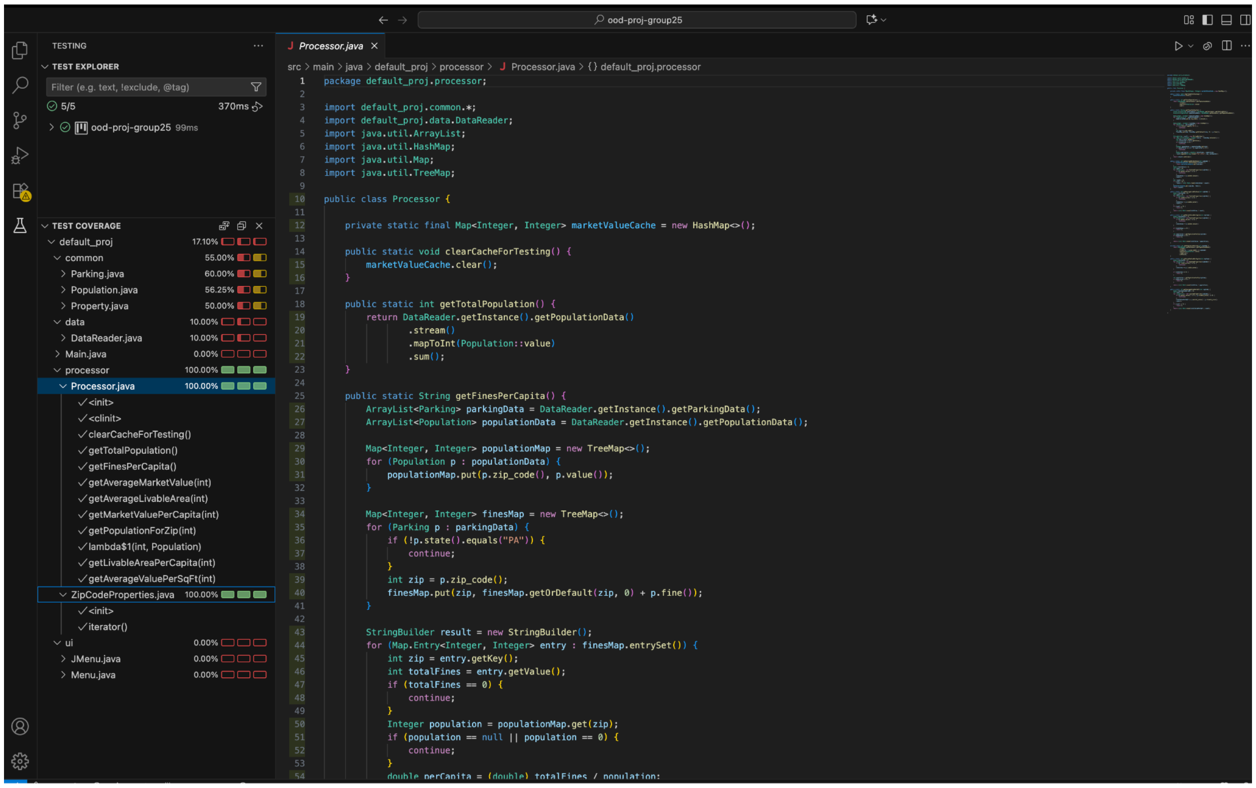
Report

Statement Coverage Screenshots:


