

**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](#_Toc121867686)

[2. Przegląd istniejących rozwiązań w zakresie konstrukcji diagramów klas UML 6](#_Toc121867687)

[2.1. Visual Paradigm 6](#_Toc121867688)

[2.2. StarUML 7](#_Toc121867689)

[2.3. draw.io 9](#_Toc121867690)

[3. Cel pracy 10](#_Toc121867691)

[4. Projekt aplikacji 12](#_Toc121867692)

[4.1. Diagram klas UML 12](#_Toc121867693)

[4.2. Diagramy sekwencji UML 16](#_Toc121867694)

[5. Implementacja rozwiązania 21](#_Toc121867695)

[5.1. Rysowanie relacji dziedziczenia 21](#_Toc121867696)

[5.2. Zapis obiektu klasy wyliczeniowej lub interfejsu do pliku .ini 21](#_Toc121867697)

[5.3. Wyznaczanie oraz aktualizacja pozycji środka kolumny 23](#_Toc121867698)

[5.4. Wykonywanie zrzutu ekranu 23](#_Toc121867699)

[6. Testowanie i badania 25](#_Toc121867700)

[7. Podsumowanie 28](#_Toc121867701)

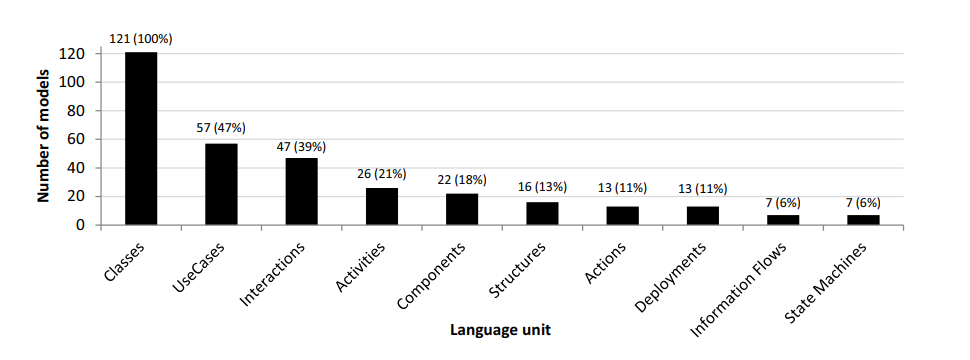
[8. Bibliografia 29](#_Toc121867702)

# Wstęp

Jednym z najistotniejszych czynników wpływających na powodzenie projektu informatycznego jest sprawna komunikacja między zespołami, nie tylko programistów, ale również na linii deweloperzy – przedsiębiorcy. 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 czy działów, konieczne jest wprowadzenie nowego sposobu wymiany informacji. Na bazie wniosków z kilkunastu lat doświadczeń zebranych przez osoby 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*) [1].

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

Specyfikacja języka UML określa wiele typów diagramów. Do najpopularniejszych należy diagram przypadków użycia (ang. *use case*), którego zadaniem jest ukazanie funkcjonalności projektowanego systemu, oprogramowania. Istotny jest także diagram sekwencji, który służy do ukazania przepływu interakcji pomiędzy obiektami przy jednoczesnym uwzględnieniu przesyłanych komunikatów. Najważniejszym i zarazem najczęściej wykorzystywanym diagramem jest jednak diagram klas [3]. Pozwala on na zadeklarowanie klas wraz z ich polami i metodami, a co najważniejsze, przedstawić relacje pomiędzy nimi, takie jak dziedziczenie klas czy wykorzystywanie obiektu klasy jako argument metody w innej klasie. Te trzy diagramy: przypadków użycia, klas oraz sekwencji umożliwiają razem pełne zamodelowanie systemu.



Rysunek 1. Częstość wykorzystywania poszczególnych typów diagramów UML.  
Źródło: [2]

Podstawowym elementem diagramu 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 diagramu. W specyfikacji języka UML wyróżnionych zostało sześć typów relacji między klasami, które można przyrównać do różnych konstrukcji w wielu językach programowania:

* zależność (ang. *dependency*) – relacja, w której 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 przez jedną klasę 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 aplikacji do edycji 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*, jak i *off-line.* Alternatywnym wyborem są edytory do tworzenia diagramów ogólnego przeznaczenia, płatne i darmowe. Wśród wielu tego typu programów, dostępne są takie, posiadające moduły umożliwiające projektowanie diagramów klas UML. Najczęściej są to aplikacje działające w przeglądarkach, posiadające ograniczone możliwości edycji, a do ich działania konieczne jest połączenie z Internetem. Rzadziej wykorzystywanym, oryginalnym wyborem mogą być projekty osób prywatnych, które udostępniają je dla użytkowników. Cechują się o wiele mniejszymi możliwościami w porównaniu do produktów komercyjnych oraz niską odpornością na błędy użytkownika, ale są najczęściej darmowe oraz dostępne w wersjach desktopowych 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.

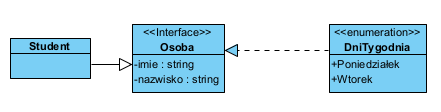
Visual Paradigm oferuje użytkownikowi wiele możliwości w projektowaniu diagramów klas. Poza podstawowym typem klasy, interfejsem czy klasą wyliczeniową (ang. *enumeration*), pozwala na utworzenie takich elementów jak: Bean, prymityw, pakiet, agent czy serwis REST. Otwierając okno ze specyfikacją danej klasy, użytkownik może określić jej typ, widoczność oraz nazwę, a także dodać pola i metody.

Istotą diagramów klas UML jest modelowanie relacji między klasami. W programie odnaleźć można każdy jej rodzaj – od zależności po dziedziczenie. Definiując połączenia między klasami, możliwe jest określenie liczności, nazw przesyłanych atrybutów, a w przypadku relacji zależności – również stereotypów.

Sposób wprowadzania danych przez użytkownika został specjalnie opracowany 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.

Visual Paradigm istotnie wyróżnia się na tle podobnych produktów funkcjonalnością pozwalającą na generowanie diagramów klas na podstawie wcześniej przygotowanego kodu źródłowego Javy lub C++, a także na operację odwrotną – wygenerowanie kodu z utworzonego wcześniej diagramu. Dla innych rodzajów 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 prywatnych i firm, chcących w pełni skorzystać z funkcjonalności Visual Paradigm, może okazać się koszt wykupienia licencji. Dodatkowo, pomimo wielu udogodnień, aplikacja nie posiada zabezpieczeń przeciwko tworzeniu niepoprawnych relacji, dlatego w programie, 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 takiej formy nadzoru, która ograniczyłaby liczbę błędów w projekcie. Za wadę uznać można 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 w pełni 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.

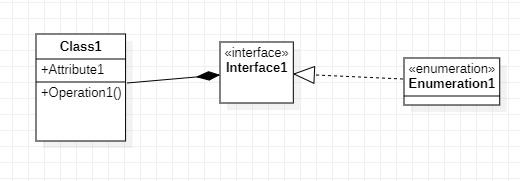
## StarUML

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 definiuje język UML.

