

**WYDZIAŁ INŻYNIERII METALI I INFORMATYKI PRZEMYSŁOWEJ**

KATEDRA INFORMATYKI STOSOWANEJ I MODELOWANIA

Projekt dyplomowy

*Projekt i implementacja aplikacji przeznaczonej do edycji diagramów klas UML*

*Design and implementation of an application for editing UML class diagrams*

Autor: *Filip Górnicki*

Kierunek studiów: *Informatyka Techniczna*

Opiekun projektu: *dr inż. Gabriel Rojek*

Kraków, 2023

[1. Wstęp 4](#_Toc121762306)

[2. Przegląd istniejących rozwiązań w zakresie konstrukcji diagramów klas UML 6](#_Toc121762307)

[2.1. Visual Paradigm 6](#_Toc121762308)

[2.2. StarUML 8](#_Toc121762309)

[2.3. draw.io 9](#_Toc121762310)

[3. Cel pracy 11](#_Toc121762311)

[4. Projekt aplikacji 13](#_Toc121762312)

[4.1. Diagram klas UML 13](#_Toc121762313)

[4.2. Diagramy sekwencji UML 17](#_Toc121762314)

[5. Implementacja rozwiązania 22](#_Toc121762315)

[5.1. Rysowanie relacji dziedziczenia 22](#_Toc121762316)

[5.2. Zapis obiektu klasy wyliczeniowej lub interfejsu do pliku .ini 22](#_Toc121762317)

[5.3. Wyznaczanie oraz aktualizacja pozycji środka kolumny 23](#_Toc121762318)

[5.4. Wykonywanie zrzutu ekranu 24](#_Toc121762319)

[6. Testowanie i badania 26](#_Toc121762320)

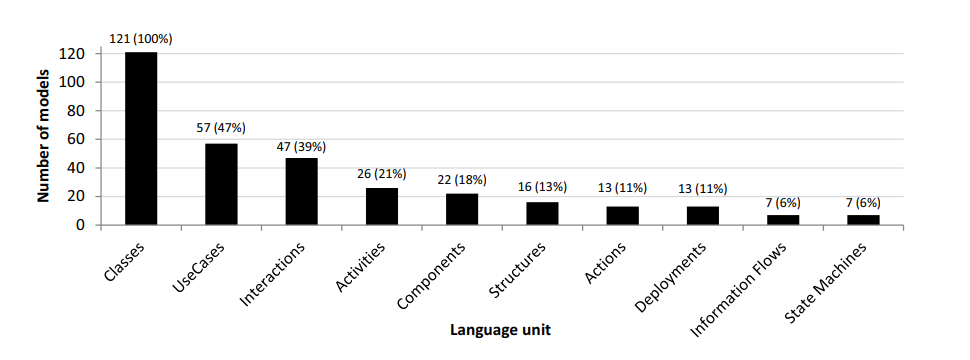
[7. Podsumowanie 28](#_Toc121762321)

[1 Bibliografia 28](#_Toc121762322)

# Wstęp

Jednym z podstawowych czynników wpływających na powodzenie projektu informatycznego jest sprawna komunikacja między zespołami, nie tylko programistów, ale również na linii informatyka – biznes. W przypadku niewielkiego przedsięwzięcia, wystarczający może okazać się język naturalny, jednak, mając do czynienia z projektami na wielką skalę, w które zaangażowana jest duża liczba osób z różnych środowisk, konieczne jest wprowadzenie nowego sposobu wymiany informacji. Na bazie wniosków z kilkunastu lat doświadczeń zebranych przez ludzi zajmujących się najbardziej popularnym paradygmatem programowania – programowaniem obiektowym, powstał formalny, zwięzły i wyczerpujący język modelowania, używany do projektowania systemów oraz oprogramowania – UML (ang. *Unified Modeling Language*).

Specyfikacja języka UML określa wiele typów diagramów. Do najpopularniejszych należy diagram przypadków użycia (ang. *use case*), którego podstawowym zadaniem jest ukazanie funkcjonalności projektowanego systemu lub oprogramowania. Istotnym diagramem jest także diagram sekwencji, który służy do ukazania kolejności interakcji pomiędzy obiektami przy jednoczesnym uwzględnieniu przesyłanych komunikatów. Najważniejszym i zarazem najczęściej wykorzystywanym diagramem jest jednak diagram klas [1]. Pozwala on na zadeklarowanie klas wraz z ich polami i metodami, a co najważniejsze, przedstawić relacje pomiędzy nimi, takie jak dziedziczenie jednej klasy przez drugą lub implementowanie interfejsu przez klasę. Te trzy diagramy: przypadków użycia, klas oraz sekwencji to podstawowe diagramy, konieczne do pełnego zamodelowania systemu.



Rysunek 1. Częstość wykorzystywania poszczególnych typów diagramów UML.  
Źródło: [1]

Podstawowym obiektem na diagramie klas jest, symbolizujący klasę prostokąt podzielony liniami na trzy sekcje. Pierwsza sekcja zawiera nazwę klasy, druga – pola, a trzecia – metody. Język UML pozwala na zdefiniowanie czterech typów widoczności pól i metod:

* „+” – dostęp publiczny,
* „#” – dostęp chroniony,
* „-„ – dostęp prywatny,
* „~” – dostęp w obrębie pakietu.

Istnieje również rozróżnienie dla typów klas oraz metod abstrakcyjnych, których nazwy zapisywane są kursywą.

Diagram klas UML, poza graficznym przedstawieniem wykorzystywanych w projekcie klas, powinien również definiować sposób implementacji kodu będącego odzwierciedleniem tego diagramu. W specyfikacji języka UML wyróżnionych zostało sześć typów relacji między klasami, które można jednoznacznie przyrównać do różnych konstrukcji w wielu językach programowania:

* zależność (ang. *dependency*) – relacja, gdy obiekt jednej klasy jest wykorzystywany w metodzie obiektu drugiej klasy, najczęściej jako parametr,
* asocjacja (ang. *association*) – relacja mogąca zastąpić atrybut klasy; logiczne połączenie dwóch klas pozwalające na korzystanie jednej klasie z obiektów drugiej klasy poprzez posiadanie atrybutu referencji do niej,
* agregacja częściowa (ang. *aggregation*) – specjalny typ asocjacji; określa, że jedna klasa jest częścią drugiej. Klasa będąca właścicielem zawiera jako pole drugą klasę,
* agregacja całkowita (ang. *composition*) – specjalny typ asocjacji; w odróżnieniu od agregacji częściowej, agregacja całkowita określa, że obiekt będący właścicielem decyduje o cyklu życia obiektu podrzędnego, to znaczy, że obiekt podrzędny nie istnieje bez obiektu macierzystego (właściciela), przykładem kompozycji może być związek klasy zewnętrznej z niestatyczną klasą wewnętrzną,
* dziedziczenie (ang. *inheritance*) – relacja, w której jedna klasa przejmuje funkcjonalności drugiej klasy. W języku Java istnieje słowo kluczowe *extends* określające dziedziczenie jednej klasy przez drugą,
* realizacja (ang. *realization*) – relacja między klasą a interfejsem symbolizująca implementacje funkcjonalności interfejsu przez klasę. Analogicznie dla przykładu dziedziczenia, w języku Java istnieje słowo kluczowe *implements* informujące o takiej relacji.

# Przegląd istniejących rozwiązań w zakresie konstrukcji diagramów klas UML

W sieci dostępne są rozwiązania pozwalające na tworzenie diagramów klas UML. Powstały projekty komercyjne, ukierunkowane na projektowanie oprogramowania w języku UML, działające na zasadach comiesięcznej subskrypcji lub po jednorazowym wykupieniu licencji. Najczęściej, programy te oferują największe możliwości edycji diagramów klas UML, a także pozwalają na pracę *on-line* lub *off-line.* Kolejnym wyborem są edytory do tworzenia diagramów ogólnego przeznaczenia, płatne i darmowe, gdzie wśród wielu opcji, dostępne są takie, pozwalające na projektowanie prostych diagramów klas UML. Najczęściej są to aplikacje działające w przeglądarkach, posiadające ograniczone możliwości edycji, a do ich działania konieczne jest połączenie z Internetem. Dodatkową możliwością są projekty osób prywatnych, którzy udostępniają je dla użytkowników, łącząc najczęściej idee poprzednich rozwiązań. Cechują się o wiele mniejszymi możliwościami w porównaniu do produktów komercyjnych oraz niską odpornością na błędy użytkownika. Dostępne są wersje desktopowe lub wymagające dostępu do Internetu.

