BINUS UNIVERSITY

BINUS INTERNATIONAL

Final Project Cover Letter

(Individual Work)

Student Information:

Surname: Sie **Given Name:** Jessica **Student ID Number:** 2502053653

Course Code: COMP6699001

Course Name: Object Oriented Programming

Class: L2BC

Lecturer: Jude Joseph Lamug Martinez, MCS

Type of Assignment: Final Project

Submission Pattern:

Due Date: June 10 2022

Submission Date: June 10 2022

The assignment should meet the below requirements.
1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating
BiNus International seriously regards all forms of plagiarism, cheating, and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity, and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality
By signing this assignment, I understand, accept, and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student: Jessica Sie

Classic Sudoku with JavaFX

Name: Jessica Sie
ID: 2502053653
Class: Object Oriented Programming - L2BC

Introduction

Background

Sudoku is a logic-based, number placement puzzle with the objective of filling in numbers from 1-9 in a 9x9 grid.

The Sudoku puzzle consists of a board of 9x9 grid of numbers, divided into 9 individual 3x 3 squares. The goal of Sudoku is to fill out the grid with numbers while following some simple rules. The rules are as follows:

- Each row must contain all the numbers between 1 to 9
- Each column must contain all the numbers between 1 to 9
- Each 3 by 3 square must contain all the numbers between 1 to 9

	5	8				2		
6		7	2					
3			4					
		5						
9		6						
	3				4			
1		9						4
				4		6	7	
8								

Figure 1 - Sudoku board

What the rules amount to is that the board cannot have the same number twice in any given row, column, or square.

Project Specification

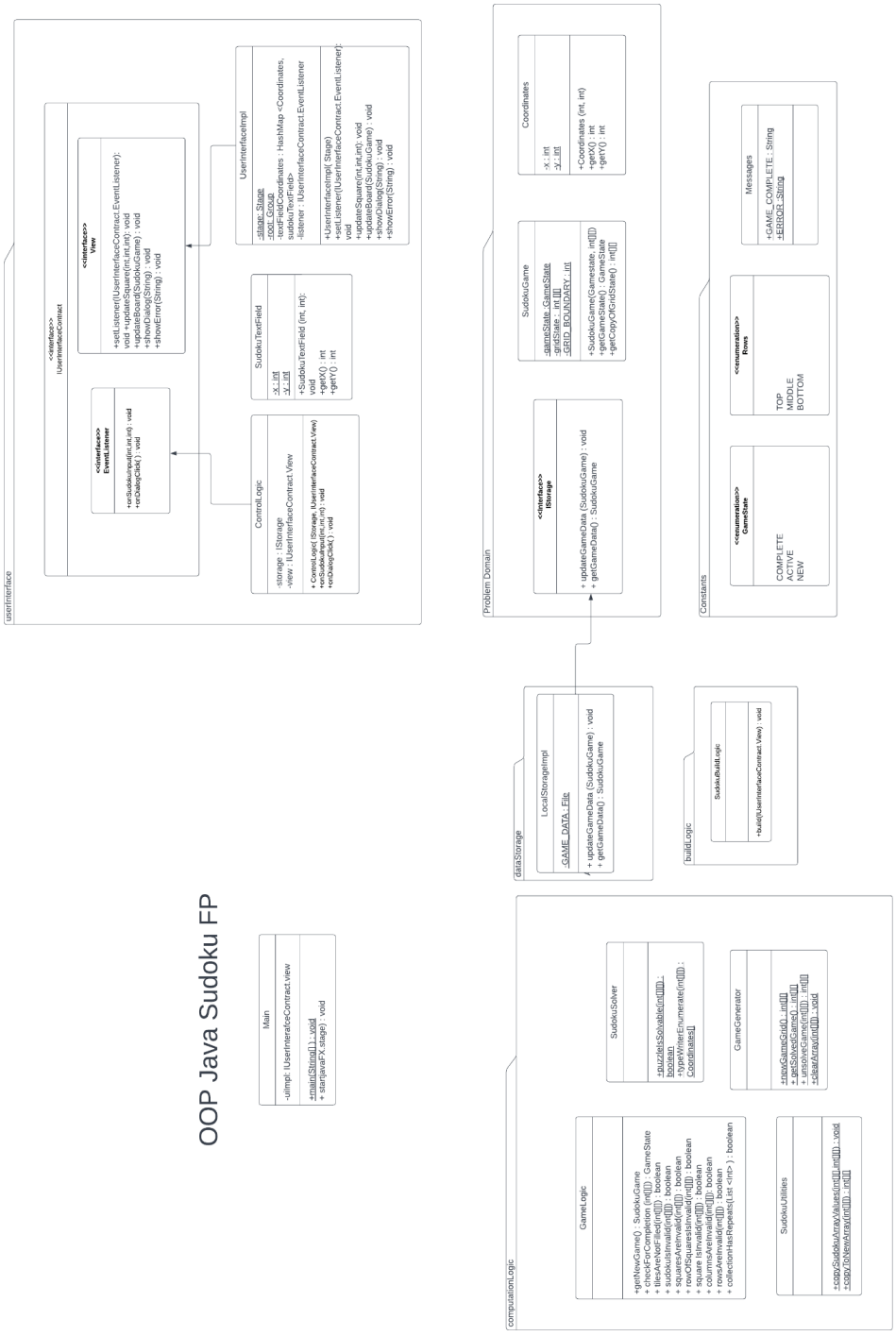
This project is a classic Sudoku game application programme made in Java using Intellij community edition IDE. The user interface is built with the JavaFX library. A HashMap data structure built into the java.utilities package, is used to store the hash values of the sudoku UI text fields.

