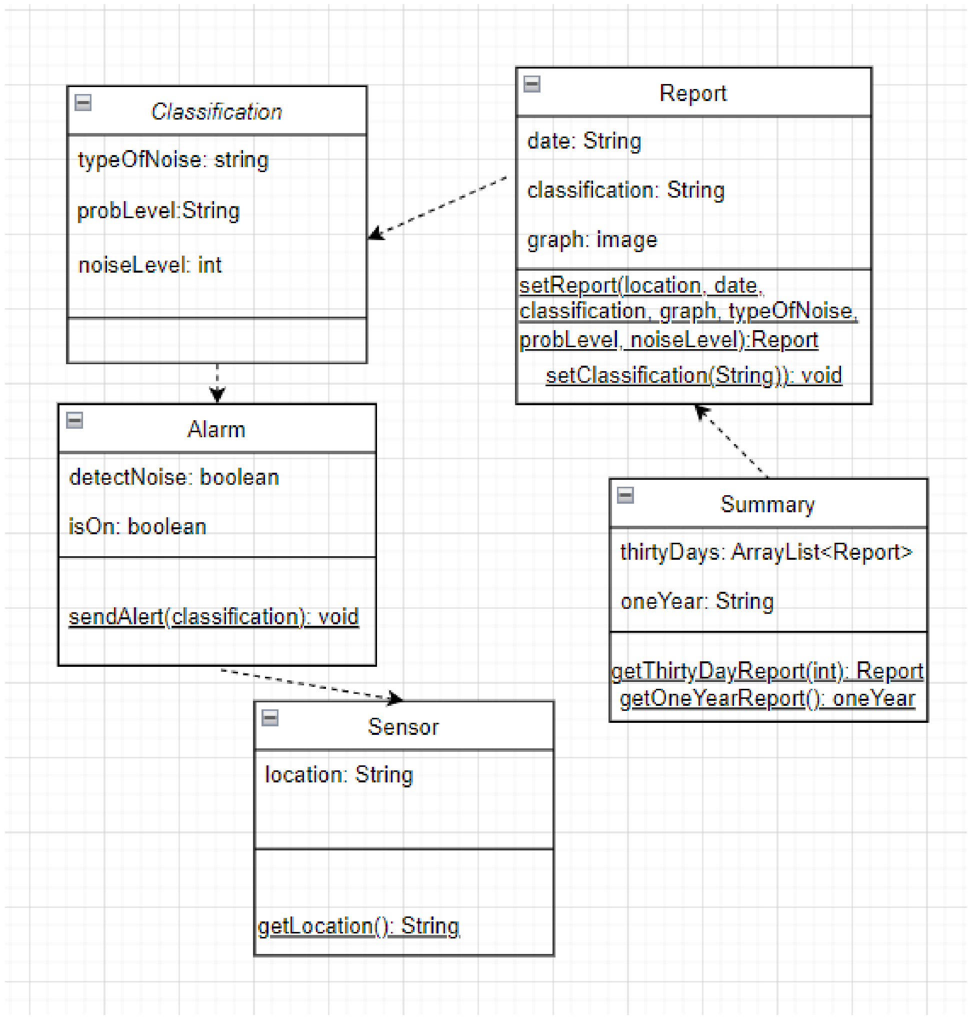
SDS: Test Plan

Name:
Aaron S.
Diego L.
Antonio T.

1. Software Design Specification

a. 
Classification
typeOfNoise: string
probLevel:String
noiseLevel: int
Report
date: String
classification: String
graph: image
setReport(location, date, classification, graph, typeOfNoise, probLevel, noiseLevel):Report
setClassification(String)): void
Alarm
detectNoise: boolean
isOn: boolean
sendAlert(classification): void
Summary
thirtyDays: ArrayList<Report>
oneYear: String
getThirtyDayReport(int): Report
getOneYearReport(): oneYear
Sensor
location: String
getLocation(): String

Key:

Class

Attributes

<u>Operations</u>

i. Sensor

1. *Location: String* - Holds the location of the specified alarm sensor.
2. <u>getLocation():String</u> - Operation gets the location of the specified alarm sensor.

ii. Alarm - uses Sensor
    1. *detectNoise: boolean* - Alarms will detect noise; if noise is detected detectNoise becomes true, otherwise it will return false
    2. *isOn: boolean* - If the alarm is on this will return true; otherwise will be false until a worker turns it back on after an alarm has been triggered
    3. <u>sendAlert(classification)</u> - When a noise is detected the alarm will use the Classification class to classify and send an alert to the control program. This will allow the creation of a report using information taken by the alarm.

iii. Classification - uses Alarm class
    1. *typeOfNoise: String* - holds the type of noise taken from the alarm
    2. *probLevel: String* - holds the probability level that the noise from the alarm is a mountain lion.
    3. *noiseLevel: String* - holds the noise level that was taken from the alarm

iv. Report
    1. *date: String* - holds current date
    2. *classification: String* - holds classification of the noise
    3. *graph: image* - holds graph of general location where noise was heard from
    4. <u>setClassification(classification): void</u> - Operation sets the classification of the noise (being of mountain lion origin or not)
    5. <u>createReport(location, date, classification, graph, typeOfNoise, probLevel, noiseLevel): Repor</u>t - creates a Report based on location, date, classification, graph, typeOfNoise, probLevel, and noiseLevel.

v. Summary
    1. *thirtyDays: ArrayList<Report>* - array list holds Reports from the last 30 days
    2. *oneYear: String* - holds all reports in a one year report
    3. <u>getThirtyDayReport(int): Report</u> - Operation gets specified report from 30 day report array list
    4. <u>getOneYearReport(): oneYear</u> - Operation gets the one year report

b.