## Visual Paradigm

Aplikacja Visual Paradigm to narzędzie do tworzenia diagramów UML. Jest jednym z najbardziej rozbudowanych tego typu programów, pozwalającym na modelowanie diagramów UML 2.x, SysML oraz Entity Relationship Diagram. Wykorzystywany jest przez największe firmy, uniwersytety oraz sektory rządowe.

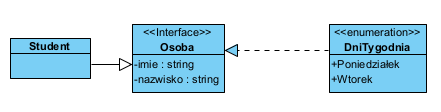
Visual Paradigm oferuje użytkownikowi wiele możliwości w projektowaniu diagramów klas. Poza podstawowym typem klasy, interfejsem czy klasą wyliczeniową (ang. *enumeration*), pozwala na utworzenie elementów: Bean, prymityw, pakiet, agent czy serwis REST. Każdy obiekt posiada funkcjonalności przenoszenia po ekranie przy pomocy myszy (ang. *drag and drop*) oraz określania wielkości. Otwierając okno ze specyfikacją danej klasy, użytkownik może określić jej typ, widoczność oraz nazwę, a także dodać pola i metody.

Istotą diagramów klas UML są relacje między obiektami. W programie odnaleźć można każdy ich typ – od zależności po dziedziczenie. Definiując relacje między klasami, możliwe jest określenie liczności, nazw przesyłanych atrybutów, a w przypadku relacji zależności – również stereotypów.

Sposób wprowadzania danych przez użytkownika został specjalnie przygotowany pod względem odporności na błędy. Architekci projektujący aplikację Visual Paradigm przygotowali gotowe pola wyboru (ang. *combo box*) z odpowiednimi wartościami, jednocześnie pozwalając niekiedy na ich lekkie modyfikacje, pozwalające osobom zainteresowanym dostosować program do swoich potrzeb.

Program istotnie wyróżnia się na tle podobnych produktów funkcjonalnością pozwalającą na generowanie diagramów klas na podstawie wcześniej przygotowanego kodu źródłowego Javy oraz C++, oraz na operację odwrotną – wygenerowanie kodu z utworzonego diagramu. Dla innych typów diagramów, przygotowano podobne możliwości generowania baz danych, API czy kodu Hibernate.

Tak rozbudowana aplikacja komercyjna pozwala na niemal nieograniczone możliwości tworzenia i edycji diagramów UML. Problemem dla osób i firm chcących w pełni skorzystać z funkcjonalności Visual Paradigm może okazać się koszt wykupienia licencji. Firma Visual Paradigm International Ltd., będąca właścicielem aplikacji, oferuje system subskrypcji lub jednorazowego wykupienia licencji. Cena pełnej wersji – Enterprise - została ustalona na 1999$ lub 89$ w modelu subskrypcji, natomiast najtańsza wersja Modeler – 99$ lub 6$/miesiąc. Dodatkowo, pomimo wielu udogodnień oraz zabezpieczeń, Visual Paradigm nie posiada ochrony przeciwko tworzeniu relacji niemających sensu. Możliwe jest błędne połączenie relacją dziedziczenia klasy z interfejsem lub implementacji interfejsu przez klasę wyliczeniową (rysunek 2).



Rysunek 2. Błędne relacje w programie Visual Paradigm  
Źródło: opracowanie własne

Mając na uwadze doświadczonych projektantów systemów, funkcjonalność ta nie byłaby potrzebna, jednak początkujący użytkownicy, niezaznajomieni w pełni z zasadami języka UML, skorzystaliby na wprowadzeniu do programu takich zabezpieczeń, które ograniczyłby ilość błędów w projekcie. Za wadę uznać można również powolne uruchamianie się aplikacji (którą wytłumaczyć można faktem, że tak bogata w funkcjonalności aplikacja potrzebuje czasu na przygotowanie) czy konieczność połączenia z Internetem w celu sprawdzenia licencji użytkownika co wyklucza możliwość pracy całkowicie off-line. Uwagę należy zwrócić także na tworzone przez Visual Paradigm pliki służące za kopię zapasową podczas zapisywania projektu, które nie są później automatycznie usuwane.

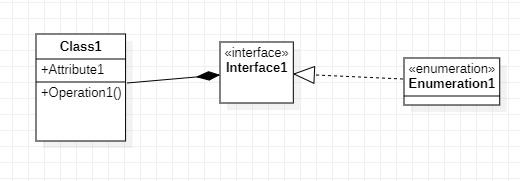
## StarUML

StarUML jest platformą typu *open-source*, umożliwiającą modelowanie systemów przy pomocy języka UML, SysML, ERD oraz DFD. Przy wsparciu społeczności, projekt udało się rozszerzyć o nowe komponenty. Tak jak w przypadku Visual Paradigm, StarUML gwarantuje dostęp do wszystkich podstawowych oraz zaawansowanych klas, które oferuje język UML.

Aplikację wyróżnia okno prezentujące strukturę projektu w formie drzewiastej. Jako swoje podstawowe funkcjonalności, program umożliwia dodawanie oraz przesuwanie po ekranie klas wraz z relacjami, a po naciśnięciu na obiekt, wyświetlane są jego parametry. Okno z właściwościami obiektu zawiera dużą liczbę informacji oraz możliwości edycji, które przedstawione są w postaci ikon, pól wyboru i pól tekstowych, których niewielkie rozmiary sprawiają jednak, że są nieczytelne. Oddzielne usuwanie klasy czy relacji z okna lub drzewa projektu, w przeciwieństwie do Visual Paradigm (podczas usuwania graficznej reprezentacji elementu automatycznie wyświetlany jest komunikat z zapytaniem o usunięcie również z drzewa projektu), nie zostało w żaden sposób ułatwione. Zaniedbanie kontrolowania usuwanych elementów może doprowadzić do nieczytelności diagramu. Co więcej, możliwe jest jedynie dodanie pól i metod do interfejsu w widoku drzewa projektu, natomiast w reprezentującym ten interfejs prostokącie, nie zachodzą zmiany (rysunek 3). Sama funkcjonalność dodawania jest nieintuicyjna ze względu na brak sortowania możliwych do dodania segmentów ze względu element, do którego miałby być dodany. Tak jak w przypadku Visual Paradigm, StarUML nie zabezpiecza użytkownika przed połączeniem obiektów nieprawidłową relacją (rysunek 4). Wyjątkiem jest jedynie relacja realizowania, gdzie, w przypadku niepołączenia jej z interfejsem, wyświetlany jest komunikat o błędzie. Możliwe jest jednak połączenie tą relacją interfejsu i klasy wyliczeniowej.



Rysunek 3. Dodanie pola i metody do interfejsu w aplikacji StarUML  
Źródło: opracowanie własne



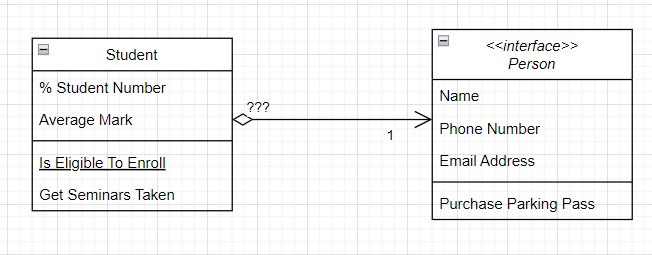
Rysunek 4. Błędne relacje w programie StarUML  
Źródło: opracowanie własne

Dzięki zaangażowaniu społeczności zrzeszonej wokół projektu StarUML, program doczekał się wielu ciekawych rozszerzeń, takich jak generowanie kodu w różnych językach programowania na podstawie diagramów, a także operacji odwrotnej. Aplikacja jest dostępna w wersji darmowej bez limitu czasowego, a także w wersjach płatnych przeznaczonych dla osób prywatnych, szkół oraz firm.

## draw.io

Aplikacja draw.io jest przykładem darmowego programu działającego *on-line* w przeglądarce internetowej, do tworzenia diagramów ogólnego typu, jednak posiadająca funkcjonalność projektowania diagramów UML.

Aplikacja posiada podstawowe możliwości modelowania systemów przy użyciu języka UML. Użytkownik może dodawać, usuwać oraz przemieszczać po ekranie klasy oraz łączące je relacje, określać widoczność pól i metod, krotności oraz role. Z racji swojego ogólnego przeznaczenia, draw.io nie gwarantuje żadnych zabezpieczeń przed błędami logicznymi spowodowanymi przez użytkownika, takimi jak niepoprawne relacje, typy widoczności czy krotności (rysunek 5).



Rysunek 5. Błędne relacje, widoczność oraz krotność w aplikacji draw.io  
Źródło: opracowanie własne

Pomimo swojej dużej elastyczności oraz szerokim możliwościom w tworzeniu diagramów, aplikacja posiada gotowe komponenty, których wybranie powoduje konieczność dodawania kolejnych, często niemożliwych później do połączenia ze sobą, na przykład, chcąc dodać krotność do asocjacji skierowanej, konieczne jest dodanie pola tekstowego, które w przypadku przeniesienia strzałki relacji nie zostanie przeciągnięte razem z nią.