Program umożliwia dodawanie oraz przesuwanie po ekranie klas oraz łączenie ich w relacje, a po naciśnięciu na element, wyświetlanie jego parametrów. Okno z właściwościami obiektu zawiera dużą liczbę informacji oraz możliwości edycji, które przedstawione zostały w postaci ikon, pól wyboru i pól tekstowych, których niewielkie rozmiary sprawiają, że stają się nieczytelne. Oddzielne usuwanie klasy czy relacji z okna lub drzewa projektu, w przeciwieństwie do Visual Paradigm, nie zostało w żaden sposób ułatwione. Zaniedbanie kontrolowania usuwanych elementów może doprowadzić do stanu, w którym diagram będzie nieczytelny. Co więcej, możliwe jest dodanie pól i metod do interfejsu jedynie w widoku drzewa projektu, natomiast w reprezentującym ten interfejs elemencie diagramu, zmiany nie zachodzą (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 związek realizowania – konieczne jest połączenie tą relacją interfejsu tak, aby grot relacji wskazywał właśnie na niego. W przeciwnym razie, aplikacja wyświetla komunikat o błędzie. Możliwe jest jednak niepoprawne zamodelowanie realizacji interfejsu przez klasę wyliczeniową.



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



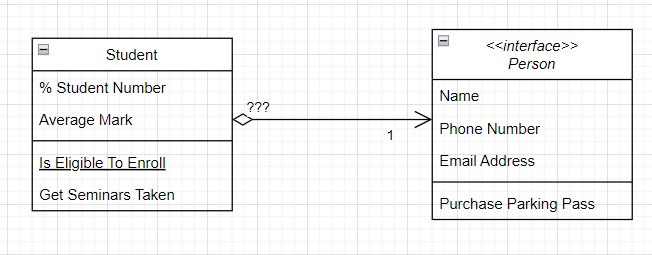
Rysunek 4. Błędne relacje w programie StarUML  
Źródło: opracowanie własne

Dzięki zaangażowaniu społeczności zrzeszonej wokół projektu StarUML, program doczekał się wielu ciekawych rozszerzeń, takich jak generowanie kodu w różnych językach programowania na podstawie diagramów. 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 jedynie *on-line,* w przeglądarce internetowej, do tworzenia wielu typów diagramów. Wśród nich, odnaleźć można moduły pozwalające na projektowanie diagramów UML.

Aplikacja posiada podstawowe możliwości modelowania systemów przy użyciu języka UML. Użytkownik ma możliwość dodawania, usuwania oraz przemieszczania po ekranie klasa oraz łączących je relacji, określać widoczność pól i metod, krotności oraz role. Z racji swojego nieokreślonego sztywnymi regułami 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 wbudowane komponenty, które często nie są poprawnie zintegrowane z innymi modułami. W przypadku, gdy potrzebne jest określenie krotności, konieczne jest niekiedy dodanie oddzielnego pola tekstowego, które w przypadku zmiany położenia strzałki reprezentującej relację, pozostanie w tym samym miejscu.

# Cel pracy

W zaprezentowanych w rozdziale drugim, dostępnych 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 [4], [5].

Celem pracy jest zaprojektowanie i implementacja programu do tworzenia oraz edycji diagramów klas UML posiadającego zabezpieczenia przed tworzeniem niepoprawnych związków między elementami. Program ten powinien pozwalać na dodawanie, usuwanie oraz edytowanie klas standardowych, klas wyliczeniowych i interfejsów, a także każdej z sześciu rodzajów relacji przedstawionych we wstępie. Modyfikowanie klas opierać się powinno na określaniu ich nazw oraz pól i metod, a w przypadku relacji – ustalaniu krotności. Aplikacja winna posiadać funkcje umożliwiające poruszanie klasami oraz odtwarzające zapisaną wcześniej pracę. Konieczny staje się wtedy zapis potrzebnych danych do pliku, którego wczytanie pozwalałoby na kontynuowanie zadania. Przydatną funkcjonalnością jest również wykonanie zrzutu okna programu oraz jego zapis do pliku graficznego. Do ważnych cech projektowanej aplikacji powinny należeć szybkość działania oraz prosta, intuicyjna obsługa.

Zasadniczym celem pracy jest wprowadzenie do programu zabezpieczeń przed popełnianiem błędów niepoprawnego łączenia klas relacjami. Uniemożliwione powinno być tworzenie błędnych powiązań, które można określić jako bezkompromisowe, jednoznaczne, czyli:

* 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 klas wyliczeniowych,
* powiązania realizacji w sposób inny niż między klasą a interfejsem tak, że grot relacji wskazuje na interfejs.

Mimo wprowadzenia tych ograniczeń, nie jest możliwe wyeliminowanie wszystkich błędów. Niewykonalne jest ustalenie poprawności niektórych relacji odgórnie, bez zagłębienia się w specyficzny, indywidualny projekt oraz jego logikę. Przykładowym problemem może być kontrola zasadności występowania związku agregacji czy kompozycji. Posiadając informacje jedynie z diagramu klas, niemożliwe jest zdefiniowanie, która relacja jest poprawna. W takich sytuacjach konieczne jest powołanie się na doświadczenie projektanta systemu i dokumentację standardu UML.

Do komunikacji z użytkownikiem, w przypadku prób niepoprawnego łączenia klas w związki, program winien wyświetlić na ekranie komunikat z informacją o popełnieniu błędu.

# 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: dostępie do obiektów przez referencje czy automatycznego usuwania niewykorzystywanych już obiektów przez *Garbage Collector* [6].

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.

