Here are some ideas for projects. If you want to do something else, you will need to get it approved. See the project specification for the deadline for that. If you are interested in a specific Java-related feature (like metaprogramming, for example) and need help coming up with an appropriate project for it, feel free to come see me.

Remember that these projects are not complete specifications for a reason. You should use these specs as a baseline and build off them to create a full, working application that is user-friendly and provides reasonable functionality given the context.

I. Gradebook

General Requirements.

Implement a gradebook system similar to a simplified version of the D2L gradebook. The system should allow for two different views – one for students, and one for teachers. The nice thing about this project is that you can use D2L as a template for what kinds of functionality to include.

Students should be able to:

- View their courses (completed and current).
- View assignments in a course (graded or ungraded).
- Calculate class average based on the graded assignments.
- Calculate GPA based on the final grades in all completed courses.

Teachers should be able to:

- View their courses (completed and current).
- Add and remove assignments to a course.
- Add and remove students from the course. (That's actually not done by instructors, but for simplicity, you can have it as an option.)
- Import a list of students to add to the course. From a text or csv file.
- View the students enrolled in a course.
- Add grades for students for an assignment.
- Calculate class averages and medians on assignments.
- Calculate a student's current average.
- Sort students by first name, last name, or username.
- Sort students by grades on an assignment.
- Put students in groups.
- Assign final grades (A, B, C, D, E) to students based on course averages.
- View ungraded assignments.
- Choose a mode for calculating class averages.*
- Set up categories of assignments with weights, allowing for dropped assignments.*

***Specifications for grading & assignments. These are general. If you need more details, just ask. Examples can be provided.**

- There should be two possible modes for calculating class averages, but only one can be used for a given course:

    - Option 1: Final Grade = Total Points Earned/Total Points Possible. Basically, all the points from all the assignments are added up. (This is how I do it in CSc 335.)
    - Option 2: The final grade is based on categories and percentages.
- In a course that uses categories, there should be the option to "drop" a certain number of assignments with the lowest grade and to assign weights to the different categories.

Ideas to Increase Complexity.

- In order to meet the expectations for complexity, you should consider adding additional functionality that would be useful for the users. The functionalities listed above are the basic requirements, but there may be other stuff that would help fill in gaps or make the UI work better.
- Your UI can be text-based or graphical, but keep in mind that a high-quality GUI that utilizes OBSERVER can be a way to increase the complexity of your project.
- Other ideas for increasing complexity:
    - Add functionality for users to login + additional security measures.
    - Add data persistence or use of databases.
    - Utilize design patterns like COMPOSITE or DECORATOR.

II. Restaurant

General Requirements.

Implement an app for managing orders in a restaurant from the perspective of a server.

- Assign tables to a server.
- Input orders for each person at a table.
- The app should include menu items, organized by different categories, such as entrees, drinks, desserts, etc. Costs should also be included.
- Some menu items may allow for modifications, and that should be implemented.
- Should include functionality for calculating:
    - the bill
    - splitting the bill
        - by evenly splitting it between all the people at the table
        - by individual orders
- Closing an order when the bill is paid – closed orders should still be maintained
- When a bill is paid, customers have the option of adding a tip and a server's tips should be tracked throughout their shift
- Implement functionality for managing multiple servers.
- Implement functionality for tracking sales of specific menu items.
    - Sort sales based on most frequently ordered items.
    - Sort sales based on the total amount of money made from a specific item – i.e. the cost per item * the number of items sold
- Determine the server who earned the most from tips.

Ideas to Increase Complexity:

- Implement a high-quality GUI using the OBSERVER pattern.
- Add additional functionality to the backend.

- Do more data analysis from the perspective of the restaurant owner – e.g. # of people, busiest times, etc.

III. Cribbage
Cribbage is a game that uses a regular deck of cards and a pegboard for keeping track of scores. Here is a link to game instructions: https://bicyclecards.com/how-to-play/cribbage

General Requirements.

- Implement a Cribbage game with the following options:
  - Multiple human players
  - Human vs. Computer
    - EASY mode – the Computer just discards randomly
    - HARD mode – the Computer evaluates and discards optimally based on a simple strategy
- Utilize the FLYWEIGHT design pattern.
- Utilize the STRATEGY design pattern for the Computer modes.
- Implement a way to track game wins/losses for individual users.

Ideas to Increase Complexity:

- Implement more complex Computer strategies.
- Implement additional metadata for the game – i.e. a leaderboard, track user data across all users, etc.
- Implement a nice GUI using OBSERVER.
- Use networks/sockets to simulate playing over the Internet.

IV. Yahtzee
Yahtzee is a game that uses Dice. Here's a link that explains how it works:
https://www.wikihow.com/Play-Yahtzee

General Requirements.

- Implement a Yahtzee game with the following options:
  - Multiple human players
  - Human vs. Computer
    - EASY mode: The computer randomly selects a scoring item.
    - HARD mode: The computer analyzes the dice rolls to select the best option.
- Utilize the FLYWEIGHT design pattern.
- Utilize the STRATEGY design pattern for the Computer modes.
- Implement a way to track game wins/losses for individual users.

Ideas to Increase Complexity:

- Implement more complex Computer strategies.
- Implement additional metadata for the game – i.e. a leaderboard, track user data across all users, etc.

- Implement a nice GUI using OBSERVER.
- Use networks/sockets to simulate playing over the Internet.

V. Modified Scrabble
Scrabble is a word game. Here are some youtube videos to explain how to play:
- https://www.youtube.com/watch?v=S8PZdXYBapI
- https://www.youtube.com/watch?v=K1KgvZwwJqo

Your version of the game can be modified to have no empty tiles, have only two players, and have only double and triple letter scores on the board.

General Requirements.
- Implement a Scrabble game for two human players.
- Utilize the FLYWEIGHT design pattern.
- Implement a way to track game wins/losses for individual users.

Ideas to Increase Complexity:
- Implement a Human vs. Computer option where the Computer uses a specific strategy for coming up with words.
- Implement additional metadata for the game – i.e. a leaderboard, track user data across all users, etc.
- Implement a nice GUI using OBSERVER.
- Use networks/sockets to simulate playing over the Internet.
- Add in the features from Scrabble that were removed from the basic requirements.