# Cel pracy

Język UML jest potężnym narzędziem wykorzystywanym do modelowania procesu tworzenia oprogramowania, wykorzystywana w 72% przypadków, w których firmy decydują się na wsparcie projektowania notacją graficzną. Ponad połowa pracowników korzystających z tego języka, jako główną zaletę przedstawia łatwiejsze zrozumienie oprogramowania i szybsze wykrywanie błędów [2].

W zaprezentowanych w rozdziale 2, dostępnych już rozwiązaniach, odnaleźć można powtarzającą się wadę – aplikacje nie posiadają zaimplementowanych zabezpieczeń uniemożliwiających użytkownikowi utworzenia niepoprawnych relacji między klasami. Błąd niezrozumienia relacji jest najczęściej spotykanym i sygnalizowanym problemem wśród początkujących [3],[4]. Chcąc wyjść naprzeciw temu problemowi, zdecydowano się na zaprojektowanie i zaimplementowanie programu do tworzenia oraz edycji diagramów klas UML uwzględniającego jednocześnie kwestię zapobiegania tworzeniu błędnych połączeń między klasami.

Pomimo wprowadzonych udogodnień, nie jest możliwe wyeliminowanie wszystkich błędów. Ustalenie poprawności niektórych relacji nie jest możliwe odgórnie, bez zagłębienia się w specyficzny, indywidualny projekt oraz jego logikę. Przykładowym problemem może być kontrola zasadności występowania związku agregacji czy kompozycji. Mając informacje jedynie z diagramu klas, niemożliwe jest zdefiniowanie, która relacja jest poprawna. W takich sytuacjach konieczne jest powołanie się na doświadczenie projektanta systemu lub dokumentację standardu UML.

Program posiada standardowe dla opisywanych w rozdziale drugim aplikacji funkcje pozwalające na projektowanie diagramów klas UML. Użytkownik jest w stanie utworzyć trzy typy klas: standardową, wyliczeniową lub interfejs, a także zmieniać ich położenie na ekranie. Do definiowania relacji między elementami diagramu, program wyposażony jest w sześć typów związków przedstawionych we wstępie: zależności, asocjacji, asocjacji skierowanej, agregacji, kompozycji, dziedziczenia oraz realizacji. Sprawdzane są wtedy zdefiniowane w programie reguły poprawności, które zatwierdzają lub odrzucają próbę powiązania elementów. Użytkownik może usuwać klasy, interfejsy oraz relacje, a także modyfikować je w specjalnym oknie. Możliwa jest edycja:

* nazwy, pól i metod klasy lub interfejsu,
* krotności w relacji.

Istotną funkcjonalnością jest możliwość odtworzenia zapisanej wcześniej pracy. Konieczny staje się wtedy zapis potrzebnych danych do pliku, który następnie będzie mógł być otwarty, aby wczytać potrzebne informacje. Program posiada możliwość zapisu oraz odczytu danych z pliku z rozszerzeniem .ini, który jest typem konfiguracyjnym systemu Windows. Jego istotną zaletą jest sposób, w jaki zapisywane są dane, mianowicie pliki .ini wykorzystują strukturę podziału na sekcje, do których przypisywane są parametry mapowane jako klucz – wartość (rysunek 6).

Obraz zawierający tekst

Opis wygenerowany automatycznie

Rysunek 6. Struktura pojedynczej sekcji pliku konfiguracyjnego .ini  
Źródło: opracowanie własne

Wprowadzono również możliwość wykonania i zapisania zrzutu ekranu aplikacji.

Zasadniczym celem programu jest wprowadzenie zabezpieczeń przed popełnianiem błędów niepoprawnego łączenia klas relacjami. Uniemożliwiono jedynie dodawanie takich błędnych powiązań, które można określić jako bezkompromisowe, jednoznaczne. Aplikacja udaremnia możliwość:

* powiązania asocjacją inaczej niż klasa z klasą,
* powiązania agregacją lub kompozycją klasy z interfejsem, gdzie interfejs jest klasą nadrzędną (całością) oraz klasy wyliczeniowej z innym obiektem
* powiązania asocjacji skierowanej z klasą wyliczeniową tak, że grot relacji wskazuje na klasę wyliczeniową
* powiązania zależnością klasy wyliczeniowej lub interfejsu z klasą wyliczeniową, lub interfejsem,
* powiązania dziedziczeniem dwóch różnych typów klas oraz dwóch klasy wyliczeniowej,
* powiązania realizacji w sposób inny niż między klasą a interfejsem tak, że grot relacji wskazuje na interfejs.

Pozostałe relacje są poprawne albo niemożliwe do ograniczenia, ponieważ związane są z logiką modelowanego programu. W przypadku wystąpienia jednego z przedstawionych błędów, aplikacja nie utworzy relacji, a także wyświetli na ekranie informację o błędzie i poda możliwe połączenia, dostępne dla wybranej relacji.

# Projekt aplikacji

Java jest jednym z najpopularniejszych języków programowania wysokopoziomowego. W myśl zasady *Write Once, Run Anywhere*, gwarantuje on pełną przenoszalność kodu źródłowego i wynikowego niezależnie od architektury systemu operacyjnego. Konieczne jest jedynie zainstalowanie JVM, czyli wirtualnej maszyny Java, której zadaniem jest interpretowanie pośredniego kodu źródłowego – bajtkodu oraz uruchomienie programu w bezpiecznym, odizolowanym środowisku maszyny. Język Java charakteryzuje również wysoka odporność na błędy poprzez zastosowanie mechanizmu wyjątków, ścisłe określenie formy bajtkodu oraz kontrolę pamięci. Nadzorowanie jej odbywa się na wielu poziomach: dostępu do obiektów przez referencje czy automatycznego usuwanie niewykorzystywanych już obiektów przez *Garbage Collector*.

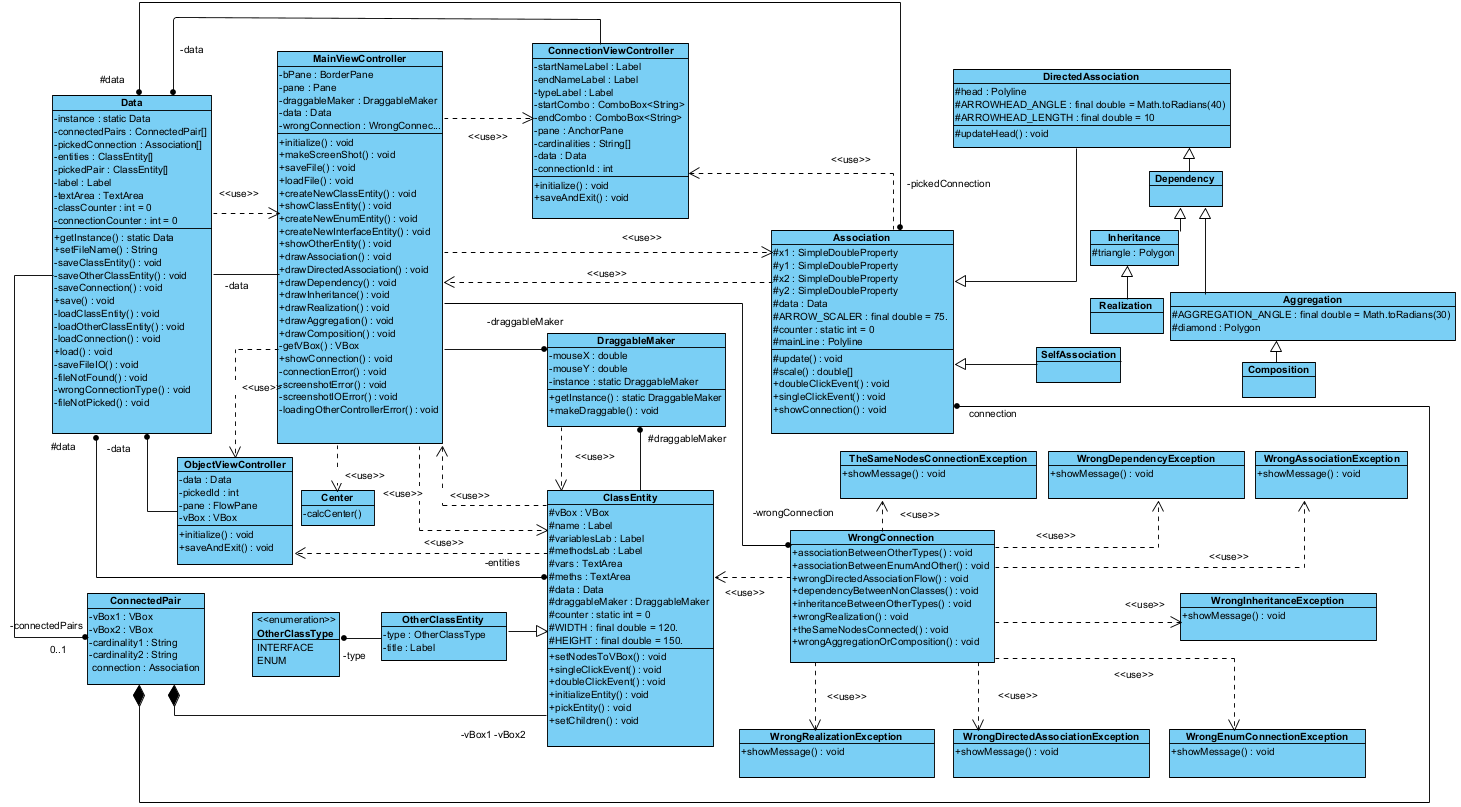
Na przestrzeni lat powstało wiele bibliotek do tworzenia graficznych interfejsów użytkownika w Javie. Najnowszą oraz rozwijaną biblioteką jest JavaFX, która wykorzystuje wszystkie dobrodziejstwa Javy oraz swojego poprzednika – biblioteki Swing, a także sama wprowadza nowe udogodnienia, takie jak:

