Adam Kapuściński | 229907
Damian Szczeciński | 230016

wtorek 17:45 –19:15
24.05.2021

Przedmiot

Podstawy analizy cyklu życia

Sprawozdanie z części praktycznej

Opis programu

Naszym programem jest projekt aplikacji tworzącym łamigłówki Sudoku do rozwiązywania przez użytkownika. Udostępnia on możliwość wyboru poziomu trudności, zapisu gry oraz jej wczytanie. Przy zapisie umożliwia ona wybór sposobu archiwizacji. Do wyboru mamy możliwość korzystania z pliku lub bazy danych. Dodatkowo aplikacja ma dodatkowe funkcje takie jak możliwość poddania się, automatycznego rozwiązania Sudoku, sprawdzenie poprawności wpisanych przez użytkownika wartości do poszczególnych pól. Poza tym, można także wybrać język interfejsu. Nasza aplikacja udostępnia 3 wersje językowe: polską, angielską oraz japońską.

Program został napisany w języku Java. Zbudowany używając JDK w wersji 14 i zarządzany przez apache-maven w wersji 3.6.3. Projekt składa się z dwóch komponentów: „Modelu obiektowego" oraz „GUI", gdzie model obiektowy ma dodatkowo dostęp do zewnętrznej bazy danych. Jest to program, w który występuje dziedziczenie, abstrakcja i polimorfizm.

Model obiektowy

Główną klasą modelu jest klasa *SudokuBoard*. Reprezentuje ona planszę na ekranie. W jej konstruktorze podajemy *solver* czyli klasę odpowiedzialną za rozwiązywanie planszy. Jest to w zasadzie implementacja wzorca projektowego strategii. *SudokuBoard* korzysta z *solvera* do tworzenia planszy, rozwiązywania jej oraz sprawdzenia, czy da się ją rozwiązać. *SudokuBoard* implementuje także interfejs *PropertyChangeListener* dzięki czemu może informować *listenera* o zmianie w polu na planszy. Do przekazywania informacji używamy klasy *SudokuActionEvent* która przechowuje informacje o współrzędnych zmienionego pola oraz czy z tą nową wartością da się rozwiązać grę.

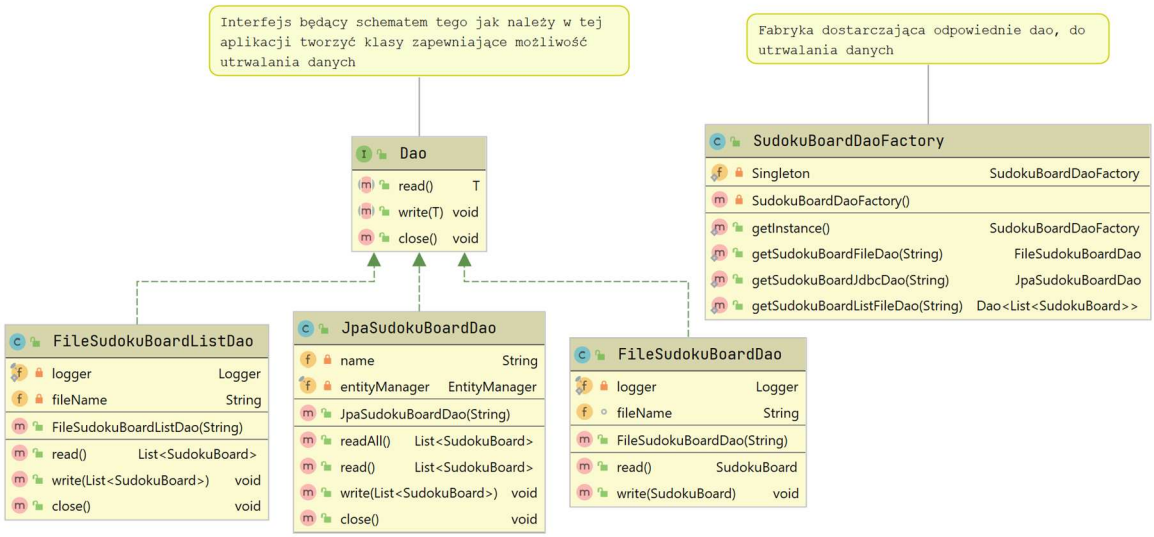
Każde pole w *SudokuBoard* jest obiektem klasy *SudokuField*. Obiekt ten odpowiada jedynie za to, aby liczby umieszczane w polach były z zakresu 0-9, gdzie 0 oznacza puste pole.

Difficulty to enum, który przechowuje definicje poziomów trudności, a także przygotowuje planszę pod dany poziom trudności.

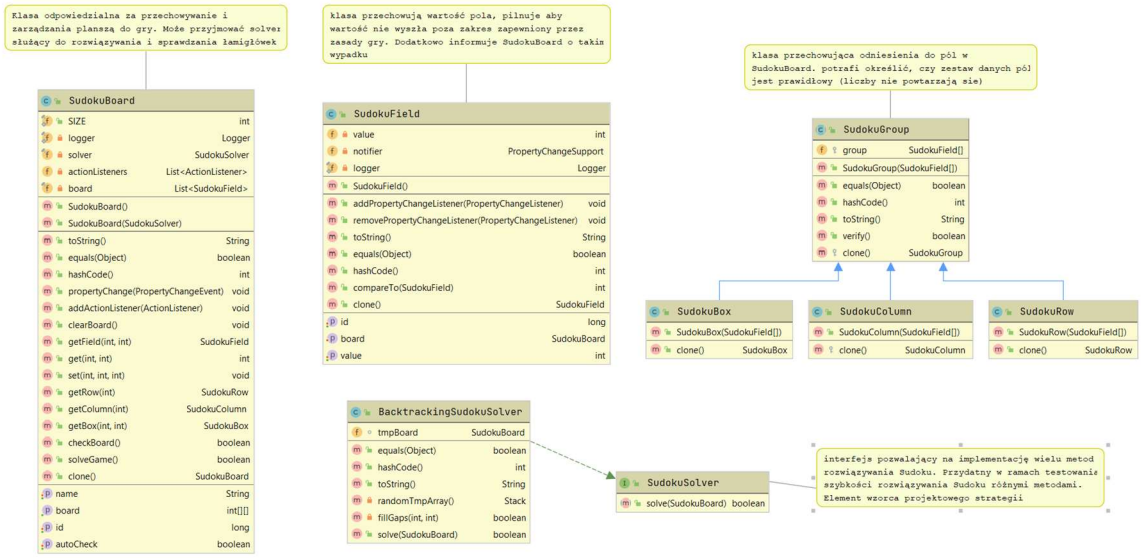
Do zapisywania planszy wykorzystujemy obiekty z interfejsem *Dao*, a pozyskujemy je przy użyciu klasy *SudokuBoardDaoFactory*, która jest Singletonem.

