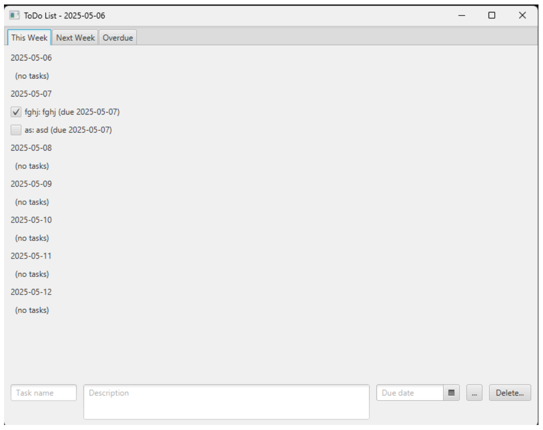
Demo:

View tasks in three tabs of This Week: tasks due over the next seven days, Next Week: tasks due in the week after that, and Overdue: tasks whose due‑date is before today. Add new tasks by entering a name, description, and due‑date. The due-date brings up a little calendar for you to pick out a date. You are able to mark tasks complete by clicking the checkbox next to each task. With the ability to delete all completed tasks with a single button amd you are able to persist tasks between runs via serialization to Object.dat.



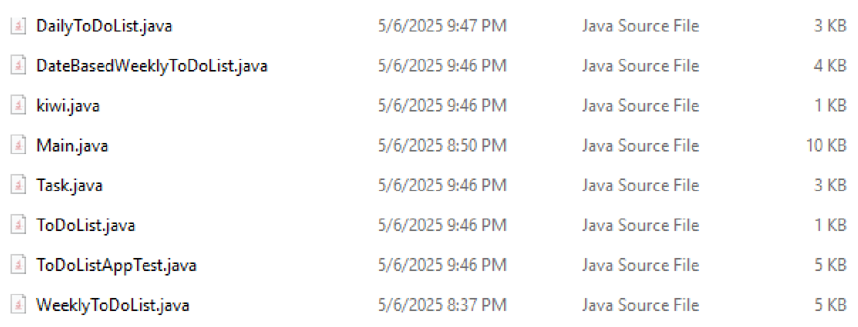
2025-05-06
(no tasks)
2025-05-07
✓ fghj: fghj (due 2025-05-07)
as: asd (due 2025-05-07)
2025-05-08
(no tasks)
2025-05-09
(no tasks)
2025-05-10
(no tasks)
2025-05-11
(no tasks)
2025-05-12
(no tasks)

Self Evaluation:

Size (~500 lines incl. comments) 5 / 5

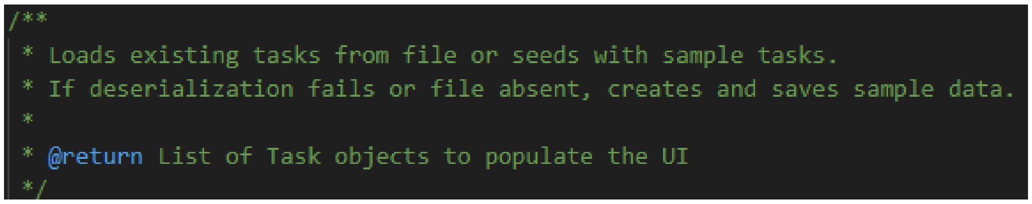
- There are definitely over 500 lines of code with 906 lines of code with comments and spacing. There's also not over 306 lines of spacing in my code either.

5+ Java files 2 / 2



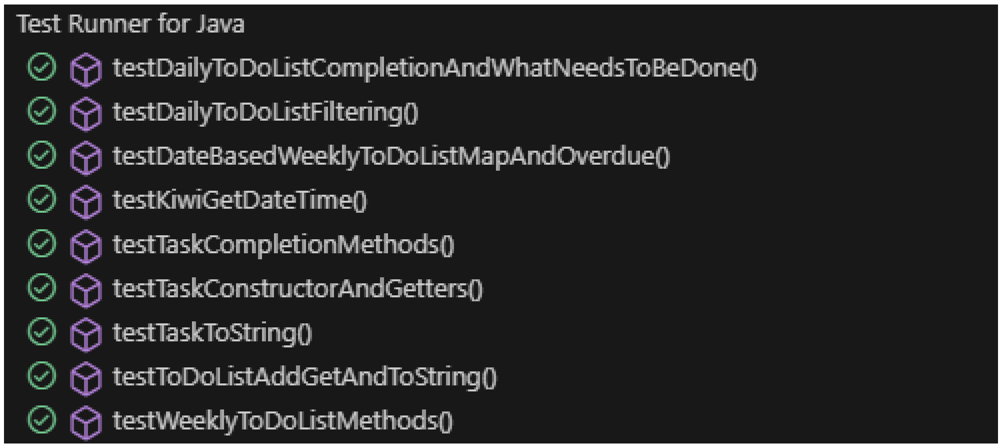
DailyToDoList.java	5/6/2025 9:47 PM	Java Source File	3 KB
DateBasedWeeklyToDoList.java	5/6/2025 9:46 PM	Java Source File	4 KB
kiwi.java	5/6/2025 9:46 PM	Java Source File	1 KB
Main.java	5/6/2025 8:50 PM	Java Source File	10 KB
Task.java	5/6/2025 9:46 PM	Java Source File	3 KB
ToDoList.java	5/6/2025 9:46 PM	Java Source File	1 KB
ToDoListAppTest.java	5/6/2025 9:46 PM	Java Source File	5 KB
WeeklyToDoList.java	5/6/2025 8:37 PM	Java Source File	5 KB