As an avid fan and long-time player of the Sudoku puzzles, I was interested in the algorithms which created the puzzles and wanted to try for myself. To provide a greater challenge, I tried to implement a UserInterface for ease of readability. So I decided to challenge myself by building this application.

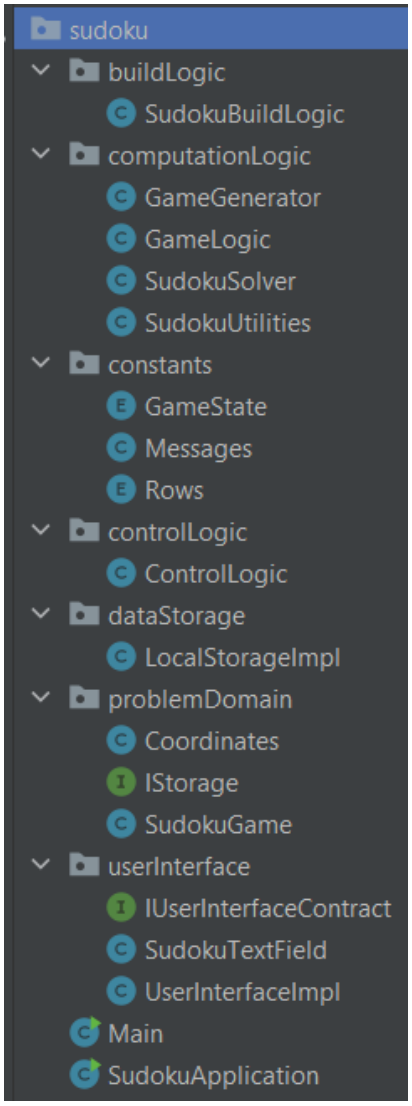
Main Body

Solution Design

Class diagram



File structure



Package name	Package Details	Classes	Class details
problemDomain	problemDomain is the backbone which the program is built from. It contains data the program needs to represent	SudokuGame	Data such as the sudoku grid, and gamestates
		Coordinates	X and y coordinates
		<<Interface>> IStorage	Separates frontend from backend
constants	The constants package Models constant data with enums or final-static-variables	GameState	An enum to store the 3 possible states of the sudoku game