Poza tym mamy także zaimplementowane własne wyjątki, jednak one jedynie opakowują zwykłe wyjątki i w większości nie przechowują dodatkowych informacji.

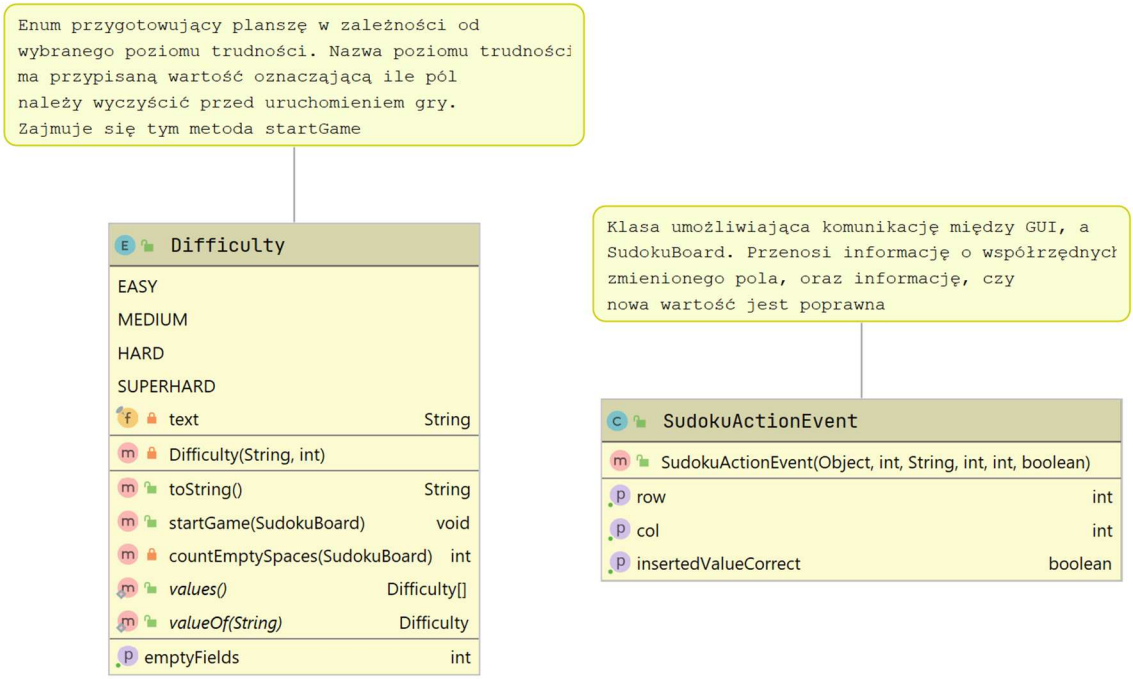
Diagram klas



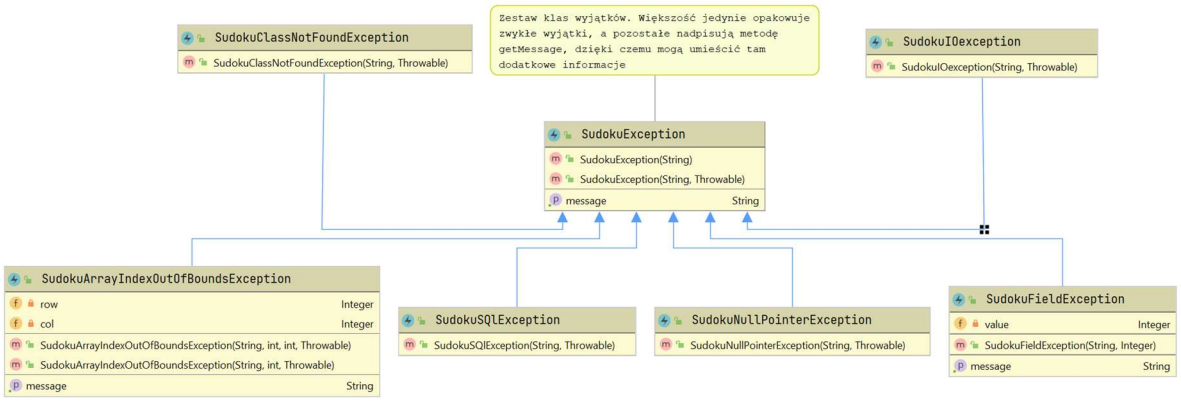
Rysunek 1. Diagram klas modelu obiektowego część 1



