CSED332 Assignment 5

Due Saturday, Nov 9th

Objectives

- Learn Observer pattern
- Learn GUI programming in Java
- Learn IntelliJ Platform

Gradle

- Gradle is another build automation tool widely used for Java. In this project, we will use Gradle instead of Maven, because IntelliJ Platform mainly uses Gradle.
- Gradle uses a build script, named `gradle.build`, just like `pom.xml` for Maven. Gradle has various commands , such as `gradle compileJava` and `gradle test`.
- See the pages `https://docs.gradle.org/current/userguide/building_java_projects.html` and `https://gradle.org/guides/` for more information on Gradle.

Problem 1

- *Even/odd Sudoku* is a variant of Sudoku (`https://en.wikipedia.org/wiki/Sudoku`). The goal of even/odd Sudoku is to fill numbers from 1 to 9 in empty squares of a 9×9 grid such that
  - $1 \sim 9$ appear exactly once in each row, column and 3×3 box.
  - All grey squares contain even numbers, and all white squares contain odd numbers.
- An even/odd Sudoku puzzle can be implemented using the Observer pattern. The key classes are `Cell` and `Group`, where the groups observe their cells.
  - A cell has a set of possible numbers that the cell can have, and may have a value. A cell changes by getting or losing a value or possibilities.
  - There is a group for each row, column and 3×3 box. If one of the members of a group has a particular value, none of its other members can have the value as a possibility.
  - For example, in the unsolved puzzle of Figure 1, the first row of the second column has no value and the set of possibilities $\{1, 3, 5\}$.