* możliwość korzystania z kaskadowych arkuszy stylu CSS oraz XML,
* pełne wsparcie dla wzorca projektowego MVC – podział na warstwę logiczną, warstwę widoku oraz spajający je kontroler,
* dodanie nowych kontrolek obsługujących między innymi multimedia oraz strony internetowe,
* wsparcie tworzenia warstwy widoku przy użyciu narzędzia SceneBuilder,
* rozwijanie wsparcia dla systemów Android oraz iOS.

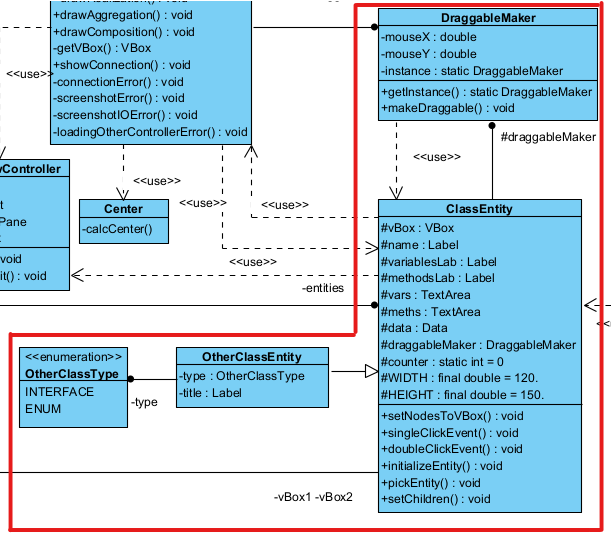
Mając na uwadze te i inne zalety języka Java, zdecydowano na wykonanie aplikacji właśnie w JavieFX.

## Diagram klas UML

Projekt aplikacji do tworzenia diagramów klas UML z zabezpieczeniami przed niepoprawnym tworzeniem związków między obiektami został zaprezentowany na rysunku 7.



Rysunek 7. Diagram klas UML przedstawiający strukturę programu  
Źródło: opracowanie własne

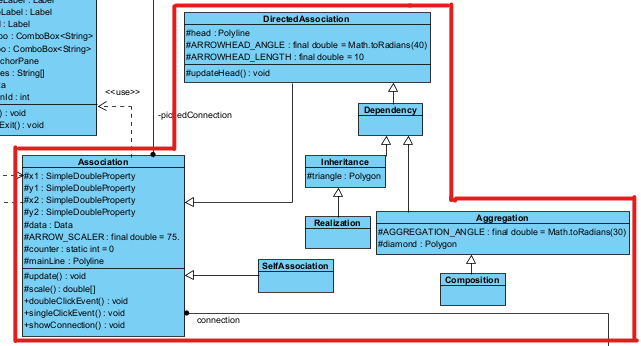


Rysunek 8. Wycinek diagramu klas prezentujący klasy odpowiadające za tworzenie oraz obsługę elementów na diagramie  
Źródło: opracowanie własne

Rysunek 8 przedstawia klasy ClassEntity oraz OtherClassEntity, które odpowiadają za reprezentację klas: ClassEntity – klasa standardowa, OtherClassEntity – interfejs lub enum, gdzie odpowiedni typ określny jest przy wsparciu klasy wyliczeniowej OtherClassType. Klasa OtherClassEntity dziedziczy po klasie ClassEntity wszystkie pola i metody, nadpisując niektóre, aby dostosować je do potrzeb klasy. Pole counter wspomaga nadawanie identyfikatorów poszczególnych obiektów, a WIDTH oraz HEIGHT określają rozmiary wyświetlanej na ekranie kolumny VBox przedstawiającej tworzoną na diagramie klasę. W skład kolumny wchodzą:

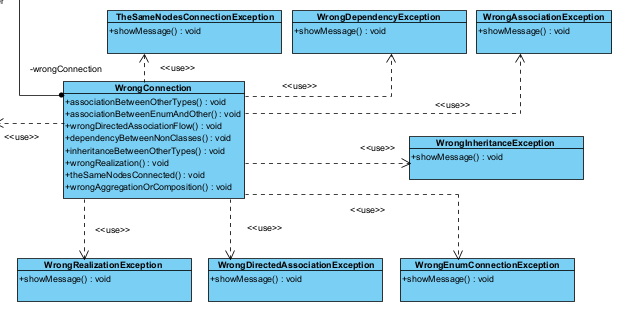
* etykieta (ang. *label*) type podająca typ klasy,
* etykieta name podająca nazwę klasy,
* etykieta variablesLab sygnalizująca sekcję z polami klasy,
* pole tekstowe (ang. text area) vars wyświetlające pola klasy,
* etykieta methodsLab sygnalizująca sekcję z metodami klasy,
* pole tekstowe meths wyświetlające metody klasy.

Klasa ClassEntity oraz dziedzicząca po niej OtherClassEntity mają zaimplementowane funkcje inicjalizujące klasę, czyli przypisujące kontrolki do kolumny i umożliwiające poruszanie obiektem, definiujące zachowanie obiektu w przypadku pojedynczego oraz podwójnego kliknięcia, a także wyświetlające informacje o tworzonym na diagramie elemencie. Za funkcjonalność poruszania obiektami odpowiada klasa DraggableMaker.



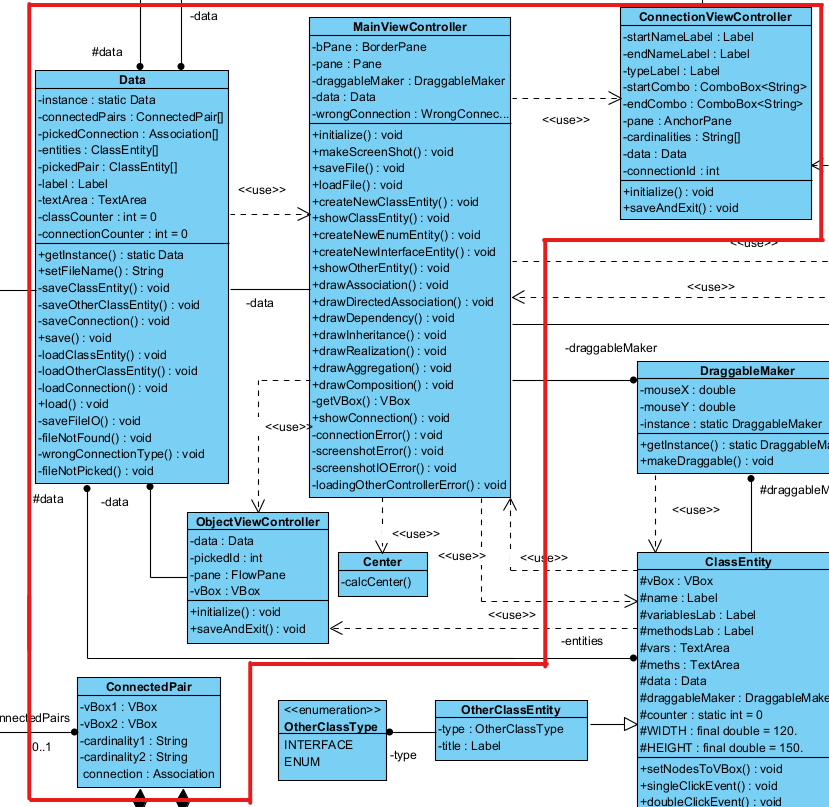
Rysunek 9. Wycinek diagramu klas prezentujący klasy odpowiadające za tworzenie oraz obsługę relacji na diagramie  
Źródło: opracowanie własne

Klasa Association oraz dziedziczące po niej klasy (rysunek 9) przedstawiają każdą z dostępnych w języku UML relacji. Pole mainLine przedstawia linę łączącą dwie klasy, a pola x1, y1, x2, y2 określają jej punkty końcowe. Są one typu SimpleDoubleProperty, dzięki czemu mają możliwość związania (ang. *bind*) z inną tego typu zmienną, gdzie przy zmianie wartości jednej ze związanych ze sobą zmiennych spowoduje jednocześnie zmianę wartości drugiej. ARROW\_SCALER określa odsunięcie od środka łączonej klasy, a counter – analogicznie jak w klasie ClassEntity – kontroluje nadawanie identyfikatorów. Dziedziczące klasy mają dodatkowe pola określające parametry grotów relacji oraz metody do ich obsługi.