Rysunek 2. Diagram klas modelu obiektowego część 2



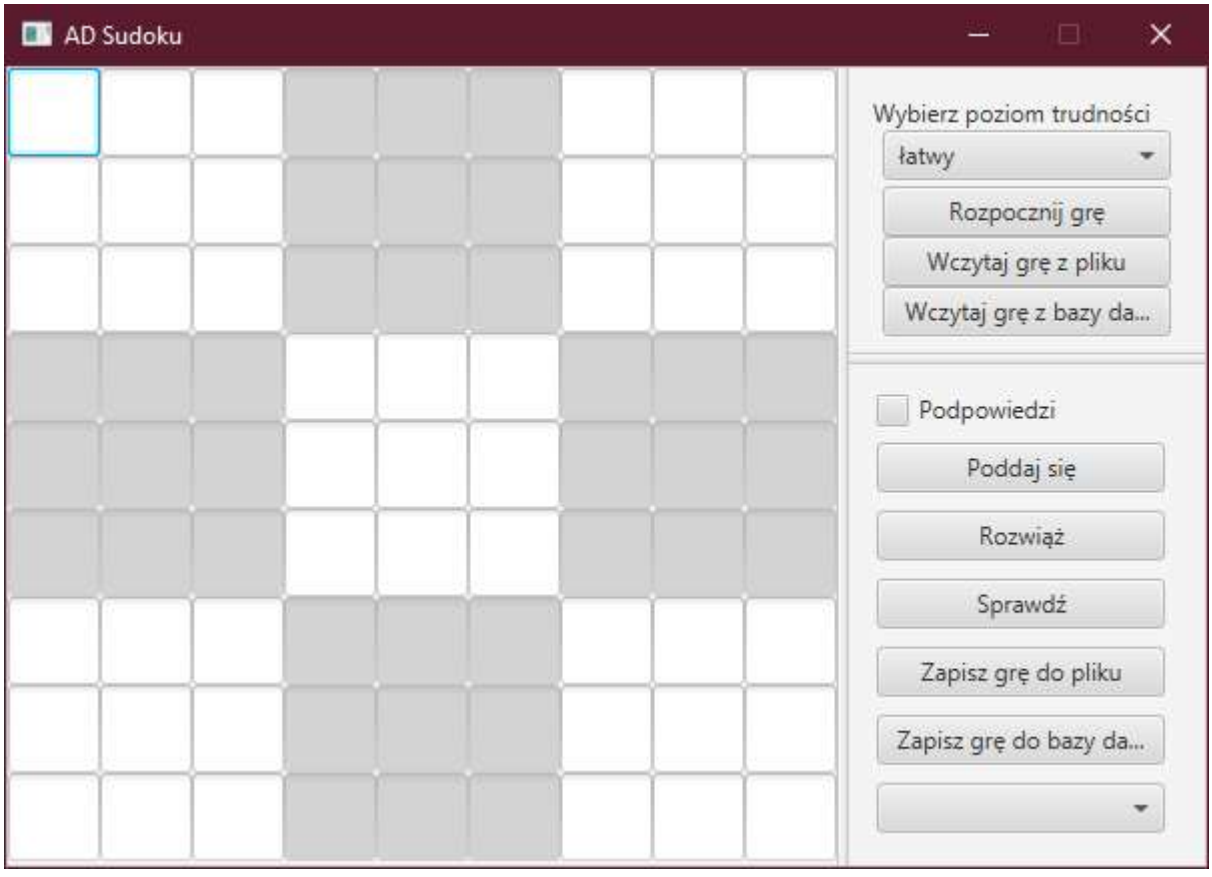
Rysunek 3. Diagram klas modelu obiektowego część 3



Rysunek 4. Diagram klas modelu obiektowego część 4

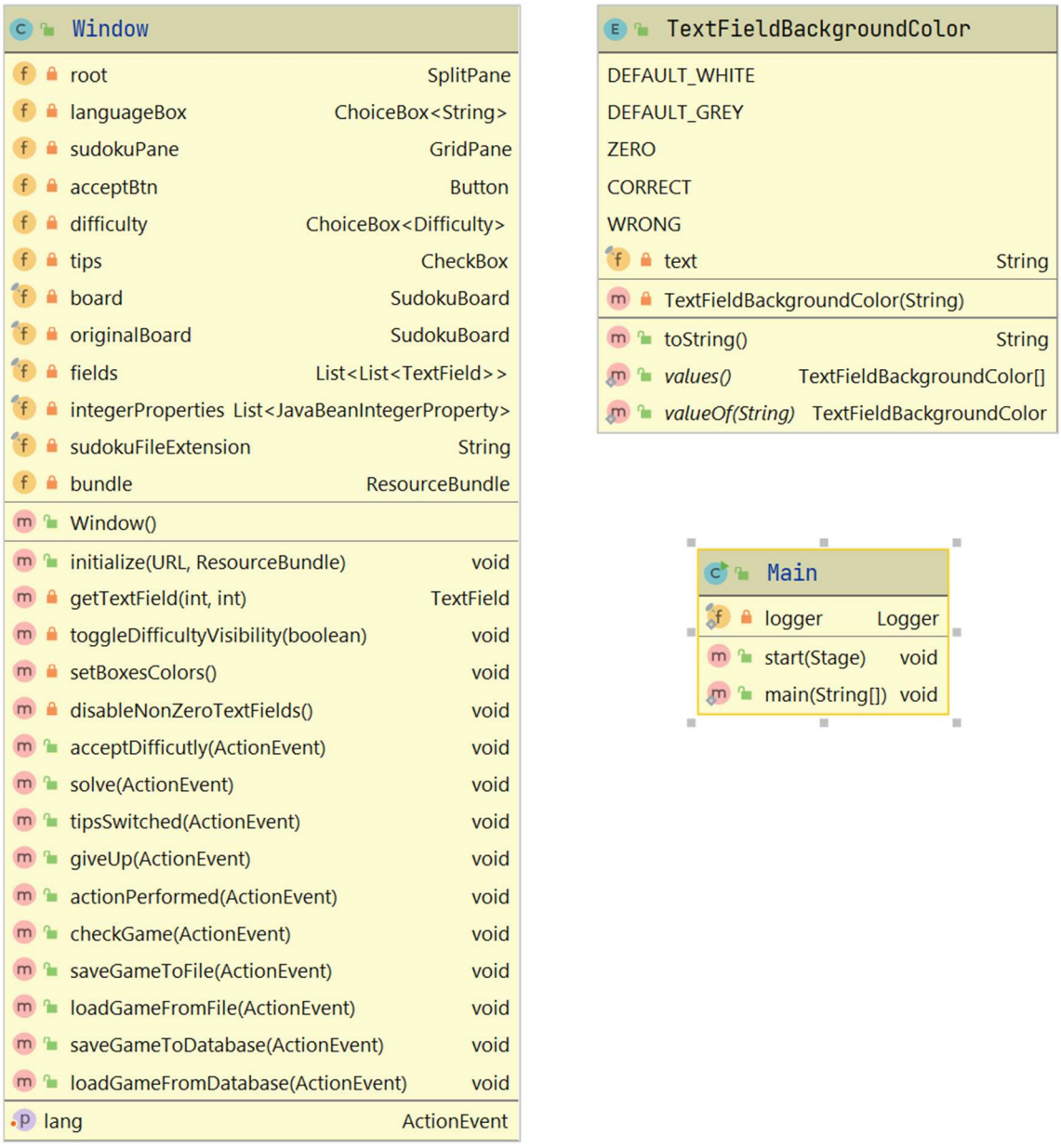
GUI

Na GUI programu składa się jedynie jedna klasa *Window* oraz enum *TextFieldBackgroundColor*, który zawiera definicje dla kolorów tła pól planszy. Klasa *Window* odpowiada za obsługę eventów takich jak: wpisanie liczby do pola w GUI, czy kliknięcie przycisków. Zawiera także kod zapewniający synchronizację pól w GUI z polami w obiekcie klasy *SudokuBoard*. Jako, że GUI tworzymy wykorzystując bibliotekę JavaFX, to cały opis okna programu przechowujemy w pliku Window.fxml który jest zasobem dla modułu *View*.