Obraz zawierający tekst

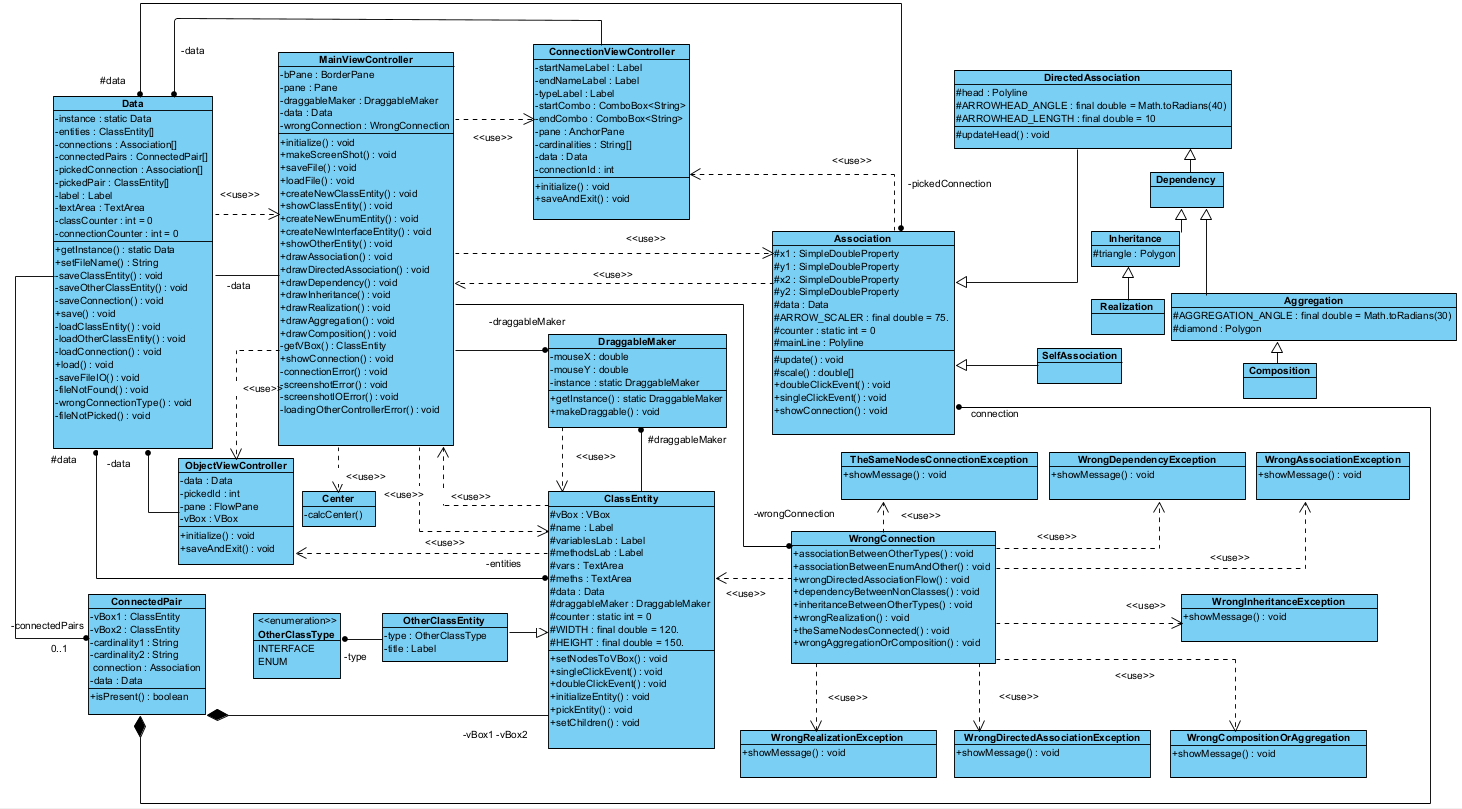
Opis wygenerowany automatycznie

Rysunek 6. Struktura pojedynczej sekcji pliku konfiguracyjnego .ini  
Źródło: [7]

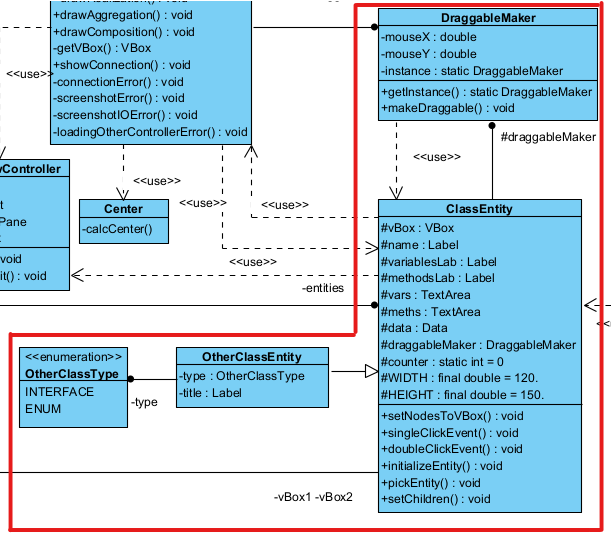
Rozszerzeniem plików z danymi, generowanych przez aplikację jest .ini, który jest typem pliku konfiguracyjnego systemu Windows. Istotną zaletą plików .ini jest silne ustrukturyzowany sposób zapisywania informacji. Zawartość pliku w takim formacie posegregowana jest w sekcje, w których parametry mapowane są na relację klucz – wartość (rysunek 6). Do komunikacji z plikami konfiguracyjnymi Windows, wykorzystano zewnętrzną bibliotekę ini4j.

## 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

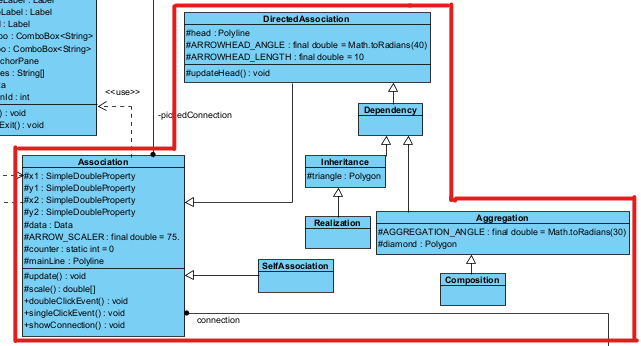


Rysunek 8. Wycinek diagramu klas prezentujący klasy odpowiadające  
za tworzenie oraz obsługę elementów na diagramie  
Źródło: opracowanie własne

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:

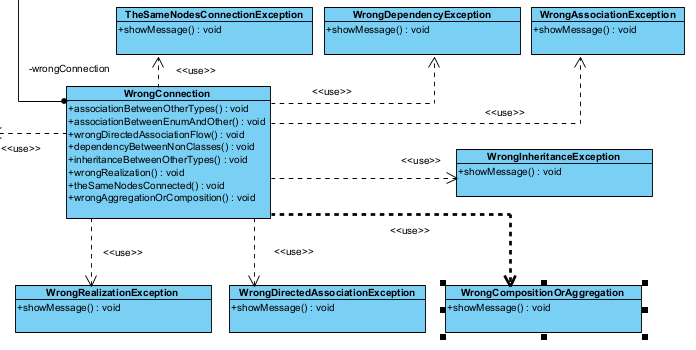
* etykieta (ang. *label*) type podająca typ klasy,
* etykieta name podająca nazwę klasy,
* etykieta variablesLab sygnalizująca sekcję z polami klasy,
* pole tekstowe (ang. text area) vars wyświetlające pola klasy,
* etykieta methodsLab sygnalizująca sekcję z metodami klasy,
* pole tekstowe meths wyświetlające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



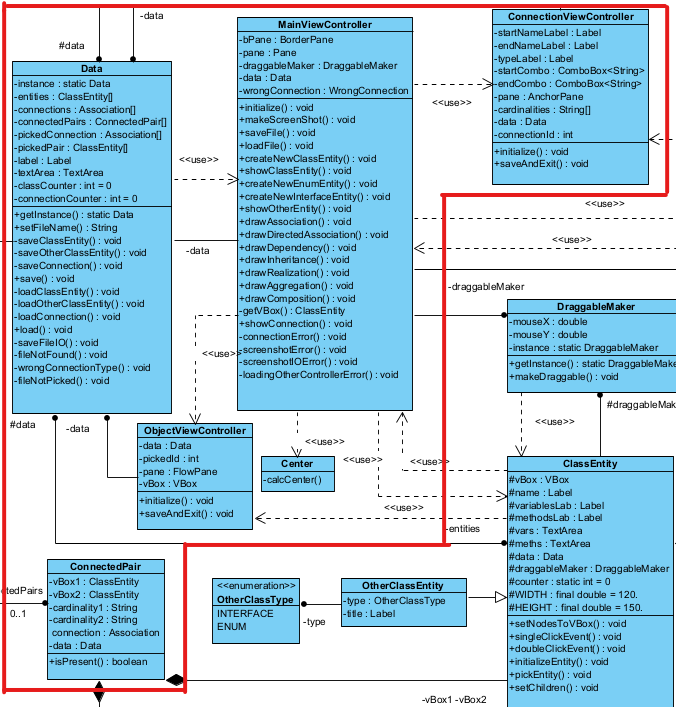
Rysunek 9. Wycinek diagramu klas prezentujący klasy odpowiadające  
za tworzenie oraz obsługę relacji na diagramie  
Źródło: opracowanie własne

Klasa Association oraz dziedziczące po niej klasy (rysunek 9) odzwierciedlają każdą z dostępnych w języku UML relacji. Pole mainLine przedstawia linę łączącą dwie klasy, a pola x1, y1, x2, y2 określają jej punkty końcowe. Będąc typu SimpleDoubleProperty, 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 zaprezentowane 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).