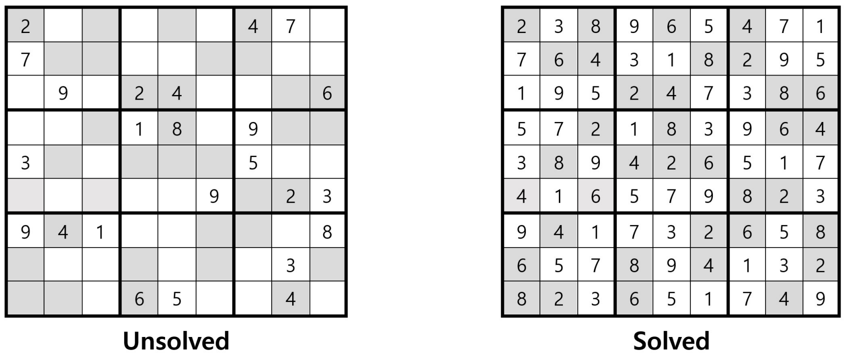


Figure 1: Even/odd Sudoku

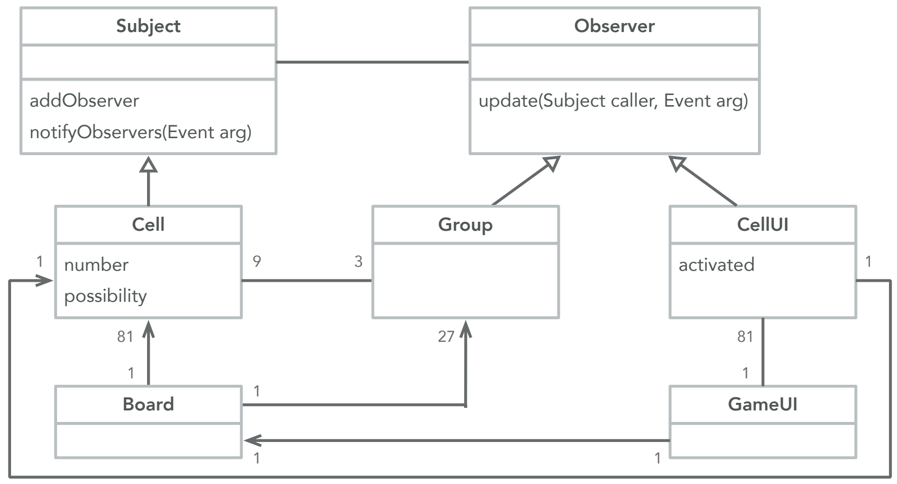


Figure 2: Class diagram for even/odd Sudoku

- Figure 2 shows the class diagram for this problem. The goal is to implement the five classes in the diagram: `Cell`, `Group`, `Board`, `CellUI`, and `GameUI`.
    - `Subject` and `Observer` are already implemented. A subject notifies an *event* to its observers. An event is an instance of `Event`, and provides additional information about changes.
    - A cell should notify appropriate events to its observers, when particular changes happen. For example, if a cell has lost all its possibilities, it will notify `DisabledEvent` to its observers.
    - A group "receives" an event when the value of its cell is set or unset (`SetNumberEvent` or `UnsetNumberEvent`), and accordingly changes the possibilities of the other cells in the group.
    - A board maintains 9 row groups, 9 column groups, 9 square groups, and 81 cells. The classes `Board`, `Group`, and `Cell` specify the object-oriented model.
- `GameUI` and `CellUI` implements a simple GUI, as shown in Fig. 3. The class `GameUI` defines the top-level container. `CellUI` observes a single cell and defines an interface for the cell.
    - If a number is written in an empty `CellUI`, it tries to update the value of the related cell. If successful, the number is retained in the `CellUI`; otherwise, the `CellUI` is emptied again.
    - If a cell loses all its possibilities (notified by `DisabedEvent`), because other cells in the same group are filled, the corresponding `CellUI` is *deactivated* and marked with red borders.
    - If a number is removed from `CellUI`, other cells in the same group may restore a possibility. If a deactivated cell gets a possibility (notified by `EnabledEvent`), the `CellUI` is *activated*.

