**ACS 560 Software Engineering**

**HW1-Programming**

**Name**: Vijayagiridharan Subramanian

### **Project Overview**

**Purpose:** This project analyzes a restaurant menu from a CSV file to gain insights into the menu items.

**Key Components:**

1. **CSVReader Class:**
   * Reads data from a CSV file containing menu items.
   * Converts each row of the CSV into a MenuItem object.
2. **MenuItem Class:**
   * Represents a single menu item with details like name, price, calories, preparation time, and availability.
3. **DataAnalyzer Class:**
   * Takes a list of MenuItem objects and performs various analyses:
     + **Count Items Per Category:** Shows how many items are in each category.
     + **Average Price Per Category:** Calculates the average price for each category.
     + **Most and Least Expensive Items:** Identifies the most and least expensive items on the menu.
     + **Average Preparation Time:** Computes the average preparation time for vegetarian and non-vegetarian items.
     + **Availability Counts:** Counts how many items are available and unavailable.
     + **Highest Calorie Item:** Finds the item with the highest number of calories.
4. **Main Class:**
   * Executes the project by:
     + Reading the CSV file.
     + Analyzing the data using DataAnalyzer.
     + Printing out the results of various analyses.

**Working:**

**1. Reading CSV File Data:**

The `CSVReader` class is responsible for reading the CSV file. It uses a `BufferedReader` for efficient reading of the file line by line. The class reads each line, splits the data using a comma delimiter, and then parses it into a `MenuItem` object. This approach follows best practices for reading large files by minimizing memory usage and handling potential exceptions using a try-with-resources statement to ensure that the file is properly closed after reading.

**Best Practices:**

- BufferedReader: Efficiently reads large files by buffering input, reducing the number of I/O operations.

- Try-with-resources: Ensures that resources are closed automatically, preventing resource leaks.

- Data Parsing: Each line is split and parsed into appropriate data types (e.g., integers, doubles, booleans), ensuring that the data is handled correctly.

**2. Appropriate Data Structure Selection:**

The data from the CSV file is stored in a `List<MenuItem>`, where `MenuItem` is a class representing each row in the CSV. This data structure is selected because it allows for dynamic resizing, easy iteration, and efficient access by index. The `List` interface is appropriate here as it allows the program to store an arbitrary number of menu items, which can be easily manipulated (e.g., sorted, filtered) during analysis.

**Optimization Considerations:**

- List Interface: Provides flexibility in handling a dynamic number of records.

- Custom Class (`MenuItem`): Encapsulates data fields specific to each menu item, making the code more readable and maintainable.

**3. Text-based Report of Analyzed Data:**

The `DataAnalyzer` class performs various analyses on the data and prints the results to the console. This includes counting items per category, calculating average prices, identifying the most and least expensive items, and more. The results are presented in a text-based format, which is easy to read and share.

**Best Practices:**

- Separation of Concerns: The `DataAnalyzer` class is focused solely on analysis, keeping the code modular and maintainable.

- Efficient Processing: Uses Java’s `Collections` utility methods (e.g., `max`, `min`) and streams for concise and efficient data processing.

- Clear Output: Results are clearly formatted and printed, making the report easy to understand.

**4. Code Standards Adherence:**

The code adheres to standard Java coding conventions, including:

- Naming Conventions Class names are capitalized (`Main`, `CSVReader`, `DataAnalyzer`, `MenuItem`), and variables/methods use camelCase.

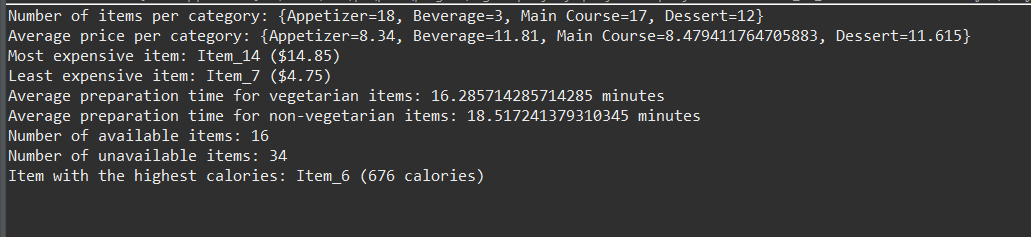
- Encapsulation: The `MenuItem` class encapsulates all related properties, with private fields and public getters.

- Modularity: Code is divided into logical units (classes) that each handle a specific aspect of the functionality, making the program easier to understand and extend.

- Comments and Documentation: Each method and class are documented to explain their purpose and usage, improving code readability.

This structure ensures that the project not only meets the functional requirements but also adheres to industry-standard best practices, optimizing both performance and maintainability.

**Result:**

****