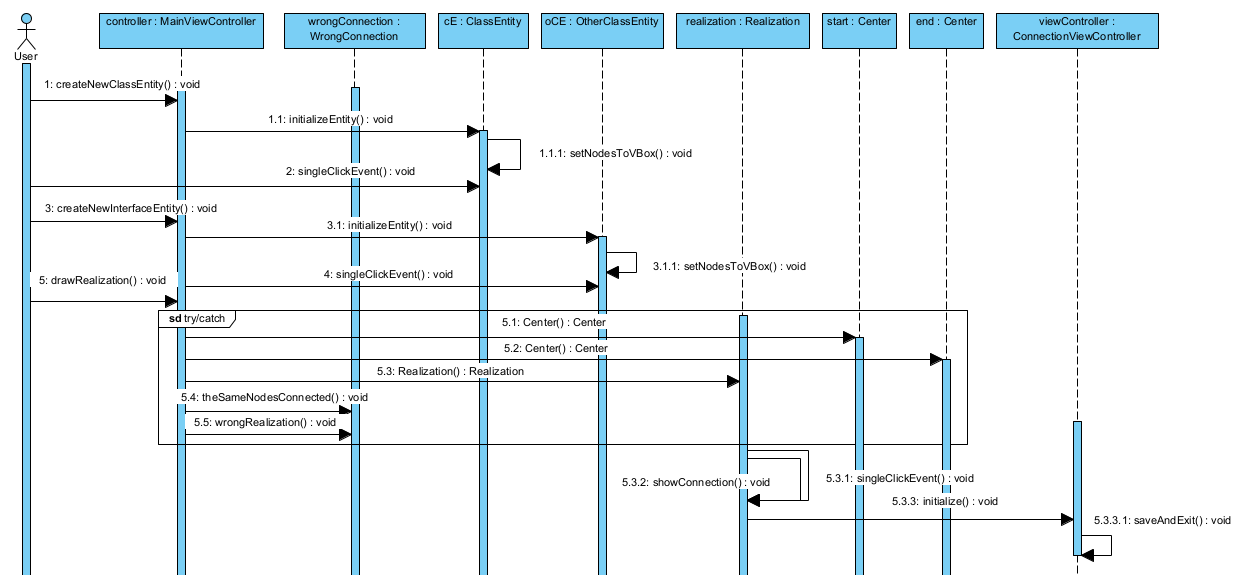
Diagram ukazany na rysunku 12 przedstawia przepływ informacji wczytywania danych z pliku. Użytkownik klikając w odpowiedni przycisk na ekranie wywołuje funkcję loadFile z klasy MainViewController, która wedle wzorca MVC, komunikuje się z klasą Data wywołując jej metodę load. W pętlach wykonywane są metody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

Obraz zawierający stół

Opis wygenerowany automatycznie

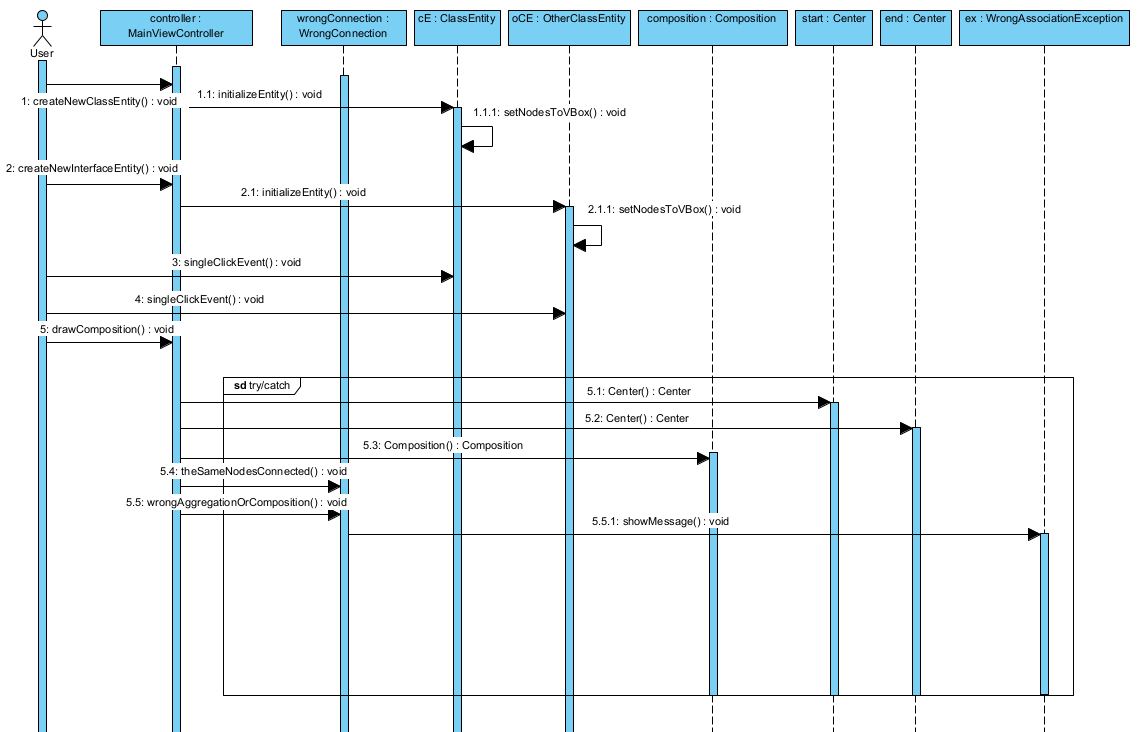
Rysunek 12. Diagram sekwencji wczytywania danych bez uwzględnienia wyjątków  
Źródło: opracowanie własne

Na rysunku 13 zamodelowano przepływ informacji tworzenia dwóch klas i powiązania ich 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.