Rysunek 10. Wycinek diagramu klas prezentujący klasy odpowiadające za sprawdzanie oraz reagowanie na wystąpienie niepoprawnych powiązań  
Źródło: opracowanie własne

Rolą Klasy WrongConnection oraz wykorzystujących ją klas z przyrostkiem Exception przedstawionych na rysunku 10 jest reagowanie na powiązywanie relacjami elementów na diagramie. Klasy wyjątków posiadają metodę wyświetlającą informację o błędzie, a funkcje klasy WrongConnection kontrolują poprawność oraz ewentualne rzucanie wyjątków.



Rysunek 11. Wycinek diagramu klas prezentujący klasy odpowiadające za przechowywanie danych oraz kontrolery z funkcjonalnościami interfejsu użytkownika  
Źródło: opracowanie własne

Na rysunku 11 przedstawione zostały klasy kontrolerów spajających modele przedstawione na rysunkach 8 – 10 z warstwą widoku zapisaną w plikach XML oraz klasa Data zajmująca się zapisywaniem oraz odczytywaniem danych z pliku. Mając na uwadze fakt, że obiekt klasa Data wykorzystywana jest w niemal wszystkich innych obiektach klas, zastosowany został wzorzec projektowy Singleton, który zapewnia istnienie wyłącznie jednego obiektu klasy Data. Inne klasy wykorzystują obiekt Data do zapisywania, edytowania oraz usuwania elementów oraz relacji. Wzorzec Singleton gwarantuje odwoływanie się ciągle do tego samego obiektu co analogicznie zapewnia wykorzystywanie tych samych list obiektów klas oraz relacji, będących polami klasy Data. Klasa Center odpowiedzialna jest za wyznaczenie aktualnej pozycji środka kolumny klasy, do której wiązany jest jeden z końców relacji. ConnectionViewController zajmuje się wyświetleniem okna z właściwościami wybranej relacji oraz daje możliwość określenia krotności. Pola typu Label wyświetlają nazwy połączonych relacją klas oraz typ tej relacji. Pola wyboru ComboBox pozwalają wybrać jeden z podstawowych typów krotności, które zapisane są w tabeli cardinalities. Analogicznie do ConnectionViewController zachowuje się ObjectViewController. Klasa ta przesyła informacje o określonym obiekcie klasy ClassEntity lub OtherClassEntity do sekcji widoku i pozwala na edycję jej nazwy oraz pól i metod. Najbardziej rozbudowaną klasą jest kontroler odpowiedzialny za główne okno programu - MainViewController. Odpowiada on za komunikację z modelami odpowiedzialnymi za dodawanie obiektów klas oraz relacji, zapis i odczyt pliku, a także za wykonanie zrzutu ekranu.

## Diagramy sekwencji UML

Do zaprezentowania potoku przesyłanych informacji, wykonano diagramy sekwencji przedstawiające przykładowe funkcjonalności programu: wczytywanie pliku (rysunek 12), tworzenie klas i relacji oraz wyświetlanie informacji o niej (rysunek 13), utworzenie błędnej relacji (rysunek 14) oraz zapis do pliku (rysunek 15).

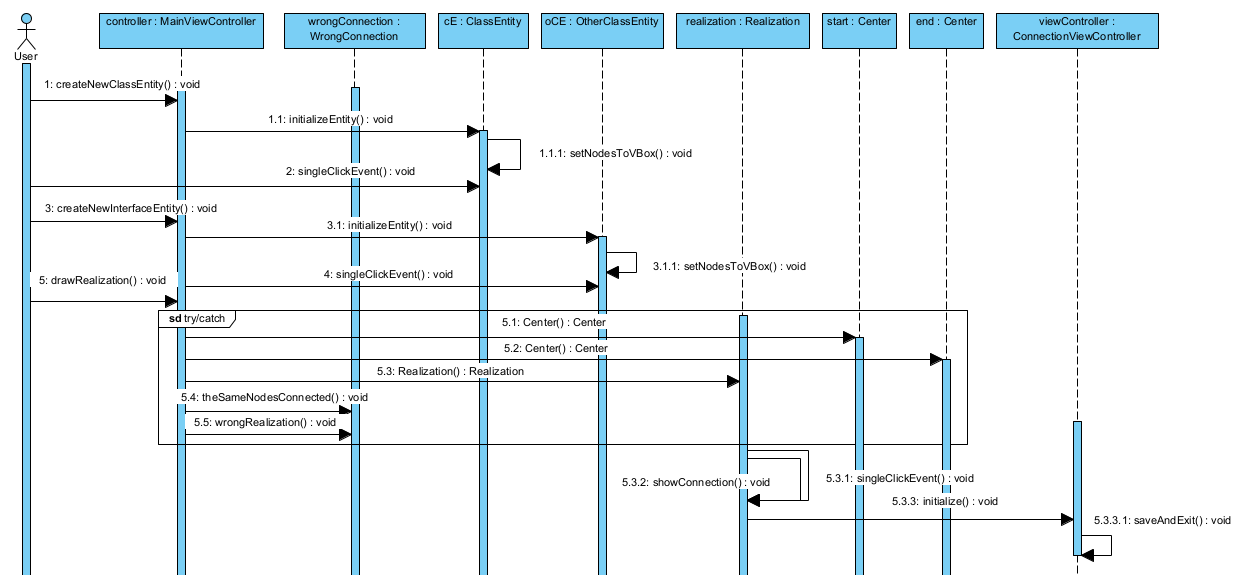
Diagram ukazany na rysunku 12 przedstawia przepływ informacji wczytywania danych z pliku. Użytkownik klikając w odpowiedni przycisk na ekranie wywołuje funkcję loadFile z klasy MainViewController, która wedle wzorca MVC, komunikuje się z klasą Data wywołując jej metodę load. W pętlach wykonywane są metody pomocnicze wczytujące najpierw klasy, a następnie relacje.

Obraz zawierający stół

Opis wygenerowany automatycznie

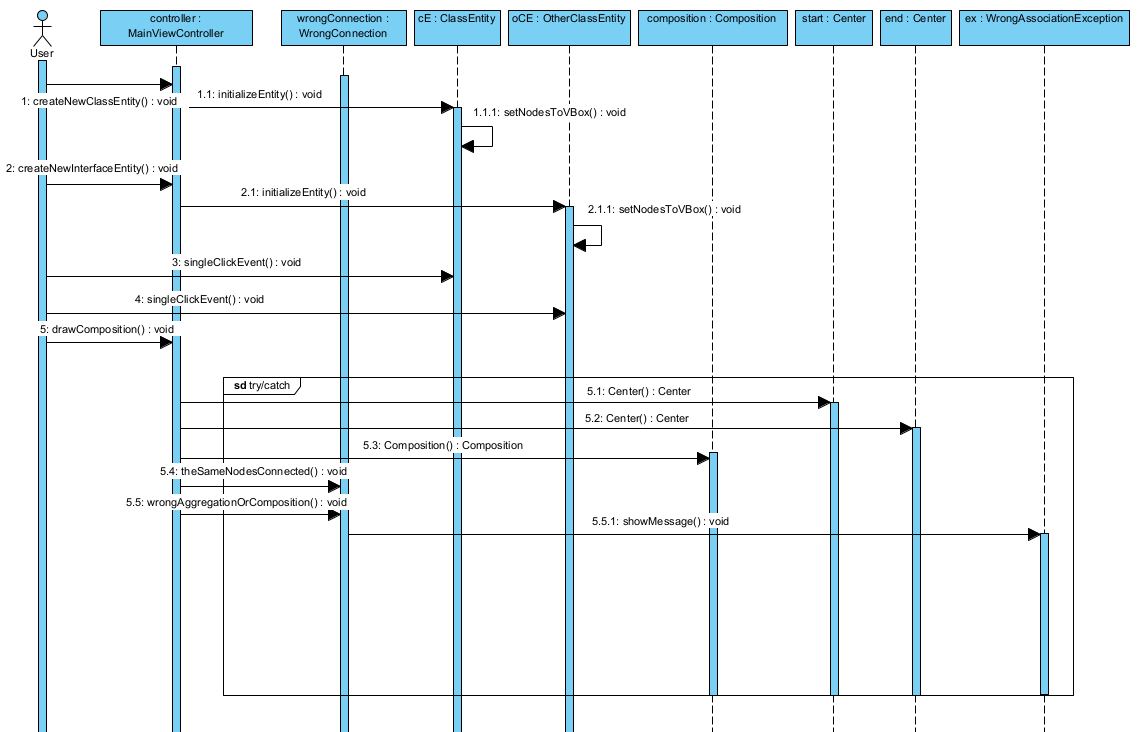
Rysunek 12. Diagram sekwencji wczytywania danych bez uwzględnienia wyjątków  
Źródło: opracowanie własne

