Practical Programming Assignment

Programme Name: Sudoku Solver
Module Name and Code: Object Oriented Programming , AG200
Student ID Number: st120-10722
Lecturer's Name: Chasapis, Athanasios
Word Count: 1850

Εισαγωγή

Ο παρών κώδικας αποτελεί μια υλοποίηση αλγορίθμου επίλυσης του παιχνιδιού Sudoku, ακολουθώντας τις αρχές της **αρχιτεκτονικής επιπέδου επιχείρησης (Enterprise-level Architecture)** για το μαθημα Object Oriented Programming (AG200) στο Aegean College τμημα Λαρισας. Η σχεδίαση εναρμονίζεται με τις πλέον σύγχρονες πρακτικές ανάπτυξης λογισμικού, όπως το **Domain-Driven Design (DDD)**, οι **αρχές SOLID**, καθώς και η **Καθαρή Αρχιτεκτονική (Clean Architecture)**, διασφαλίζοντας την επεκτασιμότητα, τη συντηρησιμότητα και την ευχρηστία του αντικειμενοστραφη προγραμματισμενου συστήματος.

Η ανάλυση της παρούσας εφαρμογής διαρθρώνεται σε τρεις βασικές ενότητες:

1. **Αρχιτεκτονική και Δομή**
2. **Επίπεδο Τομέα (Domain Layer)**
3. **Υλοποίηση Σχεδιαστικών Προτύπων και Στρατηγικών Επίλυσης**

1. Αρχιτεκτονική και Δομή

Ο σχεδιασμός του κώδικα βασίζεται σε μια πολυεπίπεδη αρχιτεκτονική, διασφαλίζοντας την αποδοτικότητα, την ευελιξία και τη συντηρησιμότητα της εφαρμογής. Η δομή ακολουθεί το μοντέλο **Domain-Driven Design (DDD)**, χωρίζοντας την εφαρμογή σε **Domain Layer**, **Infrastructure Layer** και **Presentation Layer**. Αυτή η προσέγγιση διασφαλίζει την ανεξαρτησία της επιχειρησιακής λογικής από τις τεχνολογικές λεπτομέρειες, την παρουσίαση καθως και την ευελιξια προσθηκης περισσοτερων αλγοριθμων η δομων δεδομενων για την επιλυση του προβληματος . (Sudoku)

1.1 Επίπεδο Τομέα (Domain Layer)

Το **Domain Layer** περιλαμβάνει τις βασικές οντότητες του Sudoku, καθώς και τις διεπαφές που καθορίζουν την επιχειρησιακή λογική. Οι κύριες κλάσεις είναι:

- **Grid**: Αναπαριστά τον πίνακα Sudoku ως δισδιάστατο πίνακα ακεραίων αριθμων, παρέχοντας μεθόδους για ανάκτηση και ενημέρωση κελιών.
- **Position**: Ορίζει τις συντεταγμένες των κελιών στον πίνακα, διασφαλίζοντας την ακεραιότητα των θέσεων.
- **ISudokuValidator**: Διεπαφή που επιτρέπει την επικύρωση κινήσεων σύμφωνα με τους κανόνες του παιχνιδιου Sudoku.
- **ISudokuSolver**: Ορίζει τη βασική λειτουργικότητα των αλγορίθμων επίλυσης, επιτρέποντας την εναλλαγή στρατηγικών.

1.2 Υποδομή (Infrastructure Layer)

Το **Infrastructure Layer** υλοποιεί τις διεπαφές του επιπέδου τομέα και διαχειρίζεται την πρόσβαση στα δεδομένα του επεξεργαζωμενου πινακα (Sudoku). Ενδεικτικλα οι κλάσεις:

- **BacktrackingSolver** και **DFSArrayListSolver**: Διαφορετικές στρατηγικές επίλυσης του Sudoku.
- **FileGridLoader**: Υλοποιεί τη φόρτωση πινάκων Sudoku από αρχεία (κατα στιγμη μονο *.txt), επιτρέποντας την ευέλικτη εισαγωγή δεδομένων στο προγραμμα.

