-generated content may be incorrect.  
A black background with white lines

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

**7. Extra Features Added Beyond Basic Requirements**

Beyond the minimum requirement of simply checking dogs in and out, the project includes both additional functionality and unit tests to check the new features:

* **Max capacity validation:** ensures no more than 30 dogs are checked in simultaneously.
* **Vaccination eligibility enforcement:** prevents unvaccinated dogs from being checked in.
* **Duplicate check-in prevention:** ensures the same dog cannot be checked in twice.
* **Attendance tracking:** dynamically updates the list of checked-in dogs when dogs check out.

These added features/test improve the system and ensures daycare rules are properly followed.

**8. Problems / Issues Encountered**

* No major issues with coding or tests. Mainly implemented improvements as I went though the code. Making things more efficient as I could.
* Practiced clean code by adding a clear separation between model and service classes. It took a little extra attention to maintain clean code practices, and overall decide how I would section up the project