Rysunek 5. Prezentacja GUI programu

Diagram klas GUI



Rysunek 6. Diagram klas GUI

Opis stosowanych metryk dla poszczególnych klas aplikacji

Cyclomatic Complexity (CC)

Złożoność cyklomatyczna – zwana również miarą złożoności McCabe'a jest to metryka stosowana do pomiaru stopnia skomplikowania programu. Zlicza ona złożoność wykorzystywanych w metodzie algorytmów na podstawie wszystkich przypadków testowych. Analizę tej wartości podejmuje się najczęściej na podstawie poniższych wartości:

- od 1 do 10 – nieznaczne ryzyko
- od 11 do 20 – średnie ryzyko
- od 21 do 50 –wysokie ryzyko
- powyżej 50 – bardzo wysoki poziom ryzyka.

Response For a Class (RFC)

Liczba metod i konstruktorów wywoływanych przez klasę – jest to liczba różnych metod, które mogą być wykonywane, gdy obiekt tej klasy odbiera komunikat. Im wyższe RFC tym większa złożoność.

Lack of Cohesion (LCOM)

Brak spójności w metodach – wskazuje stopień braku spójności między metodami klasy. Brak spójności w metodach jest obliczany przez odjęcie od liczby par metod nie mających wspólnych atrybutów liczby tych par, które je mają. Im mniejszy wskaźnik tym lepiej. Duża liczba wskazuje na dużą złożoność.

Coupling Between Object Classes (CBO)

Sprzężenie między klasami obiektów – jest to liczba klas połączonych z daną klasą. Występuje jako wywołanie metody, dostępu do pola, dziedziczenia, argumentów, zwracanych typów i wyjątków. Im większe sprzężenie między klasami tym większa złożoność obliczeniowa.

Depth of Inheritance Tree (DIT)

Głębokość drzewa dziedziczenia – jest to metryka zapewniająca dla każdej klasy miarę poziomów dziedziczenia z góry hierarchii obiektów. W naszym przypadku (język *Java*) wszystkie klasy dziedziczą po klasie *Object*, więc minimalną wartością jest 1. Im większy wskaźnik DIT tym większa złożoność obliczeniowa

Number of Children (NOC)

Liczba podrzędnych elementów klasy – jest to liczba mierząca liczbę bezpośrednich elementów podrzędnych klasy (dzieci). Większa liczba może oznaczać większą złożoność obliczeniową ale również może oznaczać niewłaściwą abstrakcję.

Weighted Methods per Class (WMC)

Uśrednione metody na klasę – służy do określania złożoności klasy jako zbioru metod. Jest liczbą wszystkich metod w danej klasie lub jak w naszym przypadku sumą CC. Im wyższy wskaźnik tym większa złożoność.

Opis stosowanych metryk dla całego kodu aplikacji

Lines of Code (LOC)

Liczba linii kodu – wartość określająca sumę linii kodu, pustych przestrzeni oraz komentarzy. Jej pochodnymi są NCNB oraz EXEC. Im wyższy wskaźnik, tym kod będzie bardziej złożony oraz trudniejszy do zrozumienia i utrzymania.

Non-comment non-blank (NCNB)

Liczba linii kodu – jest to liczba linii kodu z wykluczeniem linii pustych jak i komentarzy. Daje to pogląd na faktyczną ilość linii kodu pełniących jakieś zadania.

Executable statements (EXEC)

Liczba linii kodu wykonywalnego – jest to liczba wszystkich linii takich jak we wskaźniku NCNB, z dodatkowym wykluczeniem linii definiujących metod. Warunki (‘if’), pętle (‘for’, ‘while’), mające jedynie symbole definicji bloku kodu (‘{’, ’}’) i tym podobne.

Comment percentage (CP)

Procent komentarzy – metryka określająca stosunek linii kodu (NCNB) do linii komentarzy. Większy udział komentarzy świadczy o dobrej dokumentacji kodu, a więc łatwiejszego zrozumienia go

Wartości wskaźników dla poszczególnych klas

Difficulty	
nazwa funkcji	**CC**
Difficulty	1
getEmptyFields	1
toString	1
startGame	2
countEmptySpaces	4
Podsumowanie Difficulty	
RFC	9
LCOM	8
CBO	1
DIT	1
NOC	0
WMC	9

Tabela 1. Wyniki metryk dla klasy Difficulty

Klasa *Difficulty* pozwala na przygotowanie planszy. Z tego powodu jest bardzo prostą klasą z niskimi wartościami wszystkich wskaźników. Obliczone wskaźniki CC i WMC świadczą o małym współczynniku skomplikowania samego kodu klasy. Dobrze świadczy to o niezawodności badanej klasy

Wskaźnik RFC jak i CBO pokazują, że dana funkcja korzysta z metod innej klasy. Mimo małego wskaźnika możemy wnioskować, że mimo małej złożoności mogą pojawić się błędy.

LCOM na poziomie 8 wskazuje na nikłą możliwość wystąpienia braku spójności metod. Jest to zdecydowanie dobra wiadomość mówiąca nam o prostocie naszej klasy.

Metryki DIT jak i NOC wskazują nam na dziedziczenie. Jako, że piszemy to w języku Java jesteśmy w stanie powiedzieć, świadczą one o prostocie klasy. Dodatkowo mówią nam o tym, że prawdopodobnie nie jest to klasa uniwersalna co możemy potwierdzić, gdyż posiada ona specjalizację w kierunku obsługi „SudokuBoard”.