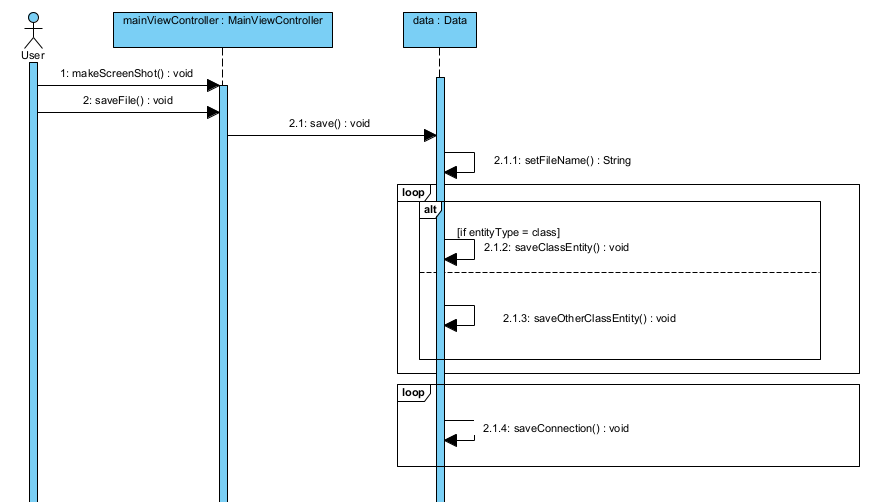


Rysunek 13. Diagram sekwencji tworzenia dwóch obiektów klas, łączenia relacją oraz wyświetlania informacji o niej  
Źródło: opracowanie własne

Diagram sekwencji przedstawiony na rysunku 14 prezentuje interakcję pomiędzy obiektami klas 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 tworzonego przez użytkownika 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 operacji 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 [8]

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.

## Zapis diagramu do pliku graficznego

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();  } } |

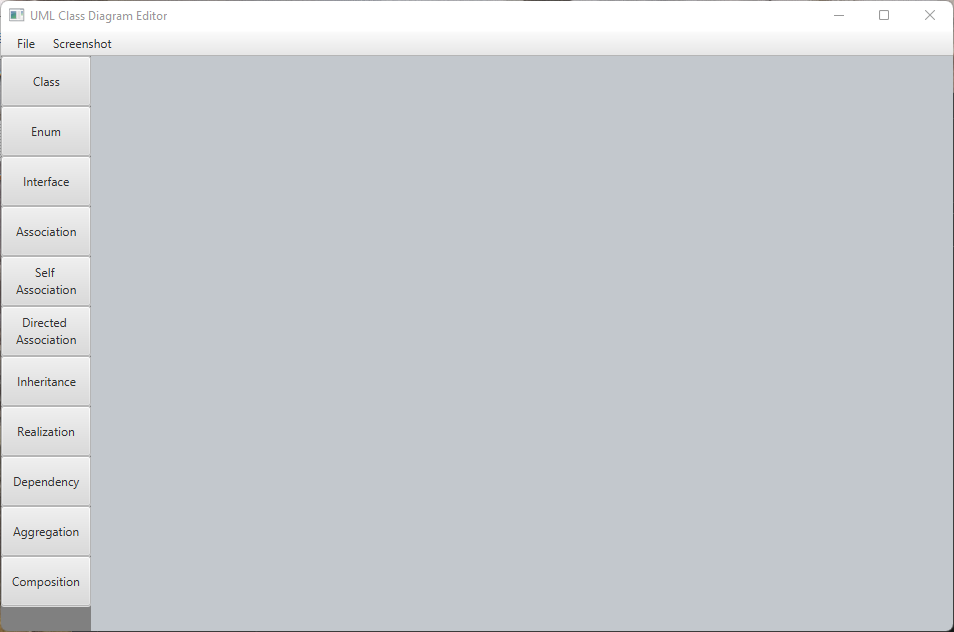
Fragment kodu 4. Wykonanie zrzutu okna aplikacji  
Źródło: opracowanie własne

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

## Uruchomienie programu

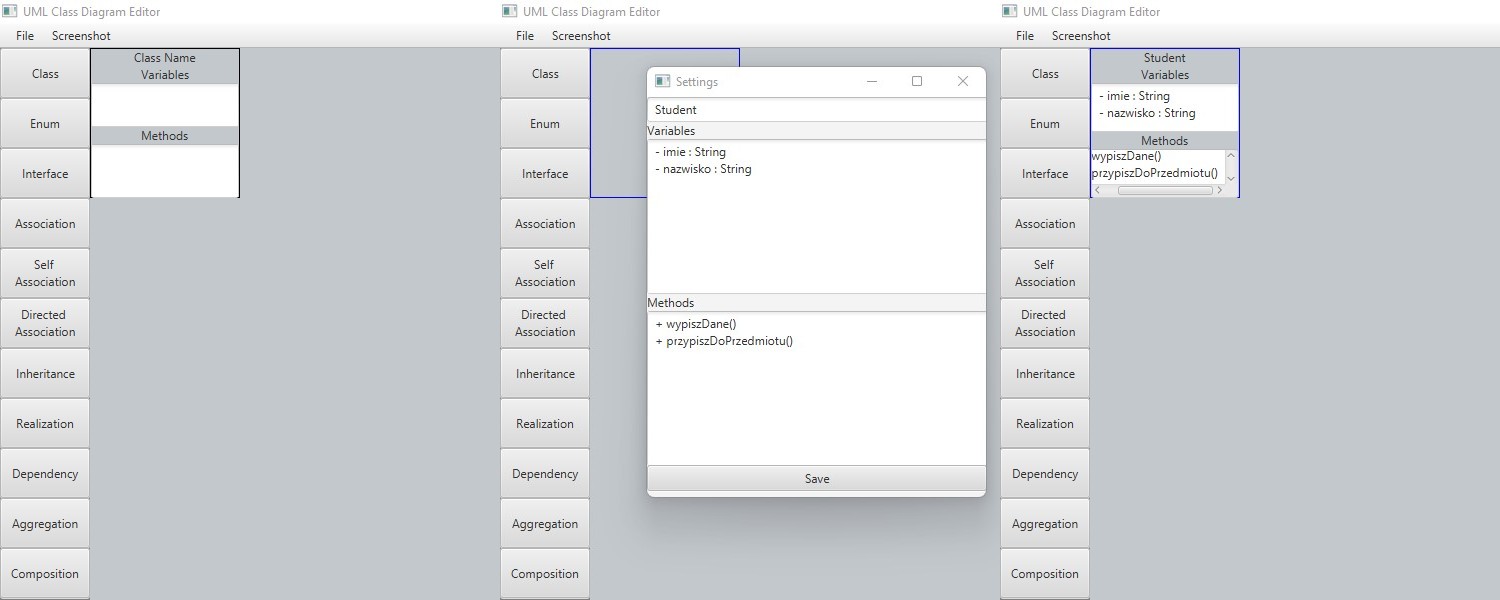
Program został napisany z wykorzystaniem narzędzi SDK w wersji 18 i skompresowany do archiwum JAR. Do jego uruchomienia, potrzebne jest zainstalowanie Javy 18 na komputerze oraz ewentualne jej skonfigurowanie w taki sposób, aby system korzystał właśnie z tej wersji oprogramowania. Do sprawdzenia aktualnie wykorzystywanej wersji konieczne jest wywołanie komendy *java -version* w konsoli Windows. Po instalacji Javy, program uruchomić można klikając dwukrotnie na jego ikonę. Na ekranie wyświetlone zostanie główne okno aplikacji (rysunek 16). Składa się ono ze znajdującego się po lewej stronie panelu z dostępnymi do utworzenia elementami oraz relacjami. Na górze okna odnaleźć można menu z opcjami pozwalającymi na zapis oraz odczyt pliku, a także zapisanie diagramu w formie pliku graficznego.