1.3 Παρουσίαση (Presentation Layer)

Το **Presentation Layer** ασχολείται με την αλληλεπίδραση χρήστη μέσω του **ConsoleUI**, παρέχοντας:

- Δυνατότητα επιλογής στρατηγικής επίλυσης.
- Προβολή της τρέχουσας κατάστασης του Sudoku.
- Μηχανισμούς επικοινωνίας με το **Domain Layer** χωρίς εξάρτηση από την επιχειρησιακή λογική.
- Σε δευτερο πλανο, αυτο το Layer μπορει να χρησιμοποιηθει για να δημιουργηθει και κανονικη UI σε γραφικο περιβαλλον για την πιο ευκολη χρηση.

1.4 Αρχές SOLID

Ο σχεδιασμός της εφαρμογής υιοθετεί τις αρχές **SOLID** για τη βελτίωση της ποιότητας του κώδικα:

- **Single Responsibility Principle (SRP)**: Κάθε κλάση έχει μία μόνο ευθύνη.

- **Open/Closed Principle (OCP)**: Νέοι αλγόριθμοι μπορούν να προστεθούν χωρίς αλλαγές στον υπάρχοντα κώδικα, καθως καθε λυση ειναι μια Κλαση/Αντικειμενο μονη της και ανεξαρτητη απο τις αλλες λυσεις.
- **Liskov Substitution Principle (LSP)**: Οι υλοποιήσεις μπορούν να αντικαθιστούν τις διεπαφές τους χωρίς παρενέργειες.
- **Interface Segregation Principle (ISP)**: Χρήση εξειδικευμένων διεπαφών για αποφυγή περιττών εξαρτήσεων.
- **Dependency Inversion Principle (DIP)**: Η διαχείριση εξαρτήσεων μέσω διεπαφών καθιστά το σύστημα πιο ευέλικτο.

1.5 Καθαρή Αρχιτεκτονική (Clean Architecture)

Η αρχιτεκτονική της εφαρμογής διαχωρίζει την λογική λυσης από τις λεπτομέρειες της υλοποίησης, εφαρμόζοντας την **Καθαρή Αρχιτεκτονική**. Τα δεδομένα και οι διεπαφές καθορίζονται στο **Domain Layer**, ενώ οι εξωτερικές λεπτομέρειες, όπως η αποθήκευση δεδομένων και η παρουσίαση, βρίσκονται στα αντίστοιχα επίπεδα, εξασφαλίζοντας την ανεξαρτησία των κρίσιμων τμημάτων του συστήματος. Στην παρουσα ασκηση δεν υπαρχει μεγαλη πολυπλοκοτητα, χωρις ομως αυτο να σημαινει οτι ενα τετοιο πλανο δεν μπορει να εφαρμοστει και σε μικρες εφαρμογες που πολυ γρηγορα μπορουν να γιγαντωθουν λογω scope creep.

Ο σχεδιασμός του κώδικα υλοποιείται σύμφωνα με δοκιμασμένες αρχές και σχεδιαστικά πρότυπα που ενδείκνυνται για εφαρμογές υψηλής κλίμακας. Το κυρίαρχο αρχιτεκτονικό πρότυπο είναι το **Domain-Driven Design (DDD)**, το οποίο οργανώνει τη δομή της εφαρμογής σε τρία διακριτά επίπεδα: **Domain**, **Infrastructure**, και **Presentation**.

2. Επίπεδο Τομέα (Domain Layer)

Το **Επίπεδο Τομέα (Domain Layer)** αποτελεί την καρδιά και λογικη της εφαρμογής, καθορίζοντας τη βασική λογική λυσης και τα θεμελιώδη συστατικά του Sudoku.

Περιλαμβάνει τις κύριες οντότητες, αντικείμενα αξίας και διεπαφές που ορίζουν τους κανόνες και τη λειτουργικότητα του συστήματος.

2.1 Οντότητες και Αντικείμενα Αξίας

Η κύρια οντότητα είναι η **Grid**, η οποία αντιπροσωπεύει τον πίνακα του παιχνιδιου Sudoku. Είναι υπεύθυνη για την αποθήκευση της κατάστασης του παιχνιδιού μέσω ενός δισδιάστατου πίνακα ακεραίων, όπου κάθε στοιχείο αντιπροσωπεύει μια αριθμητική τιμή ή ένα κενό κελί. Παρέχει επίσης βοηθητικές μεθόδους για την ανάκτηση και ενημέρωση των κελιών, καθώς και για την επικύρωση της συνολικής δομής του πλέγματος.