SudokuBoard	
nazwa funkcji	**CC**
SudokuBoard	2
SudokuBoard	2
toString	1
equals	6
hashCode	1
propertyChange	6
addActionListener	1
clearBoard	3
getName	1
setName	1
getId	1
setId	1
setAutoCheck	1
isAutoCheck	1
getField	5
get	5
set	5
getRow	4
getColumn	4
getBox	7
checkBoard	1
solveGame	1
getBoard	3
clone	3
Podsumowanie SudokuBoard	
RFC	107
LCOM	164
CBO	4
DIT	1

NOC	0
WMC	66

Tabela 2. Wyniki metryk dla klasy SudokuBoard

Klasa *SudokuBoard*, jest główną klasą modelu. Wskaźniki CC dla poszczególnych funkcji jak i metryka WMC dla całej klasy wskazują nam zdecydowanie skomplikowany kod. Takie liczby mimo małego średniego CC dla funkcji informują nas o możliwości pojawienia się błędów.

Wskaźnik RFC i CBO również pokazują nam, że mamy do czynienia ze skomplikowanym kodem. Wysoki RFC wskazuje na wielokrotne odwołania się do metod innych klas a CBO o ilości klas do których nasza klasa się odwołuje. Te wskaźniki również świadczą o możliwości wystąpienia błędów.

Wskaźnik LCOM zdecydowanie przekracza próg spójności metod pomimo rozdzielenia części odpowiedzialności na inne klasy takie jak SudokuGroup, SudokuField oraz SudokuSolver. Tak wysoki wynik powinien zostać jeszcze bardziej zredukowany poprzez oddelegowanie pewnych metod aby umożliwić prawidłową hermetyzację.

Wskaźnik DIT i NOC jak w poprzedniej funkcji świadczą o małym skomplikowaniu kodu w zakresie dziedziczenia. Jest to jednak równoznaczne z nikła szansą zastosowania tej klasy w innych projektach. Jest to zdecydowanie prawda, gdyż mówimy tutaj o głównej klasie naszego projektu.

SudokuGroup	
nazwa funkcji	**CC**
equals	4
hashCode	1
toString	1
SudokuGroup	3
verify	4
clone	1
Podsumowanie SudokuGroup	
RFC	14
LCOM	4
CBO	1
DIT	1
NOC	3
WMC	14

Tabela 3. Wyniki metryk dla klasy SudokuGroup

Klasa SudokuGroup jest abstrakcyjną klasą po której dziedziczą klasy SudokuBox, SudokuRow oraz SudokuColumn. Jej zadaniem jest weryfikacja danej grupy pól, czy są zgodne z zasadami gry w Sudoku. Wskaźniki CC jak i WMC mówią nam o tym, że nasza klasa jest napisana w sposób nieskomplikowany. To sprawia, że jest to kod łatwy w użytkowaniu

Metryki RFC jak i CBO wskazują na umiarkowane korzystanie z metod innych klas. Może to powodować zmniejszenie niezawodności kodu, jednak w bardzo nieznaczny sposób.

LCOM w przypadku rozpatrywanej klasy jest niewielki. Dzięki temu możemy wnioskować, że jest to zbiór metod o ściśle określonym polu działania co jest zdecydowanie pozytywną kwestią.

Wskaźnik DIT i NOC również w tym przypadku są niewielkie. Jednak uważamy, że w tym wypadku wyższa wartość NOC oznacza niepotrzebne wykorzystanie mechanizmu dziedziczenia.

SudokuBox	
nazwa funkcji	**CC**
SudokuBox	1
clone	1
Podsumowanie SudokuBox	
RFC	1
LCOM	1
CBO	0
DIT	2
NOC	0
WMC	2

Tabela 4. Wyniki metryk dla klasy SudokuBox

Klasa *SudokuBox* jest jedną z 3 klas dziedziczących po klasie *SudokuGroup*. Obliczone metryki CC dla każdej funkcji klasy są zdecydowanie niskie co pokazuje nam również WMC. Jest to oznaką małego skomplikowania funkcji jak i całej klasy.

Wskaźnik RFC i CBO również są niskie co wskazuje na duża niezależność naszej klasy, nikłe odwołania do innych klas w programie świadczą o małej złożoności kodu i mniejszym prawdopodobieństwie wystąpienia błędu.

Niski LCOM pokazuje, że jest to prosta klasa. Wynik taki otrzymujemy ze względu na małą ilość funkcji. Jest to również oznaka prawidłowej hermetyzacji w projekcie.

Wskaźnik DIT jak i NOC są niskie. Otrzymanie takich wartości pokazuje nam, że są to funkcje, których ponowne używanie w innych programach jest niepolecane, jednak również wskazują na małe skomplikowanie samej implementacji kodu.

SudokuColumn	
nazwa funkcji	**CC**
SudokuColumn	1
clone	1
Podsumowanie SudokuColumn	
RFC	1
LCOM	1
CBO	0
DIT	2
NOC	0
WMC	2

Tabela 5. Wyniki metryk dla klasy SudokuColumn

Klasa *SudokuColumn* jest jedną z 3 klas dziedziczących po klasie *SudokuGroup*. Obliczone metryki CC dla każdej funkcji klasy są zdecydowanie niskie co pokazuje nam również WMC. Jest to oznaką małego skomplikowania funkcji jak i całej klasy.

Wskaźnik RFC i CBO również są niskie co wskazuje na dużą niezależność naszej klasy, nikłe odwołania do innych klas w programie świadczą o małej złożoności kodu i mniejszym prawdopodobieństwie wystąpienia błędu.

Niski LCOM pokazuje, że jest to prosta klasa. Wynik taki otrzymujemy ze względu na małą ilość funkcji. Jest to również oznaka prawidłowej hermetyzacji w projekcie.

Wskaźnik DIT jak i NOC są niskie. Otrzymanie takich wartości pokazuje nam, że są to funkcje, których ponowne używanie w innych programach jest niepolecane, jednak również wskazują na małe skomplikowanie samej implementacji kodu.

SudokuRow	
nazwa funkcji	**CC**
SudokuRow	1
clone	1
Podsumowanie SudokuRow	
RFC	1
LCOM	1
CBO	0
DIT	2
NOC	0
WMC	2

Tabela 6. Wyniki metryk dla klasy SudokuRow

Klasa *SudokuRow* jest jedną z 3 klas dziedziczących po klasie *SudokuGroup*. Obliczone metryki CC dla każdej funkcji klasy są zdecydowanie niskie co pokazuje nam również WMC. Jest to oznaką małego skomplikowania funkcji jak i całej klasy.

Wskaźnik RFC i CBO również są niskie co wskazuje na duża niezależność naszej klasy, nikłe odwołania do innych klas w programie świadczą o małej złożoności kodu i mniejszym prawdopodobieństwie wystąpienia błędu.