Rysunek 16. Główne okno programu  
Źródło: opracowanie własne

## Rysowanie klasy

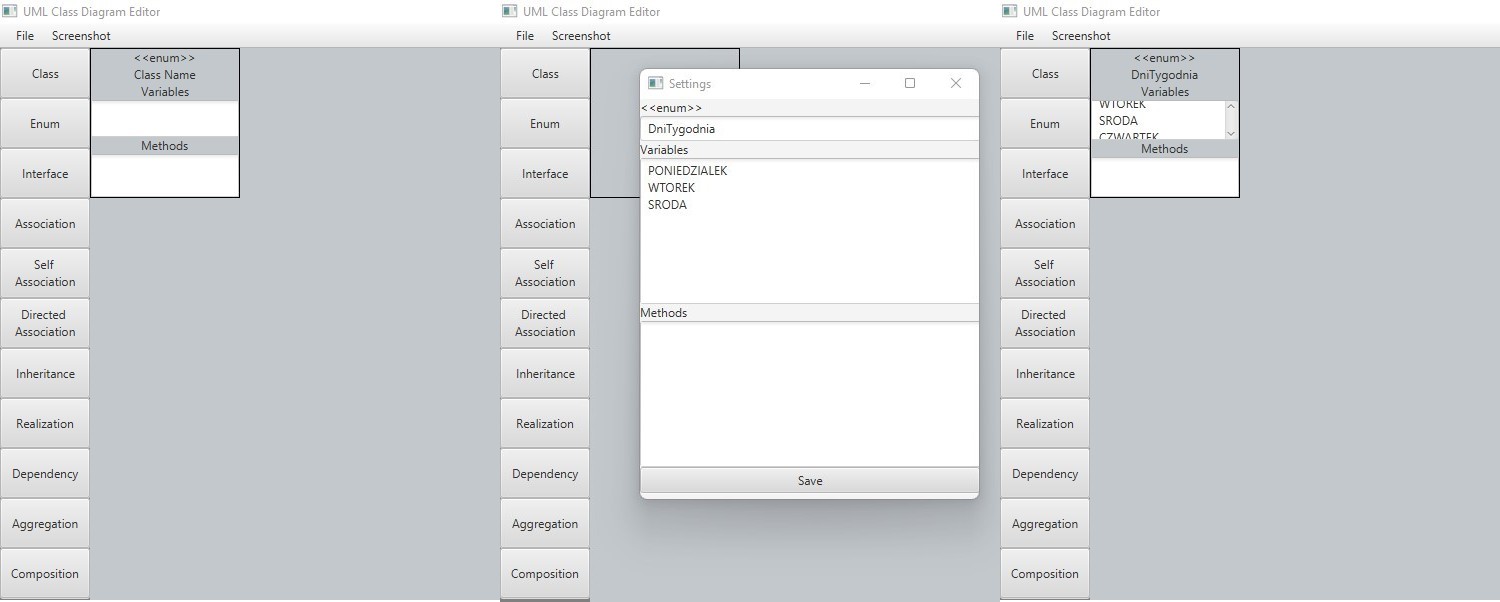
Naciśnięcie przycisku „Class” na panelu bocznym spowoduje wywołanie funkcji klasy MainViewController, rysującej na ekranie klasę, który pojawi się w lewym górnym rogu okna aplikacji. Dwukrotne kliknięcie na utworzony element sprawia, że wyświetlone zostanie okno z możliwymi do edycji informacjami: nazwie klasy, jej polach i metodach. Zapis wprowadzonych zmian następuje po naciśnięciu przycisku „Save”, który jednocześnie zamyka okno podglądu elementu. Operacje dodawania oraz edycji standardowej klasy zostały przedstawione na rysunku 17. Możliwe jest również wprowadzanie zmian w klasach w głównym oknie aplikacji, jednak ze względu na niewielki rozmiar symbolizującego klasę prostokąta, jest to mniej wygodne. Od momentu utworzenia elementu, możliwe jest poruszanie nim po oknie programu.



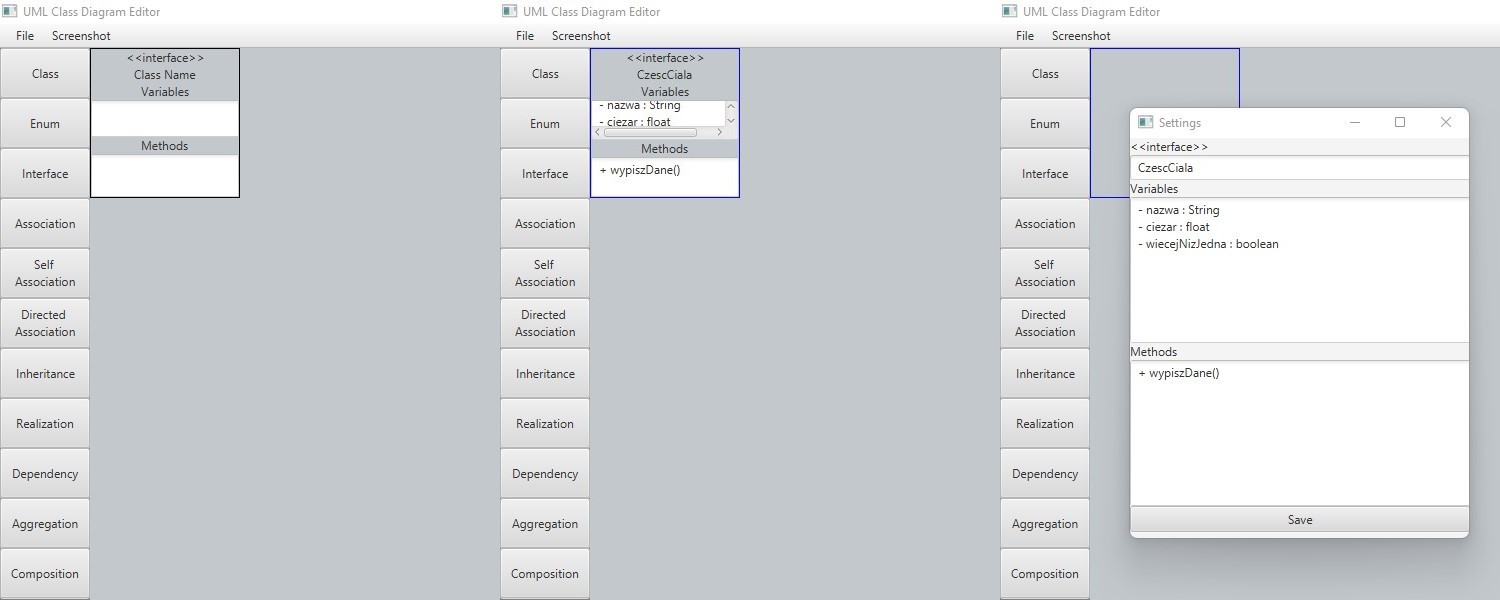
Rysunek 17. Tworzenie oraz edytowanie klasy w programie  
Źródło: opracowanie własne

## Rysowanie interfejsu i klasy wyliczeniowej

Dodawanie interfejsu lub klasy wyliczeniowej odbywa się w analogiczny sposób, jak w przypadku standardowej klasy. Metoda rysująca wywoływana jest poprzez naciśnięcie przycisku odpowiednio „Interface” lub „Enum”. Nowoutworzony obiekt klasy OtherClassEntity pojawia się w rogu ekranu. Edycja nazwy, pól i metod zachodzi w sposób identyczny do klasy standardowej. Etapy rysowania oraz modyfikacji interfejsu i typu wyliczeniowego przedstawione zostały na rysunkach 18 i 19.