Ένα άλλο βασικό αντικείμενο αξίας είναι η **Position**, το οποίο αναπαριστά μια σταθερή τοποθεσία στον πίνακα Sudoku. Η **Position** αποτελείται από δύο συντεταγμένες, τη γραμμή (row) και τη στήλη (column), και διασφαλίζει ότι οι θέσεις στο πλέγμα διαχειρίζονται με ασφάλεια και ακεραιότητα.

2.2 Διεπαφές και Λογική Λυσης

Το **ISudokuValidator** είναι μια διεπαφή που ορίζει τη λειτουργικότητα επικύρωσης κινήσεων στο Sudoku. Ελέγχει αν μια τιμή μπορεί να τοποθετηθεί σε ένα συγκεκριμένο κελί χωρίς να παραβιάζει τους κανόνες του παιχνιδιού. Η υλοποίησή του επιτρέπει διαφορετικές στρατηγικές επικύρωσης, όπως ελέγχους σε γραμμές, στήλες και περιοχές του πίνακα.

Η βασική διεπαφή για την επίλυση του Sudoku είναι το **ISudokuSolver**, το οποίο ορίζει τη μέθοδο **Solve()**. Αυτή η μέθοδος εφαρμόζεται από διαφορετικούς αλγόριθμους, επιτρέποντας την εναλλαγή στρατηγικών επίλυσης χωρίς να επηρεάζεται ο βασικός κώδικας της εφαρμογής. Έτσι, το **Domain Layer** παραμένει ανεξάρτητο από συγκεκριμένες υλοποιήσεις, διατηρώντας την αρχή της αντιστροφής εξαρτήσεων (Dependency Inversion Principle - DIP).

Μέσω αυτής της αρχιτεκτονικής προσέγγισης, το **Επίπεδο Τομέα** παρέχει έναν σαφή διαχωρισμό ανάμεσα στη βασική λογική λυσης και τις τεχνικές λεπτομέρειες της εφαρμογής. Αυτό καθιστά την εφαρμογή πιο επεκτάσιμη και ευέλικτη, διευκολύνοντας την ενσωμάτωση νέων στρατηγικών επίλυσης και μηχανισμών επικύρωσης στο μέλλον.

Το **Domain Layer** περιλαμβάνει τις βασικές οντότητες και αντικείμενα αξίας (Value Objects), που αποτελούν το θεμέλιο της εφαρμογής.

3. Υλοποίηση Σχεδιαστικών Προτύπων και Στρατηγικών Επίλυσης

*3.1 Στρατηγική **BacktrackingSolver***

Η **BacktrackingSolver** υλοποιεί τη στρατηγική αναδρομικής δοκιμής (backtracking), η οποία επιτρέπει τη διερεύνηση πιθανών αριθμών για κάθε κελί. Η διαδικασία αυτή βασίζεται σε αναδρομικές κλήσεις, όπου δοκιμάζεται κάθε πιθανός αριθμός σε ένα κενό κελί, και αν μια επιλογή αποδειχθεί λανθασμένη, το πρόγραμμα αναιρεί την τελευταία αλλαγή (backtrack) και δοκιμάζει την επόμενη πιθανή τιμή. Αυτή η τεχνική καθιστά τον αλγόριθμο ιδιαίτερα αποδοτικό για μικρού μεγέθους πλέγματα Sudoku, όμως μπορεί να γίνει αναποτελεσματική σε πολύπλοκα Sudoku λόγω της εκθετικής αύξησης του χώρου αναζήτησης.

Η υλοποίηση της **BacktrackingSolver** περιλαμβάνει τη μέθοδο **Solve()**, η οποία εκτελεί τον αλγόριθμο αναδρομικής δοκιμής. Χρησιμοποιεί βοηθητικές μεθόδους για τον έλεγχο της εγκυρότητας των αριθμών και την εύρεση των κενών κελιών. Παρά την απλότητά της, η μέθοδος αυτή εξασφαλίζει ότι ο αλγόριθμος θα βρει πάντοτε μια έγκυρη λύση αν υπάρχει.