		Messages	Stores pre-written messages into final-static-variable.
		Rows	Another enum to store the names of the top, middle and bottom row.
userInterface	userInterface package builds the UI of the application	SudokuTextField	This class extends the TextField class from JavaFX to create a blueprint for the sudoku tile objects.
		UserInterfaceImpl	Standing for user interface implementation, this class contains methods to build the elements of the application window.
		<<Interface>> IUserInterfaceContract	Contract is another word for interface. The IUserInterfaceContract is the UI Interface to facilitate communication between frontend and backend.
controlLogic	This package contains the files for the UI logic	ControlLogic	This class contains the logic behind the UI.
computationLogic	The computationLogic package contains the backend of the application, and the classes mainly use static methods.	GameGenerator	This class contains methods to create a new Sudoku puzzle.
		GameSolver	The methods in this class checks that the new puzzle created is solvable, as there are some number placement combinations that are impossible to be solved.
		GameLogic	This class reinforces the rules of the sudoku game.
		SudokuUtilities	SudokuUtilities class contains helper methods used throughout the backend code.
dataStorage		LocalStorageImplementation	This class stores the game to the local storage by

			writing to a file.
buildLogic		SudokuBuildLogic	Instantiates all classes needed to build the application
		Main	Contains Main class to launch Sudoku app
		SudokuApplication	Application container and entry point

Code explanation

User interface

- Sudoku tiles - SudokuTextField.java

  Each tile on the sudoku grid is an instance of the class SudokuTextField, which inherits from the TextField class from the JavaFX library. The methods, replaceText and replaceSelection, from TextField class are overridden in this class to stop an error where the text-fields duplicate the user inputs.

  To access the individual text fields or tiles, a HashMap data Structure is used to store the hash values of each tile by their coordinates. When a new Sudoku game is created, the tiles are iterated by their x and y coordinates while assigning a hash value to each one. This data structure of a HashMap eliminates the need to create and hold a reference variable for every single tile in the sudoku puzzle, making it simpler.

  The Key (<Key, Value> -> <Coordinates, Integer>) will be the HashCode of a given InputField for ease of lookup

  A HashMap, a data structure to store key/value pairs, is used to store references to each sudokuTextField object. Each individual object is referenced by using their X and Y coordinates as the “key”, rather than creating a reference variable for 9x9 sudokuTextField object.

- Game window - UserInterfaceImpl.java

  The game window is created from the UserInterfaceImpl class (UI Implementation), which implements the interface IUserInterfaceContract.View (named after what the user can see) and EventHandler interface.

  The main method initializeUserInterface, calls a bunch of helper functions to create the UI and display it to users. The following helper functions are called to create the window

  - drawBackground(root);
  - drawTitle(root)
  - drawSudokuBoard(root)
  - drawTextFields(root)

- drawGridLines(root)
- stage.show()

GameLogic

Use of static methods

For the backend and more logical side of the project, I made classes with public static methods, which do not have to be instantiated before it is called. This is much closer to how regular functions work in Python (my first programming language), much more suited to write algorithm based programs.

- Creating a Sudoku puzzle- GameGenerator.java
  The creation of a sudoku puzzle is divided into two main steps: 1)Creating a complete puzzle which adheres to the rules of sudoku; and 2) Removing 40 out of 81 values from the tiles randomly. The two steps are written in separate methods and called together with the getNewGameGrid(int[][]) method.

  The getSolvedGame(int[][]) method creates the complete sudoku puzzle by first generating a 2D array, and then randomly allocating each value in the range of [1,9], 9 times across the array, while following the constraints of the game.

  The unsolveGame(int[][]) method takes a solvedGame from the previous method as argument and randomly assigns 40 tiles to equal 0, or blank for the users to fill in. First, create a copy of the solvedArray, then it randomly replaces the tiles with 0s across the array while testing the puzzle by calling the puzzleIsSolvable(int[][]) method SudokuSolver class discussed below, to ensure it is still solvable for users.

- Solving the sudoku puzzle - SudokuSolver.java
  This file contains a method to solve the Sudoku puzzle based on a simple algorithm by Cornell university, along with a helper function to traverse the tiles typewriter style, left to right, top to bottom. The algorithm is as follows :

  1.Enumerate all empty cells in typewriter order (left to right, top to bottom)

  2.Our “current cell” is the first cell in the enumeration.