Rysunek 18. Tworzenie oraz edytowanie klasy wyliczeniowej w programie  
Źródło: opracowanie własne

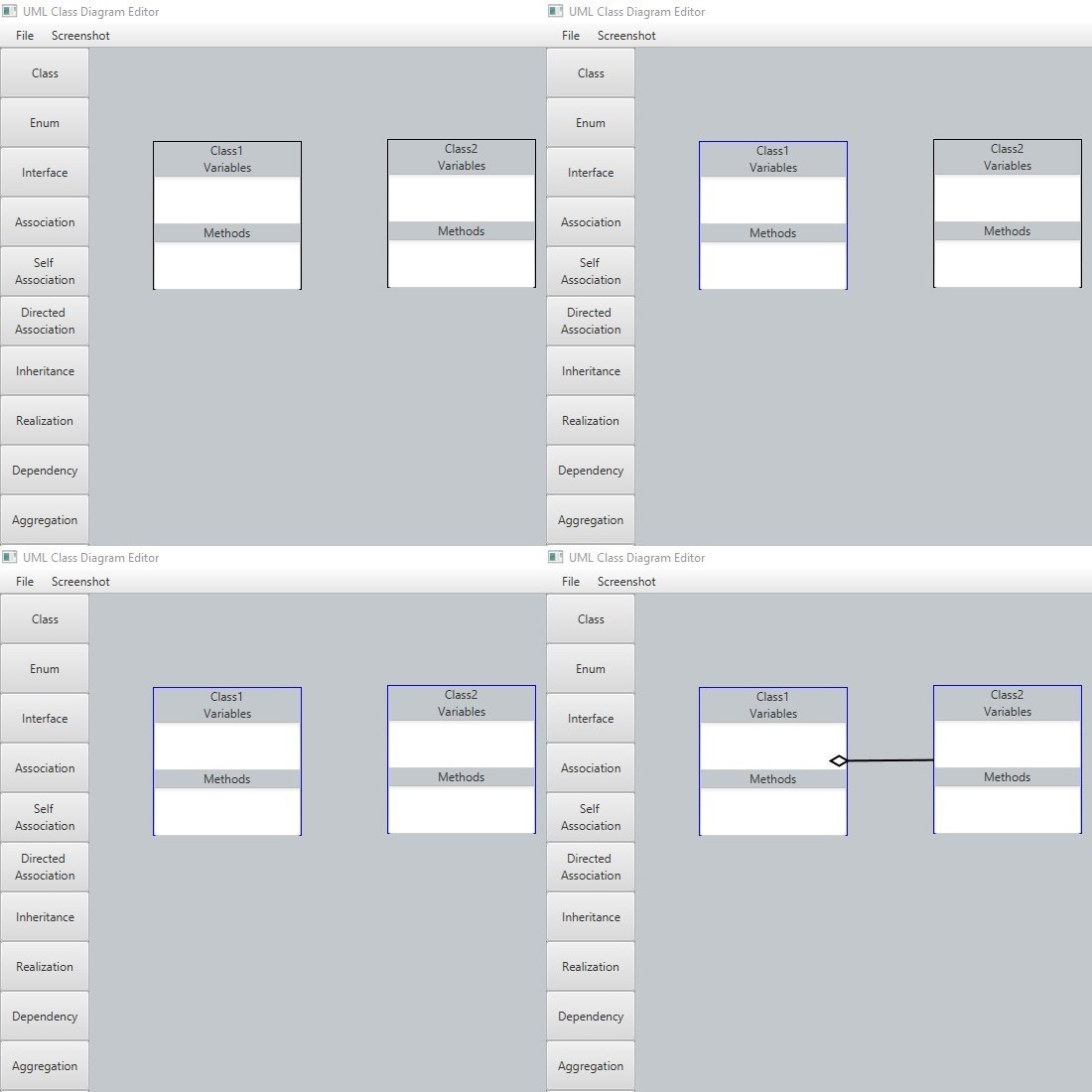


Rysunek 19. Tworzenie oraz edytowanie interfejsu w programie  
Źródło: opracowanie własne

## Tworzenie relacji między klasami

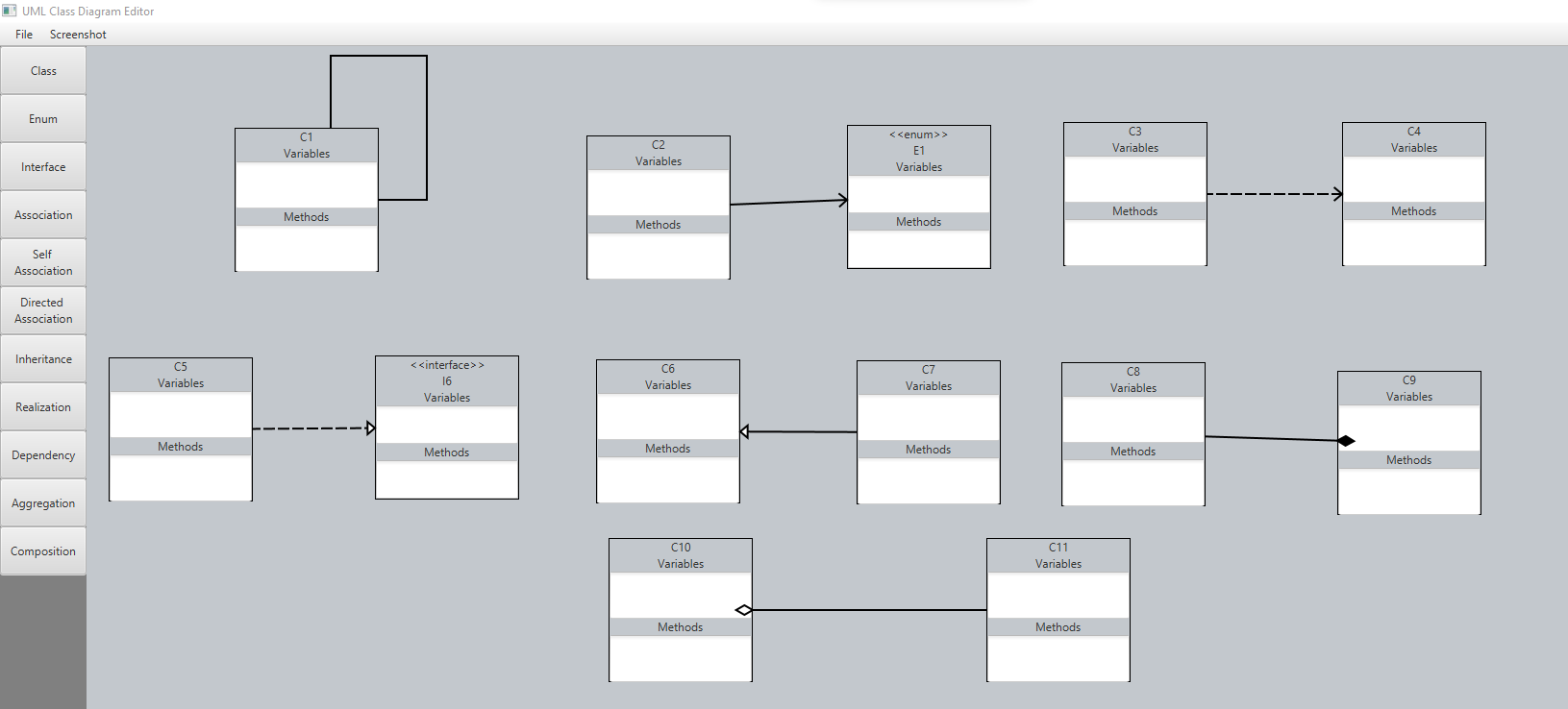
Istotą diagramu klas UML są związki, w które można łączyć elementy na diagramie. Program oferuje możliwość zamodelowania każdego typu relacji zaprezentowanej we wstępie. Przyciski na bocznym panelu odpowiadają nazwą pojedynczej relacji. Dodatkowo, wyróżnione zostały dwie dodatkowe możliwości: asocjacja skierowana oraz relacja asocjacji klasy z samą sobą.