*3.2 Στρατηγική **DFSArrayListSolver***

Η στρατηγική **DFSArrayListSolver** βασίζεται στην τεχνική **Βαθιάς Αναζήτησης (Depth-First Search - DFS)**, η οποία ακολουθεί μια διαφορετική προσέγγιση επίλυσης. Αντί να βασίζεται αποκλειστικά σε αναδρομή, χρησιμοποιεί μια **δυναμική λίστα (ArrayList)** για να αποθηκεύει τις ενδιάμεσες καταστάσεις του πλέγματος Sudoku.

Η **DFSArrayListSolver** λειτουργεί δημιουργώντας έναν **εκτεταμένο χώρο αναζήτησης**, όπου κάθε πιθανή κίνηση εξετάζεται σε βάθος πριν επιστρέψει προς τα πίσω αν η τρέχουσα διαδρομή δεν οδηγεί σε λύση. Αυτή η τεχνική μπορεί να είναι πιο αποδοτική από την απλή αναδρομική δοκιμή σε περιπτώσεις όπου το Sudoku έχει πολλές πιθανές λύσεις ή περιορισμούς, καθώς αποθηκεύει και διαχειρίζεται τις καταστάσεις δυναμικά, αντί να βασίζεται μόνο στο στοίβο αναδρομής του προγράμματος.

Η υλοποίηση της **DFSArrayListSolver** απαιτεί τη διαχείριση μιας στοίβας από αντικείμενα που αντιπροσωπεύουν τις καταστάσεις του Sudoku. Σε κάθε βήμα, η μέθοδος **Solve()** προσπαθεί να προχωρήσει βαθύτερα στη λύση, εξετάζοντας πιθανές τιμές και αποθηκεύοντας τις καταστάσεις σε μια λίστα. Αν φτάσει σε αδιέξοδο, επιστρέφει στην προηγούμενη κατάσταση και συνεχίζει την αναζήτηση.

3.3 Σχεδιαστικά Πρότυπα και Υλοποίηση

Η επιλογή στρατηγικής επίλυσης του Sudoku υλοποιείται μέσω του σχεδιαστικού προτύπου **Strategy**, το οποίο επιτρέπει την εύκολη εναλλαγή αλγορίθμων χωρίς την ανάγκη αλλαγών στην κύρια ροή του προγράμματος. Η υλοποίηση περιλαμβάνει το interface **ISudokuSolver**, το οποίο ορίζει τη μέθοδο **Solve()**, και τις διαφορετικές στρατηγικές που την υλοποιούν.

Η ενσωμάτωση των στρατηγικών στην εφαρμογή γίνεται μέσω **Dependency Injection (DI)**. Το **Infrastructure Layer** διαχειρίζεται τη δημιουργία των αντικειμένων επίλυσης και επιτρέπει τη δυναμική επιλογή στρατηγικής από τον χρήστη ή το σύστημα. Αυτή η προσέγγιση βελτιώνει την επεκτασιμότητα της εφαρμογής, επιτρέποντας την προσθήκη νέων αλγορίθμων χωρίς αλλαγές στον υπάρχοντα κώδικα.

3.4 Σύγκριση και Βελτιστοποίηση

Παρότι η **BacktrackingSolver** είναι πιο εύκολη στην υλοποίηση, η **DFSArrayListSolver** προσφέρει καλύτερη διαχείριση μνήμης και πιο ευέλικτη αναζήτηση λύσεων. Η χρήση τεχνικών όπως **παράλληλη επεξεργασία (parallel processing)** ή **ευριστικές μέθοδοι (heuristic approaches)** μπορεί να βελτιώσει την απόδοση και των δύο στρατηγικών.

Συνολικά, η υλοποίηση στρατηγικών επίλυσης Sudoku ακολουθεί τις αρχές του **καλού σχεδιασμού λογισμικού**, επιτρέποντας εύκολη επέκταση και συντήρηση. Ο διαχωρισμός των στρατηγικών μέσω του προτύπου **Strategy**, σε συνδυασμό με το **Dependency Injection**, διασφαλίζει ότι η εφαρμογή είναι αποδοτική, επεκτάσιμη και εύκολη στη χρήση.