ood-proj-group25
TESTING
TEST EXPLORER
Filter (e.g. text, !exclude, @tag)
Processor.java
src > main > java > default_proj > processor > Processor.java > {} default_proj.processor

```
5/5                                370ms
ood-proj-group25 99ms

TEST COVERAGE
default_proj                 17.10%
  common                     55.00%
    Parking.java             60.00%
    Population.java          56.25%
    Property.java            50.00%
  data                       10.00%
    DataReader.java          10.00%
  Main.java                   0.00%
  processor                 100.00%
    Processor.java          100.00%
      ✓<init>
      ✓<clinit>
      ✓clearCacheForTesting()
      ✓getTotalPopulation()
      ✓getFinesPerCapita()
      ✓getAverageMarketValue(int)
      ✓getAverageLivableArea(int)
      ✓getMarketValuePerCapita(int)
      ✓getPopulationForZip(int)
      ✓lambda$1(int, Population)
      ✓getLivableAreaPerCapita(int)
      ✓getAverageValuePerSqFt(int)
    ZipCodeProperties.java  100.00%
      ✓<init>
      ✓iterator()
  ui                          0.00%
    JMenu.java                0.00%
    Menu.java                 0.00%
```

```java
package default_proj.processor;

import default_proj.common.*;
import default_proj.data.DataReader;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.TreeMap;

public class Processor {

    private static final Map<Integer, Integer> marketValueCache = new HashMap<>();

    public static void clearCacheForTesting() {
        marketValueCache.clear();
    }

    public static int getTotalPopulation() {
        return DataReader.getInstance().getPopulationData()
                .stream()
                .mapToInt(Population::value)
                .sum();
    }

    public static String getFinesPerCapita() {
        ArrayList<Parking> parkingData = DataReader.getInstance().getParkingData();
        ArrayList<Population> populationData = DataReader.getInstance().getPopulationData();

        Map<Integer, Integer> populationMap = new TreeMap<>();
        for (Population p : populationData) {
            populationMap.put(p.zip_code(), p.value());
        }

        Map<Integer, Integer> finesMap = new TreeMap<>();
        for (Parking p : parkingData) {
            if (!p.state().equals("PA")) {
                continue;
            }
            int zip = p.zip_code();
            finesMap.put(zip, finesMap.getOrDefault(zip, 0) + p.fine());
        }

        StringBuilder result = new StringBuilder();
        for (Map.Entry<Integer, Integer> entry : finesMap.entrySet()) {
            int zip = entry.getKey();
            int totalFines = entry.getValue();
            if (totalFines == 0) {
                continue;
            }
            Integer population = populationMap.get(zip);
            if (population == null || population == 0) {
                continue;
            }
            double perCapita = (double) totalFines / population;
```

Design Pattern Usage:

- ❖ **Singleton Pattern**: Implemented in default_proj.data.DataReader via the getInstance() method. Ensures only one instance of the data reader exists to load large datasets into memory just once, preventing redundant file I/O operations.

- ❖ **Iterator Pattern**: Implemented in default_proj.processor.ZipCodeProperties, which implements Iterable<Property>. The iterator() method allows the Processor class to iterate exclusively over properties matching a specific ZIP code without exposing the underlying collection logic, used in methods like getAverageMarketValue() and getAverageLivableArea().

Java Feature Usage:

Generics:

Class: default_proj.processor.ZipCodeProperties
Line: 8
Use: implements Iterable<Property> **-** Creates a generic iterable collection that guarantees it will only yield Property objects when iterated, enabling type-safe for-each loops without casting.

Class: default_proj.processor.ZipCodeProperties
Line: 9
Use: ArrayList<Property> filteredProperties - Type-safe list that can only hold Property objects, preventing runtime ClassCastException errors.

Class: default_proj.processor.ZipCodeProperties
Line: 21
Use: Iterator<Property> - Generic iterator return type that enforces type safety, ensuring the iterator's next() method always returns a Property without requiring explicit casting.

Class: default_proj.processor.Processor
Line: 44
Use: Map.Entry<Integer, Integer> - Generic map entry type used when iterating over a map, providing compile-time type checking for both key and value types.

Streams + Lambdas:

Class: default_proj.processor.Processor
Line: 19-22
Use: Stream pipeline with method reference Population::value to sum all population values. The .stream() creates a stream from the collection, .mapToInt(Population::value) applies the method reference to extract integer values, and .sum() aggregates them into a total.

Class: default_proj.processor.Processor
Line: 119-124
Use: Stream pipeline combining a lambda expression p -> p.zip_code() == zipCode to filter populations by ZIP code, and a method reference Population::value to extract the population value. The .findFirst().orElse(0) safely returns 0 if no matching ZIP code is found, demonstrating functional-style data processing.

Memoization Usage:

Class: default_proj.processor.Processor
Method: getAverageMarketValue(int zipCode)
Implementation: Uses a static HashMap<Integer, Integer> named marketValueCache (line 12) to store previously computed results. Before calculating the average market value for a ZIP code, the method checks if the result already exists in the cache (line 61). If found, it immediately returns the cached value (line 62), avoiding the expensive iteration through the property records. If not found, it performs the calculation, stores the result in the cache (line 77), and returns it. This optimization significantly improves performance when the same ZIP code is queried multiple times during program execution.

Group Member Contributions:

- Grayson: I completed the GenAI analysis and implemented the classes in the Data and Common tiers.

- Joe: I completed the Documentation and implemented the classes in the UI tier.
- Tesfalem: I completed the processing tier and the testing for the processing tier, in addition to the report.