2. <u>Verification Test Plan</u>

a. <u>Unit Tests</u>

i. **getLocation():**
    1. <u>Valid Location:</u>

            a. Test the method with a valid location input and ensure it returns the expected location.
        2. <u>Empty Location:</u>
            a. Verify that the method can handle an empty location string and returns an empty string as expected.
        3. <u>Location with Special Characters:</u>
            a. Test the method with a location that includes special characters (e.g., apostrophes, hyphens) and confirm it correctly returns the location with special characters.
        4. <u>Location with Leading and Trailing Whitespace:</u>
            a. Ensure that the method trims leading and trailing whitespace from the location input and returns the trimmed location.
        5. <u>Null Location:</u>
            a. Test the method with a null location input and verify that it appropriately handles null values and returns null.

ii. **noiseLevel():**

1. Valid Noise Level:
    a. Test the method with a valid noise level (e.g., "Loud") and ensure it returns the expected noise level.
2. Empty Noise Level:
    a. Verify that the method can handle an empty noise level string and returns an empty string as expected.
3. Noise Level with Special Characters:
    a. Test the method with a noise level that includes special characters and confirm it correctly returns the noise level with special characters.
4. Noise Level with Leading and Trailing Whitespace:
    a. Ensure that the method trims leading and trailing whitespace from the noise level input (e.g., " Quiet ") and returns the trimmed noise level ("Quiet").
5. Null Noise Level:
    a. Test the method with a null noise level input and verify that it appropriately handles null values and returns null.

b. Integration Test

i. **setReport(...):Report**

1. Test when all variables are valid
    a. Verifies that the creation of a report is created accordingly when it has all the needed information.
    b. Example: All information is real and verifiable.

        c. Expected output: A report with all its information in place.
    2. <u>Test when all variables are non existent</u>
        a. Checks how well it is able to handle when null variables are passed to the method.
        b. Example: The strings are empty, or the image is corrupted.
        c. Expected output: Do NOT create a report.
    3. <u>Test when all variables are invalid</u>
        a. Checks that method will act accordingly to having variables that are not accepted.
        b. Example: All information is not real. I.e. the string location is not real.
        c. Expected output: Do NOT create a report.
    4. <u>Test when some variables are non existent</u>
        a. If only one or a couple of variables are null the program will handle it.
        b. Example: We are missing the location string.
        c. Expected output: create the report with "NO EXISTING INFORMATION" in the non existing fields
    5. <u>Test when some variables are invalid</u>
        a. If only one or a couple of variables are not capable of being processed by the method it should handle it.
        b. Example: Data is not reasonable/possible
        c. Expected output: create the report with "NO VALID INFORMATION" in the invalid fields.

ii. **setClassification(classification): void**
    1. <u>Test when classification is valid</u>
        a. Is able to set classification for Report because classification is a verifiable object.
        b. Expected output: The classification is set for report.
    2. <u>Test when classification is invalid</u>
        a. The classification is not possible or does not correspond to the rest of the report's data. Program will be able to handle it
        b. Expected output: The classification in the report is set with an invalid classification object.
    3. <u>Test when classification is non existent</u>
        a. Method will handle when there is no classification found, or it's empty.
        b. Expected output: The method returns an error: " No classification passed".

c. **System Test**

   i. **Check if alarms are triggered for different types of mountain lion sounds as well as subsequent classifications. <u>Target is classification, typeOfNoise, probLevel, and noiseLevel in the report.</u>**

      1. Play mountain lion sounds
         a. Verify alarm is triggered and classification is correct when a high-probability mountain lion sound is played.
      2. Play non-mountain lion sound
         a. Verify classification is correct when on-mountain lion sound is played.
      3. Play mixed sound of mountain lion and non-mountain lion
         a. Verify alarm is triggered and classification is correct when mixed sounds are played.
      4. Play sounds at varying levels
         a. Verify probability level is accurately portrayed on report when previous tests are played at varying levels.
      5. Play with continuous sounds
         a. Test report classification for when continuous sounds are played, such as repetitive mountain lion sounds and is able to detect each sound.
      6. Play sounds with environmental challenges (background noise/interference)
         a. Verify alarm is triggered and correctly identified while environmental challenges are played for previous tests.

   ii. **getOneYearReport(): oneYear**

      <u>The target is testing the getOneYearReport() operation's output.</u>

      1. Empty report set
         a. Testing the command for when there may be no reports, or reports are empty. This is important to make sure the command is able to handle no reports being filed. Expected output is a successful handling of the operation
      2. Single report for the year
         a. Testing the command for when there may only be one report. Expected output is a successful call to the operation.
      3. Different report types
         a. Testing the command's output for reports of different types of data, including mountain lion and non-mountain lion reports. Expected output is a successful call to the operation.

4. Reports with diverse data
    a. Testing command when reports have varying data. Expected output is a successful call to the operation.
5. Report with different formats
    a. Testing the command’s output when a report may be missing information, such as location, noiseLevel, etc. Also for different formats of dates or graphs. Expected output is a successful call to the operation.
6. Test with reports that contain large amounts of data
    a. Testing for when printing large amounts of report data. Expected output is a successful call to the operation as it handles any overflow exception.
7. Test edge cases
    a. Testing command’s edge cases, such as leap years and daylight saving time changes to check operation’s handling of date and time.
    b. Also test for when a report was just made and for a report made 364/365 days ago.Expected output is a successful call to the operation as it handles date and time.
8. Test for missing data
    a. Test command for potential errors, such as missing or corrupted data. Expected output is a successful error handling saying that the file is corrupted.