Niski LCOM pokazuje, że jest to prosta klasa. Wynik taki otrzymujemy ze względu na małą ilość funkcji. Jest to również oznaka prawidłowej hermetyzacji w projekcie.

Wskaźnik DIT jak i NOC są niskie. Otrzymanie takich wartości pokazuje nam, że są to funkcje, których ponowne używanie w innych programach jest niepolecane, jednak również wskazują na małe skomplikowanie samej implementacji kodu.

SudokuField	
nazwa funkcji	**CC**
SudokuField	1
getId	1
setId	1
setBoard	1
addPropertyChangeListener	1
removePropertyChangeListener	1
getValue	1
setValue	3
toString	1
equals	4
hashCode	1
compareTo	2
clone	1
Podsumowanie SudokuField	
RFC	17
LCOM	29
CBO	1
DIT	1
NOC	0
WMC	19

Tabela 7. Wyniki metryk dla klasy SudokuField

Klasa *SudokuField* jest klasą przechowującą wartość pola. Dba również o to, aby wpisywana wartość nie wykraczała poza zakres akceptowalnych cyfr według zasady gry. Dla badanej klasy WMC jak i wszystkie CC wskazują na dość niewielką możliwość wystąpienia błędów. Zasługą tego jest małe skomplikowanie kodu co możemy zaobserwować wyznaczając sobie średnią CC dla metod klasy.

Wskaźnik RFC i CBO pokazują nam, że nasza klasa korzysta w swoich funkcjach z metod innej klasy. Mimo niskich wskaźników może to powodować występowanie błędów pomimo małej złożoności kodu.

Metryka LCOM daje nam możliwość sprawdzenia jak wygląda nasza spójność metod. Według uzyskanych danych jesteśmy w stanie wywnioskować, że poszczególne metody klasy powinniśmy oddelegować innym klasom.

DIT i NOC wskazują, że mamy do czynienia z kodem nieskomplikowanym i pozbawionym łatwej implementacji w innych projektach.

SudokuActionEvent	
nazwa funkcji	**CC**
SudokuActionEvent	1
getRow	1
getCol	1
isInsertedValueCorrect	1
Podsumowanie SudokuActionEvent	
RFC	0
LCOM	0
CBO	0
DIT	4
NOC	0
WMC	4

Tabela 8. Wyniki metryk dla klasy SudokuActionEvent

Klasa *SudokuActionEvent* jest bardzo prostą klasą będącą jedynie nośnikiem informacji dla *ActionListener* przy wywoływaniu zdarzeń w obiekcie nasłuchiwanym. Wskaźniki CC jak i metryka WMC definiują nam tezę, że mamy do czynienia z kompletnie nieskomplikowanym kodem.

Wskaźniki RFC, CBO jak i LCOM wskazują nam na nikłe komplikacje związane z kodem. Możemy również powiedzieć, że metryki te występują w swojej wzorowej postaci.

DIT jak i NOC wskazują na dziedziczenie. NOC mówi nam, że nasza klasa nie ma żadnych dzieci. Wskaźnik DIT natomiast wskazuje na dziedziczenie aż po 4 klasach, co wskazuje nam na większe skomplikowanie kodu, ale również na większą możliwość zastosowania danego zbioru metod w innym projekcie.

FileSudokuBoardDao	
nazwa funkcji	**CC**
FileSudokuBoardDao	1
read	3
write	2
Podsumowanie FileSudokuBoardDao	
RFC	14
LCOM	3
CBO	3
DIT	2
NOC	0
WMC	6

Tabela 9. Wyniki metryk dla klasy FileSudokuBoardDao

Klasa *FileSudokuBoardDao* jest klasą pozwalającą na odczyt i zapis SudokuBoard do i z pliku. Wskaźniki CC jak i WMC wskazują na małe skomplikowania kodu. CC metod nie przekracza 10 więc możemy powiedzieć, że występuje w nich nikłe wystąpienie błędu.

Metryka LCOM jest niska. Nawiązuje to do dobrze zdefiniowanego obszaru działania danej klasy. Jednak wskazuje też to na możliwość dalszego oddelegowywania metod w celu umożliwienia poprawnej hermetyzacji w projekcie.

Wskaźniki takie jak NOC i DIT pokazują nam, że również ta klasa jest praktycznie niemożliwa do zastosowania w innych projektach przy jednoczesnym małym skomplikowaniu kodu.

RFC jak i CBO są wskaźnikami wskazującymi nam na możliwość wystąpienia nieznacznych problemów z powodu korzystania z metod innych klas. Jest to również powodem większej złożoności pisanego kodu.

FileSudokuBoardListDao	
nazwa funkcji	**CC**
FileSudokuBoardListDao	1
read	3
write	2
close	1
Podsumowanie FileSudokuBoardListDao	
RFC	14
LCOM	2

CBO	3
DIT	2
NOC	0
WMC	7

Tabela 10. Wyniki metryk dla klasy FileSudokuBoardListDao

Klasa *FileSudokuBoardListDao* odpowiada za zapisywanie postępów naszej gry do plików. Zapisuje ona 2 widoki, jeden początkowy i drugi wypełniony przez użytkownika. Wyliczony wskaźnik WMC jak i CC dla poszczególnych funkcji w klasie świadczą o nikłym skomplikowaniu kodu. Średnia CC dla każdej funkcji wynosi jedynie 1-2.

Wskaźnik RFC na poziomie 14 wskazuje na umiarkowane wywoływanie metod innych klas. Z tego powodu można powiedzieć, że wskazana klasa jest bardziej złożona od klasy *Difficulty* ale również zdecydowanie mniej od klasy *Window.*

Metryka LCOM dla analizowanego pliku jest mała. Świadczy to o ściśle określonym polu działania klasy wskazującym na poprawną hermetyzację.

CBO na poziomie 3 możemy określić jako małą zależność od innych klas prowadzącą do małego skomplikowania samego kodu jak i mniejszego prawdopodobieństwa wystąpienia błędu.

Wskaźnik DIT jak i NOC pokazują nam, że dana klasa ma małe skomplikowanie samego kodu. Jednak również wskazuje na nikłą możliwość ponownego jej użycia w innych miejscach.