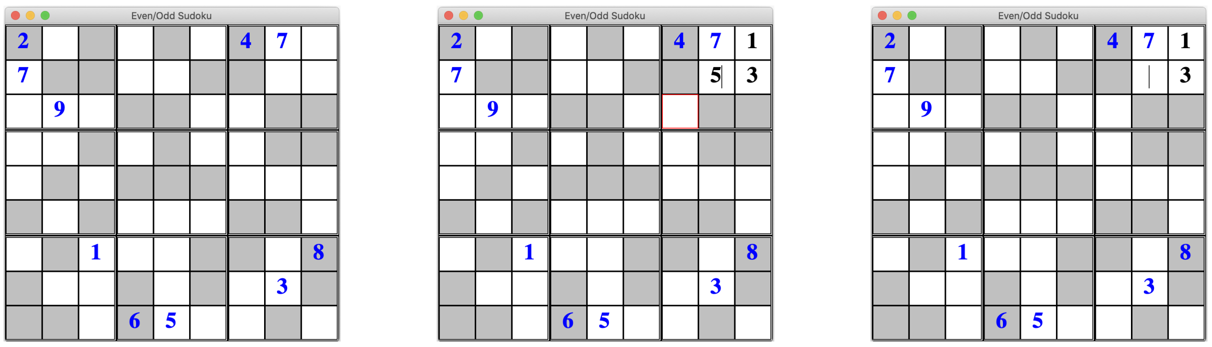


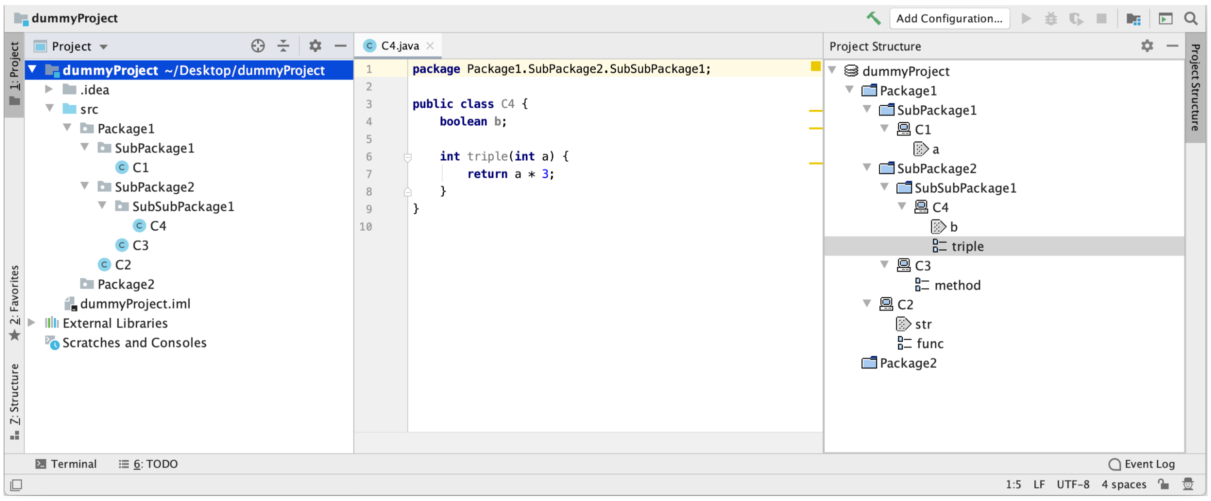
Figure 3: Even/odd Sudoku GUI

- General Instruction
  - The `src/main` directory contains the skeleton code. You should implement all classes and methods with *TODO* in the above classes.
  - The `src/test` directory contains some test methods for non-GUI classes in `BoardTest.java`. Your code will be graded by Gradle, using extra test cases written by teaching staff.
  - Your code must follow the Model-View-Controller architectural pattern. In particular, the model classes (`Board`, `Group`, and `Cell`) should *not* depend on GUI classes.
  - The command `gradle jar` will create a jar file in the `build/libs` directory, which can be executed using the command: `java -jar problem1-1.0-SNAPSHOT.jar`.
  - Do not modify the existing interfaces, the class names, and the signatures of the public methods. You can add private methods or member variables if you want.

- Java Swing References
  - Java Swing Tutorial
    `https://www.javatpoint.com/java-swing`
  - Using Swing Components
    `https://docs.oracle.com/javase/tutorial/uiswing/components/`
  - Laying Out Components Within a Container
    `https://docs.oracle.com/javase/tutorial/uiswing/layout/`
  - Writing Event Listeners
    `https://docs.oracle.com/javase/tutorial/uiswing/events/index.html`

Problem 2

- In this problem, you will implement the Project Structure plugin of IntelliJ IDEA that displays the summarized view of a java project.
  - The plugin displays a project as a tree structure in the order: Projects – Packages – Classes – Class members (fields and methods), as shown in Figure 4.
  - For example, `dummyProject` is a project, which includes package `Package1`, which in turn contains a class `C2`. The `C2` class has one method `func` and one field `str`.



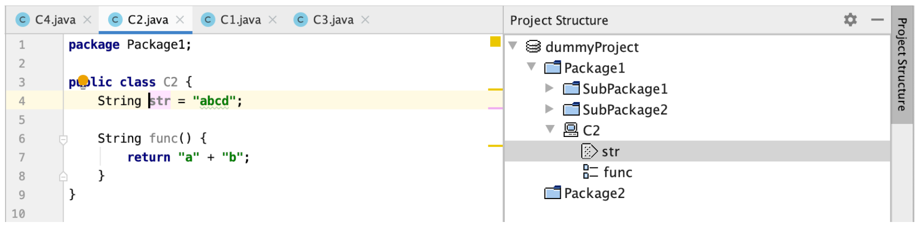
```
package Package1.SubPackage2.SubSubPackage1;

public class C4 {
    boolean b;

    int triple(int a) {
        return a * 3;
    }
}
```

Figure 4: Project Structure plugin (right)

- Each node of Project Structure is decorated with an icon that identifies the types of nodes (e.g., projects, packages, classes, etc), and interacts with the user as follows:
  - When you double-click a terminal node (e.g., fields and methods), the corresponding java file will be displayed in the editor, and the code of the selected item will be highlighted.

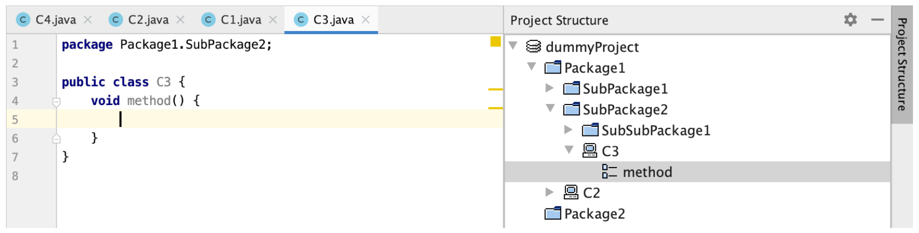


```
package Package1;

public class C2 {
    String str = "abcd";

    String func() {
        return "a" + "b";
    }
}
```

  - Project Structure observes the underlying project; i.e., whenever you change the project, the tree will be updated accordingly, and the corresponding node is shown in the GUI.



```
package Package1.SubPackage2;

public class C3 {
    void method() {

    }
}
```

  - When you double-click a nonterminal node, the item will toggle between expanded and collapse states. This is the default behavior of `Tree` in the IntelliJ platform.