  3.Enter a 1 into the current cell. If this violates the Sudoku condition, try entering a 2, then a 3, and so forth, until
  a. the entry does not violate the Sudoku condition, or until
  b. you have reached 9 and still violate the Sudoku condition.

  4.In case a: if the current cell was the last enumerated one, then the puzzle is solved. If not, then go back to step 2 with the “current cell” being the next cell.
  In case b: if the current cell is the first cell in the enumeration, then the Sudoku puzzle does not have a

solution. If not, then erase the 9 from the current cell, call the previous cell in the enumeration the new "current cell", and continue with step 3. (Cornell, 2009)

- Checking if values entered by users is correct - GameLogic.java
  If user has finished the game, this file checks the values entered against the 3 main rules of the game, written separately in different methods:

Method name	Sudoku Rule
rowsAreInvalid()	Each row must contain all the numbers between 1 to 9
columnsAreInvalid()	Each column must contain all the numbers between 1 to 9
squareIsInvalid()	Each 3 by 3 square must contain all the numbers between 1 to 9

-

Working Program

To run this application, clone the github repository and open it in an IDE for java (intellij recommended). Configure your IDE of choice so that it has the javaFX library added to the application folder. Refer to the official javaFX documentation for the exact steps. In the Src folder, open the Main.java file and run it. The following window should appear.

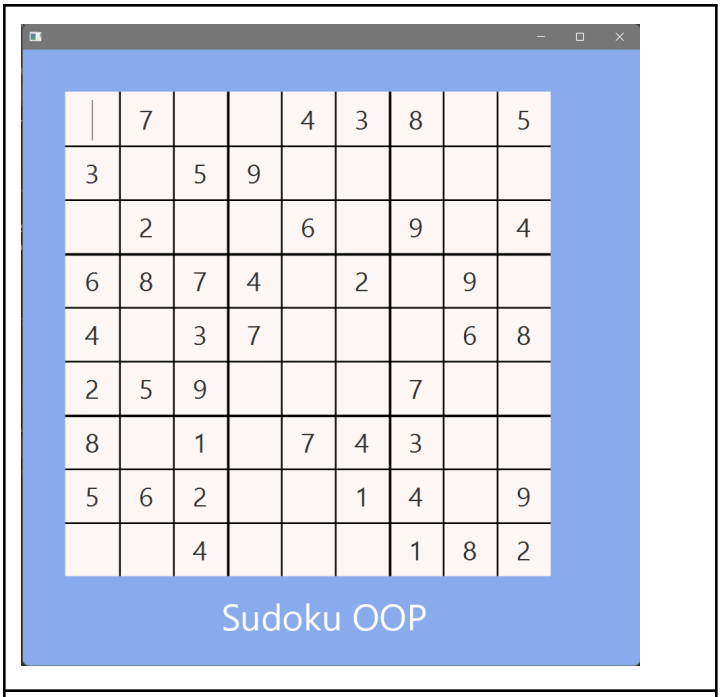


Figure 2 - Sudoku application window

To solve the Sudoku puzzle, simply click on any empty tile and enter a number from 1 to 9.