Na rysunku 13 zamodelowano przepływ informacji tworzenia dwóch klas i powiązania ich łączenia ich relacją zgodną z zasadami aplikacji, a następnie wyświetlenia informacji o relacji. Użytkownik tworzy dwie standardowe klasy odpowiednim przyciskiem po czym zaznacza je. Wywoływane metody nowoutworzonych obiektów klas nadają im odpowiednie identyfikatory oraz przypisują kontrolki do kolumn. Następnie uruchamiana jest metodę tworzenia połączenia między wybranymi klasami. Sprawdzane są warunki utworzenia relacji – metody kontrolujące ten przebieg nie rzucają wyjątków – relacja zostaje utworzona. Ostatecznie, klikając na relację, otwierane jest okno z danymi relacji, wprowadzone ewentualne zmiany zostają zapisane poprzez naciśnięcie przycisku a samo okno zostaje automatycznie zamknięte.

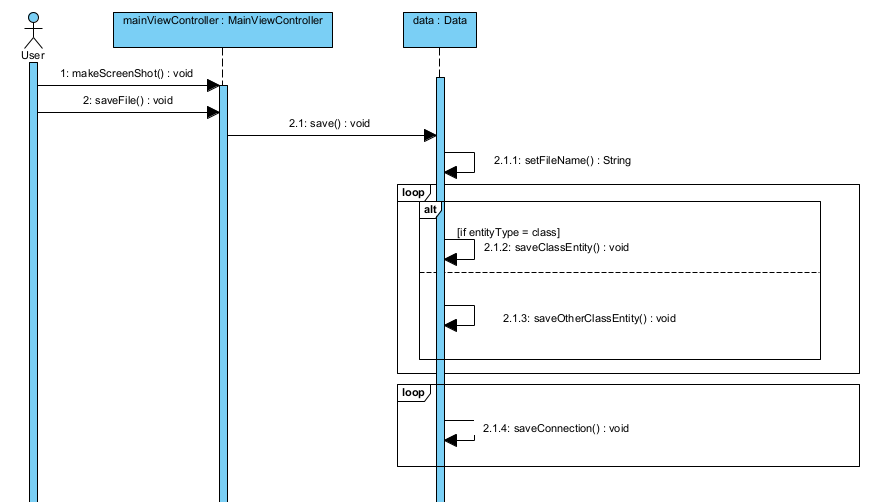


Rysunek 13. Diagram sekwencji tworzenia dwóch obiektów klas, łączenia relacją oraz wyświetlania informacji o niej  
Źródło: opracowanie własne

Diagram sekwencji przedstawiony na rysunku 14 prezentuje zachowanie programu w przypadku utworzenia błędnej relacji kompozycji. Użytkownik, jak w przykładzie z rysunku 13, tworzy dwie klasy – podstawową oraz wyliczeniową, a następnie zaznacza je (metoda singleClickEvent). Klikając odpowiedni przycisk, wywoływana jest metoda utworzenia relacji kompozycji. W bloku try – catch, którego nie można jednoznacznie zamodelować na diagramie sekwencji, tworzony jest obiekt relacji oraz sprawdzane są metody kontrolujące jej poprawność. Ponieważ nie jest możliwe połączenie standardowej klasy z klasą wyliczeniową relacją kompozycji (relacja całość – część, gdzie część nie może istnieć samodzielnie bez całości), rzucany jest wyjątek WrongAggregationOrComposition z metodą showMessage wyświetlającą na ekranie informację o nieutworzeniu relacji.



Rysunek 14. Diagram sekwencji tworzenia błędnej relacji kompozycji  
Źródło: opracowanie własne



Rysunek 15. Diagram sekwencji wykonania zrzutu ekranu oraz zapisu danych do pliku  
Źródło: opracowanie własne

Rysunek 15 przedstawia diagram sekwencji zapisu diagramu do pliku .ini. Użytkownik wybierając odpowiednie przyciski z menu na ekranie wykonuje najpierw zrzut ekranu, który zapisywany jest w pliku .jpg o podanej nazwie, a następnie zapisuje dane z diagramu. Po określeniu nazwy pliku do zapisu diagramu, w oddzielnych pętlach wywoływane są funkcje sczytujące informacje z odpowiednich list będących polami klasy Data i wpisywane są oddzielnie do kolejnych sekcji pliku konfiguracyjnego .ini.

# Implementacja rozwiązania

## Rysowanie relacji dziedziczenia

Fragment kodu 1 przedstawia implementację metody klasy MainViewController rysującej na ekranie relację dziedziczenia. Metoda ta zawarta jest w bloku try – catch, który tu pominięto.

|  |
| --- |
| Center startCenter = new Center(data.getPickedPair().get(0).getVBox());  Center endCenter = new Center(data.getPickedPair().get(1).getVBox());  wrongConnection.theSameNodesConnected(data.getPickedPair().get(0), data.getPickedPair().get(1));  wrongConnection.inheritanceBetweenOtherTypes(data.getPickedPair().get(0), data.getPickedPair().get(1));  Inheritance inheritance = new Inheritance(  startCenter.getCenterX(),  startCenter.getCenterY(),  endCenter.getCenterX(),  endCenter.getCenterY(), this);  pane.getChildren().add(inheritance);  data.getConnectedPairs().add(new ConnectedPair(  data.getPickedPair().get(0).getVBox(),  data.getPickedPair().get(1).getVBox(),  inheritance)); |

Fragment kodu 1. Budowa metody drawInheritance  
Źródło: opracowanie własne

Wykonywane jest mapowanie zaznaczonych na ekranie elementów z instancjami klasy Center, która wyznacza środek klasy na ekranie. Wywołując odpowiednie metody klasy WrongConnection, sprawdzana jest poprawność relacji. W przypadku błędu, rzucany jest wyjątek, który przerywa operację. Tworzony jest obiekt reprezentujący związek dziedziczenia, który następnie dodawany jest do rozkładu (ang. *pane*) i wyświetlony na ekranie. Tworzony, a także dodawany do odpowiedniej listy jest również obiekt klasy ConnectedPair, którego zadaniem jest zapamiętanie informacji o relacji oraz klasach nią połączonych, co ułatwia późniejszy, ewentualny ich zapis do pliku.

## Zapis obiektu klasy wyliczeniowej lub interfejsu do pliku .ini

We fragmencie kodu 2 przedstawiono ciało metody zapisywania pojedynczej instancji klasy OtherClassEntity (klasa wyliczeniowa lub interfejs) do pliku z klasy Data. Metoda ta jest wykorzystywana w nadrzędnej funkcji zapisu, w której następuje połączenie z plikiem oraz finalny zapis informacji. Do komunikacji z plikami .ini wykorzystano zewnętrzną bibliotekę ini4j.

|  |
| --- |
| TextArea temp = new TextArea();  ini.put("class" + classCounter, "id", vBox.getId());  label = (Label) vBox.getChildren().get(0);  ini.put("class" + classCounter, "type", label.getText());  label = (Label) vBox.getChildren().get(1);  ini.put("class" + classCounter, "name", label.getText());  ini.put("class" + classCounter, "layoutX", vBox.getLayoutX()); ini.put("class" + classCounter, "layoutY", vBox.getLayoutY());  textArea = (TextArea) vBox.getChildren().get(3);  temp.setText(textArea.getText());  temp.setText(temp.getText().replace(‘\n’,’`’)); ini.put("class" + classCounter, "vars", textArea.getText());  textArea = (TextArea) vBox.getChildren().get(5);  temp.setText(textArea.getText());  temp.setText(temp.getText().replace(‘\n’,’`’)); ini.put("class" + classCounter, "meths", textArea.getText());  classCounter++; |