Javadoc comments for classes & methods 1 / 1



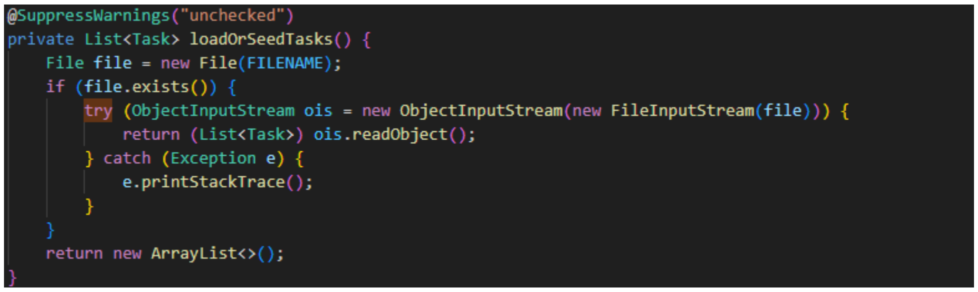
```
/**
 * Loads existing tasks from file or seeds with sample tasks.
 * If deserialization fails or file absent, creates and saves sample data.
 *
 * @return List of Task objects to populate the UI
 */
```

JUnit tests 1 / 1



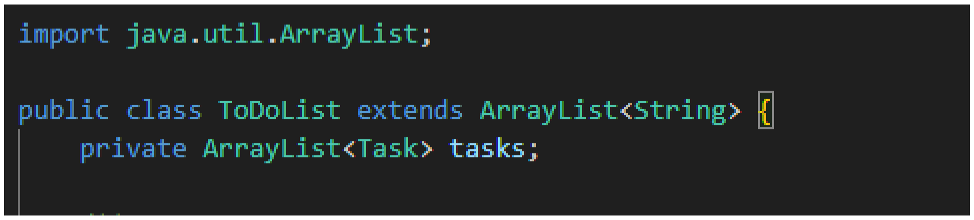
```
Test Runner for Java
testDailyToDoListCompletionAndWhatNeedsToBeDone()
testDailyToDoListFiltering()
testDateBasedWeeklyToDoListMapAndOverdue()
testKiwiGetDateTime()
testTaskCompletionMethods()
testTaskConstructorAndGetters()
testTaskToString()
testToDoListAddGetAndToString()
testWeeklyToDoListMethods()
```

Exception handling 1/1



```
@SuppressWarnings("unchecked")
private List<Task> loadOrSeedTasks() {
    File file = new File(FILENAME);
    if (file.exists()) {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(file))) {
            return (List<Task>) ois.readObject();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    return new ArrayList<>();
}
```

Interface/Extends usage

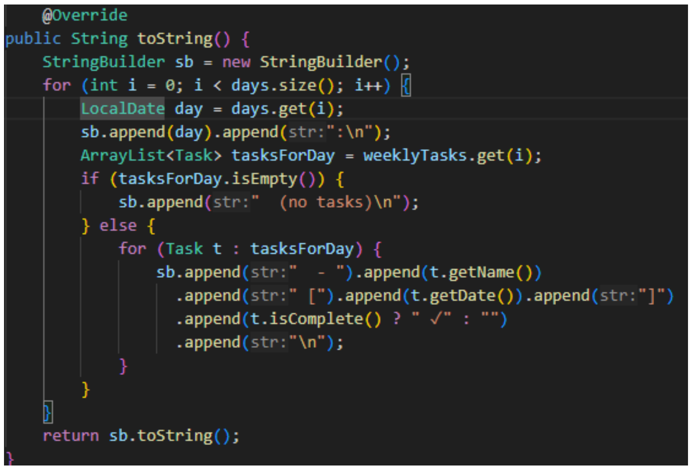


```
import java.util.ArrayList;

public class ToDoList extends ArrayList<String> {
    private ArrayList<Task> tasks;
```

Just want to clarify that I am using an external Interface aka the arraylist string for this credit

Inheritance & overriding 1/1



```
    @Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < days.size(); i++) {
        LocalDate day = days.get(i);
        sb.append(day).append(str:":\n");
        ArrayList<Task> tasksForDay = weeklyTasks.get(i);
        if (tasksForDay.isEmpty()) {
            sb.append(str:"  (no tasks)\n");
        } else {
            for (Task t : tasksForDay) {
                sb.append(str:"  - ").append(t.getName())
                  .append(str:" [").append(t.getDate()).append(str:"]")
                  .append(t.isComplete() ? " ✓" : "")
                  .append(str:"\n");
            }
        }
    }
    return sb.toString();
}
```

A little more insight that the weekly todo list

Self Evaluation:

This course states that "Students will learn OOP concepts such as classes, objects, encapsulation, messaging, data hiding, inheritance, and polymorphism." With class and objects I used the following like Task, ToDoList, DailyToDoList, and DateBasedWeeklyToDoList. The encapsulation keeps all the variables in class required in that section keeping the methods in that class in the same idea and realm of the area. I personally believe I achieved this really well and keep it all very versatile on how it can be used which is the main reason for it. I also achieve a lot of messaging that occurs throughout the entire system. I feel as though a lot of data hiding and classification go hand and hand. I am keeping the class and data separated with a lot of other important things as well. The inheritance is through the todolist system which covers this effectively. I effectively use Polymorphism in said todolist system to effective

I know in my proposal I had a grander ambition than what I achieved in this code. I planned on doing something else but ran into a lot of issues with the GUI element and incorporating local data time. However, I am very pleased in what I was able to achieve with this assignment as well. Also the Junit testing made me run into a bug with my code that I did not realize until writing the code up. That I had a major flaw with my code as a whole. This leads me to fixing the problem.