JpaSudokuBoardDao	
nazwa funkcji	**CC**
JpaSudokuBoardDao	1
readAll	3
read	3
write	3
close	2
Podsumowanie JpaSudokuBoardDao	
RFC	31
LCOM	10
CBO	2
DIT	2
NOC	0
WMC	12

Tabela 11. Wyniki metryk dla klasy JpaSudokuBoardDao

Klasa *JpaSudokuBoardDao* jest klasą pozwalającą na odczytywanie i zapisywanie *SudokuBoard* do jak i z bazy danych. Według wskaźników dla CC jak i WMC możemy wnioskować, że analizowana klasa zawiera w sobie nieskomplikowany kod z nikłą możliwością wystąpienia błędów.

Wskaźnik RFC jak i DIT wskazują jednak na znaczne korzystanie z metod innych klas co zwiększa możliwość wystąpienia błędów w naszym programie. Jest to również oznaka, że sam kod można doprowadzić do mniej skomplikowanej formy. Powodem tak znacznej liczby jest konieczność obsługi bazy danych, korzystając z ściśle określonych metod w innych klasach.

Metryka LCOM wskazuje na niedostateczne oddelegowanie funkcji do innych klas. Jest to negatywne i wskazuje na brak pożądanej własności klasy, jaką jest spójność.

Dla wskaźników NOC i DIT otrzymaliśmy zdecydowanie niskie wyniki. Dzięki temu możemy wskazać, że owa klasa nie jest przeznaczona do implementacji w innych projektach a jej złożoność jest niewielka.

SudokuBoardDaoFactory	
nazwa funkcji	**CC**
SudokuBoardDaoFactory	1
getInstance	2
getSudokuBoardFileDao	1
getSudokuBoardJdbcDao	1
getSudokuBoardListFileDao	1
Podsumowanie SudokuBoardDaoFactory	
RFC	4
LCOM	9
CBO	1
DIT	1
NOC	0
WMC	6

Tabela 12. Wyniki metryk dla klasy SudokuBoardDaoFactory

Klasa *SudokuBoardDaoFactory* jest bardzo prostą implementacją wzorców projektowych *Singleton* oraz *Factory*. Przygotowuje ona i zwraca żądany obiekt, który pozwala na zapisywanie *SudokuBoard* w odpowiednej formie. CC jak i metryka WMC należą zdecydowanie do grupy niskich wyników. Wynika z tego mała złożoność klasy ale również nikła możliwość wystąpienia błędów.

Wskaźniki takie jak RFC i CBO wskazują na minimalne korzystanie z metod innych klas. Dzięki temu możemy powiedzieć, że możliwość wystąpienia błędów w programie jest zmniejszona jak również poziom komplikacji w kodzie jest niewielki.

LCOM pokazuje nam, że powinniśmy w celu uzyskania lepszego kodu oddelegować kilka metod do innych klas.

NOC i DIT w badanym przez nas fragmencie programu występuje w praktycznie 0 wartości. Wskaźnik DIT jest większy z powodu implementacji projektu w języku Java. Wskazuje to na minimalną złożoność klasy, ale również na minimalną możliwość wykorzystania danego pliku w innych projektach.

BacktrackingSudokuSolver	
nazwa funkcji	**CC**
equals	4
hashCode	1
toString	1
randomTmpArray	2
fillGaps	15
solve	7
Podsumowanie BacktrackingSudokuSolver	
RFC	31
LCOM	18
CBO	1
DIT	2
NOC	0
WMC	30

Tabela 13. Wyniki metryk dla klasy BacktrackingSudokuSolver

Klasa *BacktrackingSudokuSolver* jest implementacją wzorca projektowego *Strategy* i korzysta z interfejsu *SudokuSolver*. Jest klasą, która zapewnia *SudokuBoard* możliwość rozwiązywania oraz sprawdzania łamigłówek. Metryki takie jak CC i WMC wskazują nam niestety na umiarkowanie złożoność kodu ale również dla poszczególnych metod średnie ryzyko wystąpienia błędów.

Metryka LCOM pokazuje nam, że powinniśmy spróbować przenieść większa ilość metod do innych klas w celu zwiększenia określoności pola działania naszej klasy.

Wskaźnik RFC i CBO wskazują na duże wykorzystanie metod innej klasy co może skutkować występowaniem błędów a także zwiększeniem złożoności naszego kodu.

NOC i DIT natomiast pokazują, że jest to kod sprecyzowany do naszego programu z niewielką możliwości implementacji w dalszych projektach. Jednak zaletą jest małe skomplikowanie kodu co daje zmniejszoną możliwość wystąpienia błędów.

Window	
nazwa funkcji	**CC**
Window	4
initialize	4
getTextField	1
toggleDifficultyVisibility	1
setBoxesColors	11
disableNonZeroTextFields	4
acceptDifficutly	4
solve	1
tipsSwitched	1
giveUp	1
actionPerformed	3
checkGame	6
setLang	5
saveGameToFile	2
loadGameFromFile	6
saveGameToDatabase	3
loadGameFromDatabase	7
Podsumowanie Window	
RFC	184
LCOM	102
CBO	7
DIT	1
NOC	0
WMC	64

Tabela 14. Wyniki metryk dla klasy Window

Klasa Window jest jedyną klasą interfejsu graficznego. Jest to bardzo złożona klasa, która odpowiada za wszystkie interakcje z oknem aplikacji. WMC jednoznacznie wskazuje na to, że mamy do czynienia ze złożoną klasą. Wskaźnik CC jednak pokazują nam że ryzyko wystąpienia błędów jest niewielkie.

Metryka LCOM niestety dobitnie pokazuje, że powinniśmy oddelegować pracę wielu metod do innych klas w celu uzyskania dobrej hermetyzacji kodu.

Wskaźniki takie jak DIT i NOC tak jak w sporej ilości poprzednich klas wskazują na mała złożoność kodu przy jednoczesnej wąskiej możliwości implementacji danego kodu w innych programach. Jest to w pełni zrozumiałe patrząc na jej złożoność oraz specjalizację w obsłudze interfejsu napisanego pod konkretną bibliotekę (nasz model).

RFC i CBO wskazują na bardzo dużą złożoność danej części programu. Z metryk wynika, że korzystamy z metod wielu innych klas co niewątpliwie zwiększa możliwość wystąpienia awarii, co nie jest pożądane.