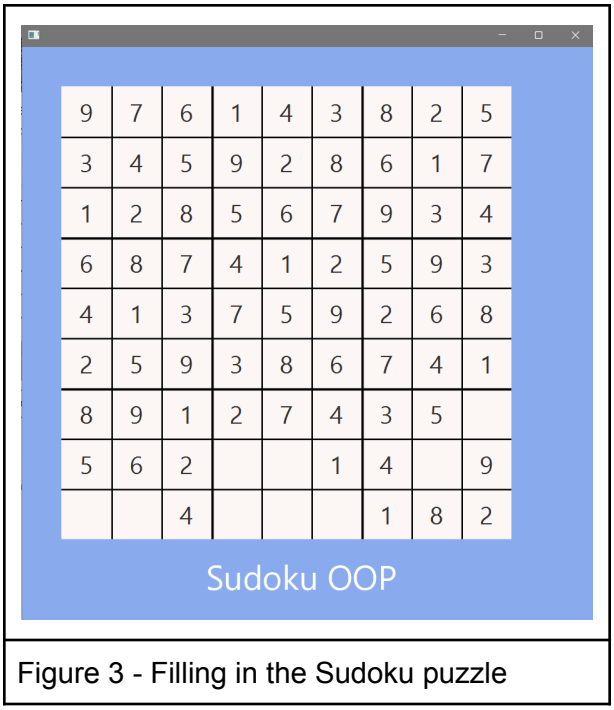


Figure 3 - Filling in the Sudoku puzzle

When the user has filled in the entire puzzle while adhering to the rules of Sudoku, a dialog box appears with a congratulations message and asks users for a new game. Clicking on the OK button resets the game and generates a new Sudoku board.

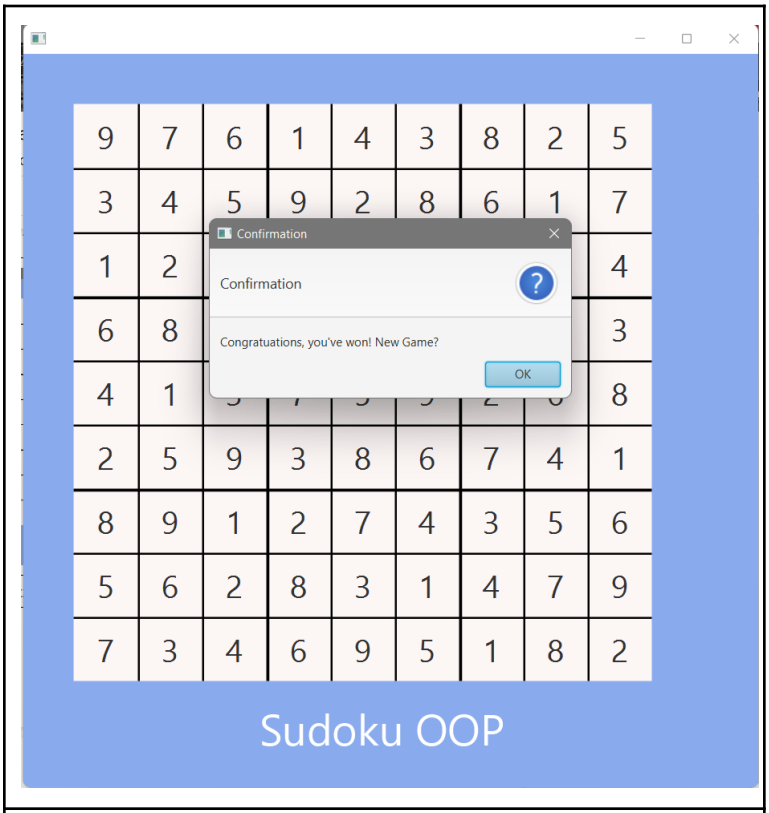


Figure 4- Successfully completing the puzzle

Conclusion

Building this project has provided myself with a deeper understanding of when to use Interfaces, final methods and static methods. which would definitely be useful in future Object oriented projects. Though simple, this was my first individual fullstack project, I learned to build a UI from scratch with java fx and how to “connect” the frontend to backend and allow them to communicate with each other. Overall creating this project has built up my oop and application development skills with java, and strengthened my resolve to explore more about frameworks for developing more complex applications.

Links

Video demo

https://www.youtube.com/watch?v=I16nZVOVjgk

Github repository

https://github.com/Jes-sie22/OOP_FinalProject

Resources used

Sudoku Game instructions
https://masteringsudoku.com/sudoku-rules-beginners/

Sudoku solving algorithm
https://www.youtube.com/watch?v=mcXc8Mva2bA&t=318s
http://pi.math.cornell.edu/~mec/Summer2009/meerkamp/Site/Solving_any_Sudoku_I.html

Install and set up javaFX in intellij IDE
https://openjfx.io/openjfx-docs/
https://www.youtube.com/watch?v=Ope4icw6bVk&t=182s

User Interface with JavaFX
https://www.youtube.com/watch?v=9XJicRt_FaI