Fragment kodu 2. Metoda zapisu obiektu klasy OtherClassEntity do pliku .ini  
Źródło: opracowanie własne

Metoda ta wykonywana jest w pętli w nadrzędnej funkcji save klasy Data. Pobiera ona jako parametr kolejne obiekty OtherClassEntity, które zapisane są w pamięci programu na liście entities. W pliku inicjalizującym, do którego zapisywane są informacje, tworzona jest dla każdego elementu sekcja z identyfikatorem bazującym na wartości zmiennej classCounter. Do sekcji po kolei wpisywane są informacje o danym elemencie: identyfikator, typ, nazwa, aktualne położenie na ekranie oraz pola i metody. Z powodu problemów biblioteki ini4j z zapisem wartości pola tekstowego, w którym znajdowały się znaki nie będące literami w przypadku, gdy zapisywana jest więcej niż jedna linijka tekstu, posłużono się zmienną pomocniczą temp. Jej zadaniem było pobranie wartości z pola tekstowego, podmienienie znaku końca linii na rzadko wykorzystywany znak grawis i zapisanie do pliku tej właśnie zmienionej kopii*.* W metodzie odczytywania z pliku zachodzi proces odwrotny – grawis zamieniany jest na znak końca linii. Analogicznie wykonywany jest zapis oraz odczyt podstawowej klasy – obiektu ClassEntity.

## Wyznaczanie oraz aktualizacja pozycji środka kolumny

Wbudowana w JavaFX klasa Bounds odpowiada za rozmiar i położenie obiektu na ekranie. W programie, klasa Center wykorzystuje obiekty Bounds oraz mechanizm ChangeListener do nasłuchiwania zmian oraz reagowania na nie. Center łączony jest z kolumną VBox klasy ClassEntity lub OtherClassEntity. W trakcie poruszania obiektem na ekranie, Center oblicza nowe wartości położenia obiektu względem okna aplikacji (fragment kodu 3).

|  |
| --- |
| public Center(VBox vb) {  calcCenter(vb.getBoundsInParent());  vb.boundsInParentProperty().addListener(  (observableValue, oldBounds, bounds) -> calcCenter(bounds));  }  private void calcCenter(Bounds bounds) {  centerX.set(bounds.getMinX() + bounds.getWidth() / 2);  centerY.set(bounds.getMinY() + bounds.getHeight() / 2);  } |

Fragment kodu 3. Nasłuchiwanie oraz obliczanie nowych wartości środka kolumny  
Źródło: opracowanie własne na podstawie [5]

Funkcja calcCenter pobiera informacje o położeniu górnego lewego rogu kolumny i na jego podstawie oraz wysokości i szerokości kolumny, wyznacza środek obiektu. Nasłuchiwacz (ang. *listener*) dodawany do każdego obiektu, kontroluje wystąpienie zmiany położenia na ekranie oraz oblicza nowe wartości środka obiektu.

## Wykonywanie zrzutu ekranu

Mimo, że Java Swing nie jest już wspierany i rozwijany, to ciągle wykorzystywany jest do tworzenia GUI. Dzieje się tak, między innymi dlatego, że JavaFX wykorzystuje biblioteki swojego poprzednika.

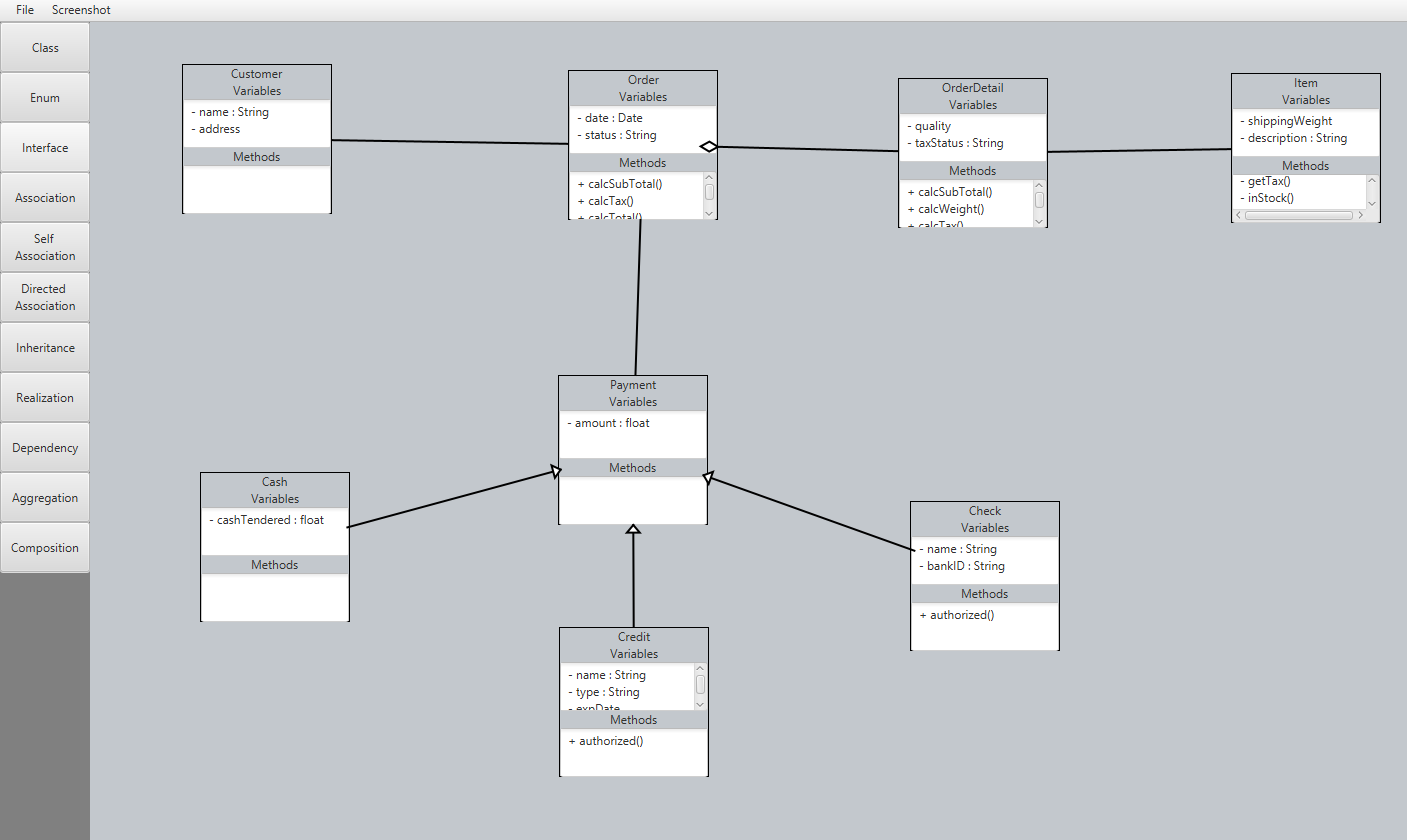
|  |
| --- |
| FileChooser fileChooser = new FileChooser(); FileChooser.ExtensionFilter filter = new FileChooser.ExtensionFilter( "PNG (\*.png)","\*.png"); fileChooser.getExtensionFilters().add(filter);  Stage stage = (Stage) pane.getScene().getWindow();  File file = fileChooser.showSaveDialog(stage); if(file!=null) {  try {  WritableImage image = stage.getScene().snapshot(null);  ImageIO.write(SwingFXUtils.fromFXImage(image, null), "png", file);  } catch (NoSuchElementException exception) {  screenshotError();  } catch (IOException exception) {  screenshotIOError();  } } |

Fragment kodu 4. Wykonanie zrzutu okna aplikacji  
Źródło: opracowanie własne

Do opracowania metody wykonującej zrzut ekranu wykorzystano komponent biblioteki Swing – SwingFXUtils.fromFXImage. Otwierane jest okno z plikami na dysku, które przefiltrowano pod kątem występowania tych z rozszerzeniem .png. Po pobraniu okna aplikacji oraz podaniu nazwy pliku pobierany jest aktualny obraz okna aplikacji oraz, wykorzystując metodę z biblioteki Swing, zapisywany jest on w wybranym miejscu na dysku.

# Testowanie i badania

Cele jakie zostały postawione aplikacji to możliwość projektowania diagramów klas UML oraz kontrola poprawności modelowanych powiązań między elementami diagramu. Program spełnia oba postawione mu zadania. Na rysunku 16 przedstawiono zrzut ekranu z wykonanej aplikacji, w której zaprojektowano przykładowy system zamówień i sprzedaży w formie diagramu klas UML.