Przewidywane prawdopodobieństwo wystąpienia błędu na podstawie wartości liczby π

Wzór z które skorzystaliśmy to:

$$\pi = \frac{1}{1 + e^{-(-3.97 + .464NAI + 1.47OCMEC + 1.06DIT)}}$$

gdzie:
NAI – liczba atrybutów w klasie
OCMEC – liczba innych klas korzystających z metod badanej klasy
DIT – głębokość zakorzenienia w drzewie dziedziczenia

Wartość zbliżona do 1, oznacza klasę wysoce podatną na błędy, a zbliżona do 0, praktycznie odporną na występowanie błędów.

Klasa	NAI	OCMEC	DIT	π
Difficulty	2	1	1	0.375
SudokuBoard	8	2	1	0.977
SudokuBox	0	1	2	0.406
SudokuColumn	0	1	2	0.406
SudokuRow	0	1	2	0.406
SudokuGroup	1	1	1	0.274
SudokuField	5	2	1	0.913
SudokuActionEvent	3	2	4	0.99
FileSudokuBoardDao	2	1	2	0.634
FileSudokuBoardListDao	2	1	2	0.634
JpaSudokuBoardDao	2	1	2	0.634
SudokuBoardDaoFactory	0	1	1	0.192
BacktrackingSudokuSolver	1	1	2	0.521
Window	12	0	1	0.935

Tabela 15. Wyniki metryk błędów dla klas

Patrząc na wartości tych wskaźników widzimy, że większość klas ma stosunkowo wysoką podatność na wystąpienie błędu. Za tak wysokie wartości częściowo odpowiada fakt, że przy programowaniu w Javie minimalny DIT wynosi 1, jako że wszystkie obiekty dziedziczą po klasie Object. Gdyby obniżyć DIT o 1, to większość z wyliczonych wartości π byłaby o połowę mniejsza.

Dodatkowo, w przypadku *SudokuActionEvent* dziedziczymy z klasy zewnętrznej biblioteki, na której proces dziedziczenia się nie kończy, co ma duży wpływ na wartość π w tym przypadku. Jedynymi klasami które prawdziwie mają duże szansa na wystąpienie błędu to *Window* oraz *SudokuBoard*, co pokrywa się z obliczanymi wcześniej wskaźnikami. Taką samą sytuacją jest objęty także *SudokuField*.

W ostatnim przypadku wynika to z dużej ilości atrybutów obiektów tej klasy potrzebnych do komunikacji między klasami (*event listener*) jak i również tworzenia logów. Dodatkowo powiązania między *SudokuField* oraz *SudokuBoard* w ramach zapisywania tych dwóch klas w bazie danych powoduje możliwość wystąpienia błędów z powodu wykorzystania zewnętrznej biblioteki do obsługi tego procesu. Niemniej oznacza to oddelegowania odpowiedzialności za wszelkie błędy na właśnie te biblioteki, dlatego tak wysoka wartość π w tym przypadku jest w pełni uzasadniona.

Podsumowanie

PODSUMOWANIE	
LOC [Lines of Code]	1581
NCNB [Lines of Code]	1227
EXEC [Executable statements]	891
CP [Comment percentage]	5.1 %
CCavg [average Cyclomatic Complexity]	2.41
WMCavg [average Weighted method per class]	13.21

Tabela 16. Wyniki metryk dla całego projektu

Podsumowując wszystkie poprzednie tabele jak i wnioski jakie z nich wyciągnęliśmy możemy uznać, że kod programu jest względnie odporny na błędy. Jednak warunkiem koniecznym do spełnienia poprzedniego zdania jest odpowiedniego pokrycia kodu testami, co w naszym przypadku jest spełnione. Większość naszego programu została napisana za pośrednictwem nie złożonego i odpowiednio łatwego w utrzymaniu kodu. Średnio na klasę wskaźnik WMC wskazywał na małą złożoność kodu przy równie małym ryzyku wystąpienia błędu związanego z metryką CC na każdą metodę klasy. Wskaźnik LCOM dla większości naszych zbiorów metod pokazał nam, że w celu lepszej hermetyzacji powinniśmy oddelegować poszczególne elementy klas do innych. CBO dla wszystkich naszych pomiarów było bardzo małe. Dzięki temu możemy stwierdzić, że nasz kod jest właściwie uodporniony na błędy związane z importowaniem. Wskaźnik RFC jednak wskazał nam, że większość naszych klas korzysta z więcej niż 10 metod należących do innych obiektów, co może powodować ryzyko występowania błędów a także zwiększenie złożoności. Minimalny wskaźnik DIT dla wszystkich badanych plików wynosi minimum 1. Jest to wynikiem paradygmatu języka Java, gdzie każda klasa dziedziczy po *Object.* Z tego powodu złożoność każdego przez nas pisanego kodu wzrasta. W naszym projekcie wskaźniki NOC praktycznie wszędzie wynosi 0. Dzięki

temu możemy powiedzieć, że ze względu na dziedziczenia złożoność naszego kodu jest niewielka co zmniejsza ryzyko występowania awarii. Problemem jednak może okazać się małe pokrycie komentarzami bo jest to zaledwie 5% linii projektu. Jest to aspekt, który odpowiada za znaczne utrudnienie jego zrozumienia. Kolejnym aspektem nad którym się pochyliliśmy jest niska przenoszalność kodu. Wszystkie nasze klasy były pisane z myślą tylko o danym projekcie. Z tego powodu jakakolwiek implementacja poszczególnych plików w innych pracach jest znacznie ograniczona. Wyjątkowo dużym problemem naszego programu są klasy takie jak *SudokuBoard* i *Window*. Owe zbiory metod powinny zostać podzielone na wiele innych obiektów z zawężoną odpowiedzialności i ściśle określonym polem działania. Wskazują na to przede wszystkim wskaźniki takie jak: π, RFC, LCOM a także WMC. Dzięki wyżej opisanemu działaniu moglibyśmy znacząco zmniejszyć możliwość wystąpienia błędów. Specjalnie dla klasy *SudokuBoard* moglibyśmy wyeliminować dziedziczenie klas pomiędzy *SudokuGroup* i jej dziećmi *SudokuBox, SudokoRow, SudokuColumn.* Również dobrym sposobem działania byłoby rozdzielenie logiki w *SudokuBoard* od danych. Dla klasy *Window* dobrą praktyką byłoby pogrupowanie rodzajów odpowiedzialności i stworzenie dla nich odpowiednich klas.