Venn Backend Solution Writeup

I've finished solving the problem successfully.

I ran this program on the supplied input and compared the output generated with the supplied reference output text file. The output from this program and the reference output are both the same.

How do I run this program myself?

1. Open the project in IntelliJ
2. Run the script from the root of the project(This script runs a postgres docker necessary for the app):

```
./scripts/runPostgres.sh
```

3. Inside IntelliJ run the Spring boot app

How does this program work?

1. This is a Spring Boot application
2. It uses Postgres db as its storage
3. It reads the input text file and processes input line by line
4. The output of the program is by default written to "output/output.txt" file in the project root.
5. There is a schema.sql file in the resources dir which is used to create the schema in postgres.

How does this program get its input and write its output?

1. This program expects an input text file as input and an output text file as output.
2. Both input and output files are configurable(through application.properties).
3. Description of io files directories:
    1. testInput
       This is where the input files live.
    2. output
       This is where the output file lives by default
    3. referenceOutput
       This is where the referenceOutput files live, that are used for comparison when running a test.

Are there tests in this project?

1. Yes, there are 3 tests in this project.
2. All tests are written with testcontainer postgres container.
3. So each test runs a temporary postgres db container and then the spring boot app connects to that.
4. Test name:

    1. CompareWithProblemOutputTest.java
       This is the main test that uses the supplied input to run the program, then compares the program output with the reference output supplied with the problem statement. This test will fail if the 2 outputs don't match.

    2. InputWithUnparseableEntriesTest.java
       This test adds bunch of unparseable input lines to the input. The program ignores any unparseable input line that it encounters and still gives a correct output.

    3. TooManyLoadsPerDayTest.java
       This test exercises the path where too many load events per day are attempted to be added. The system will not process any load beyond the limit of 3 loads per day.