Rysunek 16. Diagram klas UML wykonany w aplikacji  
Źródło: opracowanie własne na podstawie [6]

Na panelu bocznym, po lewej stronie, znajdują się przyciski odpowiedzialne za umieszczanie na ekranie klas oraz relacji, a przy pomocy pól menu na pasku u góry ekranu, możliwe jest wykonanie zapisu oraz odczytu danych, a także wykonanie zrzutu ekranu.

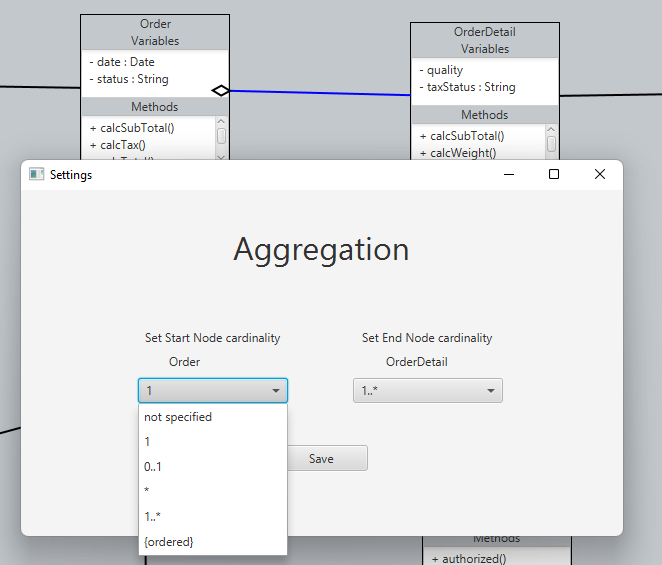
Otwarcie okna z danymi klasy dokonuje się poprzez podwójne kliknięcie na wybrany element diagramu. Na rysunku 17 przedstawiono wygląd okna z informacjami o klasie OrderDetail, w którym określona została nazwa klasy, jej pola oraz metody. Pola tekstowe do wpisywania zmiennych oraz funkcji nie mają nałożonych żadnych kontroli poprawności. Od użytkownika wymaga się nadzoru nad prawidłowym zapisem tych informacji. Zapis zmian dokonuje się poprzez naciśnięcie przycisku „Save”.

Obraz zawierający tekst

Opis wygenerowany automatycznie

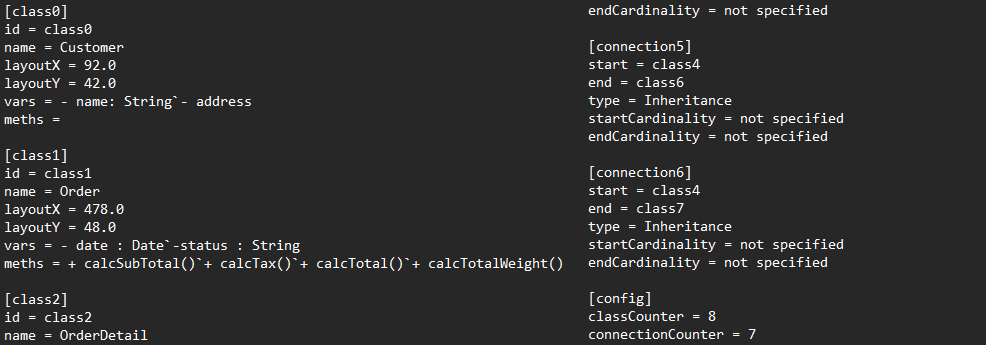
Rysunek 17. Okno z informacjami o klasie OrderDetail  
Źródło: opracowanie własne

Klikając podwójnie na graficzną reprezentację relacji, otwarte zostaje nowe okno z informacjami o danym związku, co przedstawia rysunek 18. Wykorzystując pola wyboru, możliwe jest określenie krotności obiektów klas jakie można powiązać z daną cechą, których nazwy zapisane są powyżej odpowiednich pól. Analogicznie do okna z informacjami o klasie, zapis zmian dokonywany jest poprzez naciśnięcie przycisku „Save”.



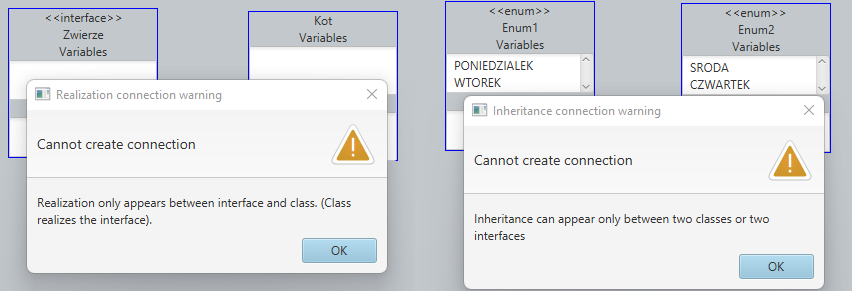
Rysunek 18. Okno z informacjami o związku między klasą Order a OrderDetail  
Źródło: opracowanie własne

Diagram klas zaprezentowany na rysunku 16 zapisano do pliku, którego część przedstawiono na rysunku 19. Wyróżnić można z niego pojedyncze sekcje odpowiadające za indywidualne elementy diagramu. Zauważyć można również działanie metod zapisujących – zawartość pól tekstowych zapisana jest w jednej linii, a każde pole czy metodę oddziela grawis.



Rysunek 19. Zawartość pliku z zapisanymi danymi diagramu klas przedstawionego na rysunku 16

Najważniejszą funkcjonalnością, będącą jednocześnie czynnikiem wpływającym na niepowtarzalność programu na tle innych, opisywanych w rozdziale drugim, jest kontrola poprawności modelowanych relacji. Logika ograniczeń zawiera się w metodach klasy WrongConnection, która w przypadku wykrycia błędu, rzuca odpowiedni wyjątek. Rezultatem jest wyświetlenie na ekranie informacji o wykrytej nieprawidłowości oraz usunięcie tworzonego połączenia (rysunek 20).



Rysunek . Informacje o próbie zamodelowania niepoprawnego połączenia między elementami

Ponieważ niepoprawna jest relacja realizowania pomiędzy klasą a interfejsem, w której grot relacji zwrócony jest w stronę klasy, na ekranie pojawia się komunikat z informacją o popełnieniu błędu. Podobnie jest z połączeniem dwóch klas wyliczeniowych związkiem dziedziczenia – program wykrywa niezgodność z regułami i rzuca wyjątek z informacją o możliwych poprawnych połączeniach z wykorzystaniem danej relacji.

# Podsumowanie oraz dalszy rozwój

Wszystkie cele założone w rozdziale trzecim zostały osiągnięte. Przy pomocy programu, użytkownik może zamodelować system w postaci diagramu klas UML, a tworzone niepoprawne połączenia między elementami diagramu zostaną wykryte, zgłoszone użytkownikowi oraz usunięte.

# Bibliografia

1. On the Usage of UML: Initial Results of Analyzing Open UML Models, P. Langer, T. Mayerhofer, M. Wimmer, G. Kappel, Gesellschaft für Informatik eV 2014, Bonn
2. On the Use of UML Documentation in Software Maintenance: Results from a Survey in Industry, A. M. Fernandez-Saez, D. Caivano, M. Genero, M. R. V. Chaudron, IEEE 2015, Ottawa
3. Insights in Students’ Problems during UML Modeling, R. Reuter, T. Stark, Y. Sedelmaier, D. Landes, J. Mottok, C. Wolff, IEEE 2020, Porto
4. Identifying Difficulties in Learning UML, K. Siau, P.-P. Loo, Information Systems Management 2006, Abingdon
5. John Smith ”Jewelsea”, https://stackoverflow.com/questions/19748744/javafx-how-to-connect-two-nodes-by-a-line (dostęp: 13.12.2022)
6. Sonoo Jaiswal, https://www.javatpoint.com/uml-class-diagram (dostęp: 13.12.2022)

# Bibliografia

1. P Langer, T Mayerhofer, M Wimmer, G Kappel - Modellierung 2014, On the Usage of UML: Initial Results of Analyzing Open UML Models (<https://cs.emis.de/LNI/Proceedings/Proceedings225/289.pdf>)
2. https://sci-hub.se/10.1109/MODELS.2015.7338260
3. <https://sci-hub.se/10.1109/EDUCON45650.2020.9125110>
4. <https://sci-hub.se/10.1201/1078.10580530/46108.23.3.20060601/93706.5>
5. <https://stackoverflow.com/questions/19748744/javafx-how-to-connect-two-nodes-by-a-line> (30.11.2022)
6. <https://www.javatpoint.com/uml-class-diagram> (12.12.2022)