- Figure 5 shows a class diagram of Project Structure. The goal is implement the two classes: `ProjectStructureTree` and `ProjectTreeModelFactory`.
  - `ProjectStructureTree` provides a GUI for our plugin. It is a subclass of `JTree`, which observes a tree structure given as an instance of `TreeModel`.
  - `ProjectTreeModelFactory` creates a tree model from a given project (an instance of `Project`). Note that `Project`, `TreeModel`, and `ProjectTreeModelFactory` are *not* GUI classes.
  - `ProjectStructureTree` also observes `Project`; whenever a change in the project is notified, it updates its tree model using `ProjectTreeModelFactory` and displays the changed node.
  - `ProjectStructureWindow` is a top-level GUI container of our plugin. `MyToolWindowFactory` creates `ProjectStructureWindow` for IntelliJ IDEA.

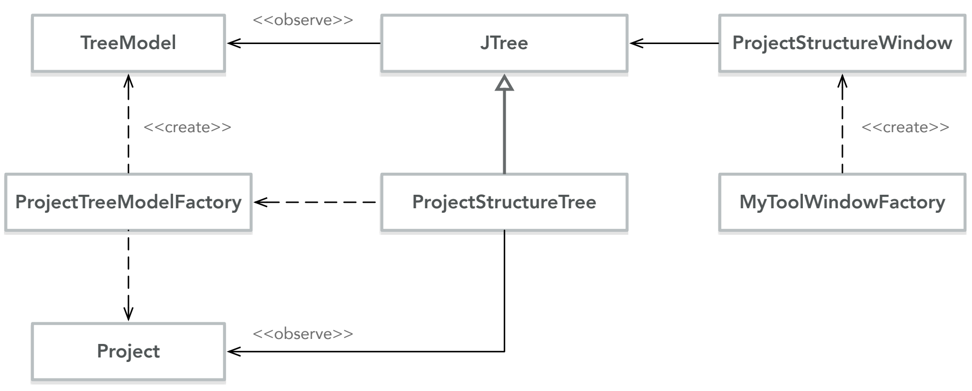


Figure 5: Class diagram of the Project Structure plugin

- General Instruction
  - The `src/main` directory contains the skeleton code. You should implement all classes and methods with *TODO* in the above classes.
  - The `src/test` directory includes some test methods, based on the IntelliJ Platform testing infrastructure (*not* JUnit 5). See the following link to learn about IntelliJ plugin testing:
    - `https://www.jetbrains.org/intellij/sdk/docs/basics/testing_plugins.html`
  - Again, your code must follow the Model-View-Controller architectural pattern. In particular, `ProjectTreeModelFactory` should *not* depend on GUI classes.
  - The command `gradle runide` will launch IntelliJ IDEA with your plugin. After opening a project, you can find PROJECT STRUCTURE on the right toolbar in the window.
  - Do not modify the existing interfaces, the class names, and the signatures of the public methods. You can add private methods or member variables if you want.
- IntelliJ Platform References
  - The attached document "A Short Guide for IntelliJ Plugin" explains how to create a very simple plugin for IntelliJ IDEA.
  - IntelliJ Platform SDK DevGuide
    `http://www.jetbrains.org/intellij/sdk/docs/`
  - Project Structure
    `http://www.jetbrains.org/intellij/sdk/docs/basics/project_structure.html`
  - Program Structure Interface (PSI)
    `http://www.jetbrains.org/intellij/sdk/docs/basics/architectural_overview/psi.html`
  - How to Use Trees (`JTree`)
    `https://docs.oracle.com/javase/tutorial/uiswing/components/tree.html`

Pairs

- You will work with a pair partner for this assignment. You have to work with your pair partner, but students from different pairs should not work with one another.
- Ideally you would be working on one computer, as in the actual pair programming. We recommend contacting your partner and scheduling a common time slot to meet for the assignment early.
- In addition to your code, each of you needs to submit the peer self-evaluation form *independently*. Complete an honest evaluation of work effort for the assignment.
- Only one of each pair needs to create a project in the repository and submit the code. But both have to submit the peer self-evaluation form to LMS (not GitLab).

Turning in

1. Create a private project with name homework5 in `https://csed332.postech.ac.kr` (one for each pair), and clone the project on your machine.
2. Commit your changes in your homework5 project that includes two directories `problem1` and `problem2`, and push them to the remote repository.
3. Tag your project with "submitted" and submit your homework. We will use the tagged version of your project for grading.
4. Submit your individually peer self-evaluation form (`peer-self.md`) *to LMS* (not GitLab). Note that the teaching staff can adjust the individual score based on the evaluation.