4 Code

Ενδεικτικα η κυριως κλαση του προγραμματος. Για να μπορεσετε να κατεβασετε και να τρεξετε την Solution παρακαλω απευθηνθειτε στην τρεχουσα URL : https://github.com/georgiosbalatzis/SudokuSolverFinal ενναλακτικα στο επισυναπτομενο αρχειο.

```
using SudokuSolver.Domain.Entities;
using SudokuSolver.Domain.Interfaces;
using SudokuSolver.Domain.Models;
using SudokuSolver.Infrastructure.IO;
using SudokuSolver.Infrastructure.Solvers;
using SudokuSolver.Infrastructure.Validators;
using SudokuSolver.Presentation.ConsoleUI;

public class Program
{
    private readonly ISudokuSolver[] _solvers;
    private readonly IGridLoader _gridLoader;
    private readonly IGridPrinter _gridPrinter;

    public Program(
        ISudokuSolver[] solvers,
        IGridLoader gridLoader,
        IGridPrinter gridPrinter)
    {
        _solvers = solvers;
        _gridLoader = gridLoader;
        _gridPrinter = gridPrinter;
    }

    public void Run()
    {
        while (true)
        {
            try
            {
                DisplayMenu();
                var choice = GetUserChoice();
                if (choice == _solvers.Length + 1) break;

                var grid = LoadGrid();
                if (grid == null) continue;

                Console.WriteLine("\nΑρχικό Sudoku:");
```

```
            _gridPrinter.Print(grid);

            var result = _solvers[choice - 1].Solve(grid);
            DisplayResult(result);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"\nΣφάλμα: {ex.Message}");
        }

        Console.WriteLine("\nΠατήστε οποιοδήποτε πλήκτρο για συνέχεια...");
        Console.ReadKey();
        Console.Clear();
    }
}

private void DisplayMenu()
{
    Console.WriteLine("Καλώς ήρθατε στον Επιλυτή Sudoku!");
    Console.WriteLine("Επιλέξτε μέθοδο επίλυσης:");
    for (int i = 0; i < _solvers.Length; i++)
        Console.WriteLine($"{i + 1}: {_solvers[i].GetType().Name}");
    Console.WriteLine($"{_solvers.Length + 1}: Έξοδος");
}

private Grid LoadGrid()
{
    Console.Write("\nΕισάγετε το όνομα του αρχείου εισόδου: ");
    var fileName = Console.ReadLine();
    return _gridLoader.LoadFromFile(fileName);
}

private void DisplayResult(SolverResult result)
{
    if (result.Success)
    {
        Console.WriteLine("\nΛύση βρέθηκε!");
        Console.WriteLine($"Μέθοδος: {result.AlgorithmUsed}");
        Console.WriteLine($"Χρόνος επίλυσης: {result.SolvingTime.TotalSeconds:F3}
δευτερόλεπτα");
        Console.WriteLine("\nΤελική λύση:");
        _gridPrinter.Print(result.Solution);
    }
    else
    {
```

```
            Console.WriteLine("\nΔεν βρέθηκε λύση για το συγκεκριμένο Sudoku.");
        }
    }

    private int GetUserChoice()
    {
        while (true)
        {
            Console.Write("\nΕπιλογή: ");
            if (int.TryParse(Console.ReadLine(), out int choice) &&
                choice >= 1 && choice <= _solvers.Length + 1)
                return choice;

            Console.WriteLine("Μη έγκυρη επιλογή. Παρακαλώ προσπαθήστε ξανά.");
        }
    }

    public static void Main(string[] args)
    {
        var validator = new SudokuValidator();
        var cellFinder = new CellFinder();

        var solvers = new ISudokuSolver[]
        {
            new BacktrackingSolver(validator, cellFinder),  // Backtracking solver
            new DFSArrayListSolver(validator, cellFinder)    // DFS solver
        };

        var gridLoader = new FileGridLoader();
        var gridPrinter = new GridPrinter();

        var program = new Program(solvers, gridLoader, gridPrinter);
        program.Run();
    }

}
```

Συμπέρασμα

Η παρούσα εργασία παρουσιάζει μια βέλτιστη αρχιτεκτονική επίλυσης του Sudoku, ακολουθώντας τις αρχές της **Clean Architecture**, του **Domain-Driven Design**, καθώς και των **αρχών SOLID**. Ο διαχωρισμός της επιχειρησιακής λογικής από την παρουσίαση και την υποδομή, σε συνδυασμό με τη χρήση σχεδιαστικών προτύπων όπως τα **Repository** και **Strategy**, καθιστά την εφαρμογή επεκτάσιμη και συντηρήσιμη, επιτρέποντας την εύκολη προσαρμογή και βελτίωσή της στο μέλλον.