Aby móc utworzyć relację, konieczne jest zaznaczenie dwóch różnych elementów poprzez naciśnięcie myszą na ich graficzne reprezentacje (wyjątkiem jest relacja *self association*). Wybrane do połączenia klasy zostają oznaczone poprzez zmianę koloru obramowania na niebieski. Istotne jest zapamiętanie, która klasa została oznaczona jako pierwsza, ponieważ to w jej stronę skierowany zostanie grot relacji. Po wybraniu elementów, połączenie następuje po naciśnięciu przycisku z nazwą relacji. Na rysunku 20 zaprezentowano etapy łączenia dwóch klas relacją agregacji: utworzenie dwóch klas, zaznaczenie najpierw klasy Class1, potem Class2, utworzenie relacji naciskając przycisk „Aggregation”.



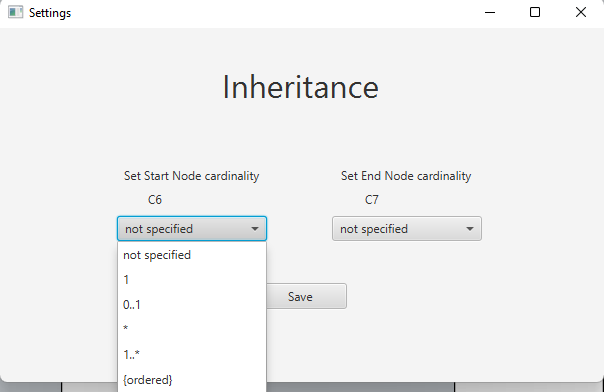
Rysunek 20. Etapy tworzenia relacji agregacji między klasami  
Źródło: opracowanie własne

Na rysunku 21 przedstawiono możliwe do utworzenia w programie relacje: asocjacja z samym sobą, asocjacja skierowana, zależność, realizowanie, dziedziczenie, kompozycja oraz agregacja.



Rysunek 21. Prezentacja dostępnych w programie relacji  
Źródło: opracowanie własne

Dwukrotne naciśnięcie na graficzną reprezentację relacji spowoduje otwarcie nowego okna zawierającego informacje o związku. 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 22 przedstawia okno relacji dziedziczenia klasy C6 przez C7.



Rysunek 22. Okno z danymi relacji  
Źródło: opracowanie własne

## Reakcje programu na próby utworzenia błędnych związków

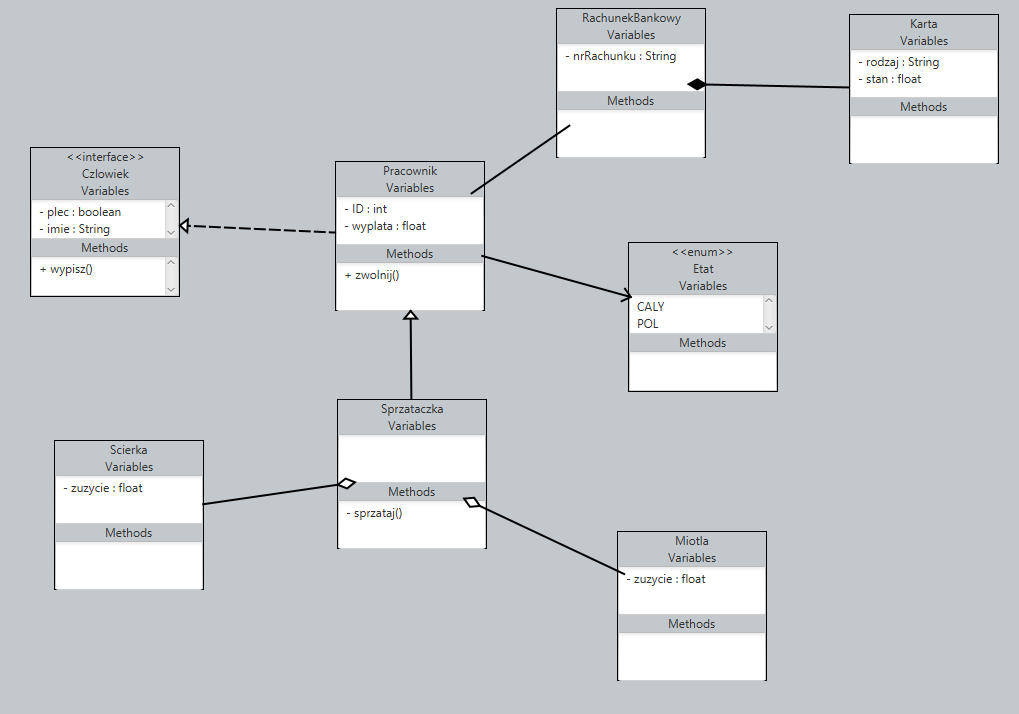
Głównym założeniem programu było wprowadzenie zabezpieczeń, które ograniczyłyby przypadki tworzenia niepoprawnych relacji. Zaimplementowane zostały wszystkie zasady opisane w rozdziale trzecim. Wprowadzony został także system informowania użytkownika o popełnieniu błędu w formie wyświetlających się okien, w których podpowiedziane zostały możliwe sposoby na łączenie tego typu relacji.

## Zapis diagramu do pliku

## Odczyt diagramu z pliku

## Zapis diagramu do pliku graficznego

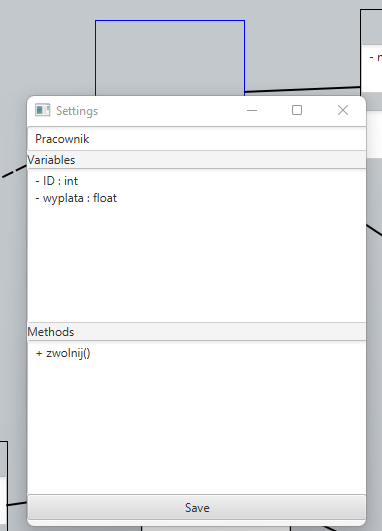
Cele jakie zostały postawione aplikacji to możliwość projektowania diagramów klas UML oraz kontrola poprawności modelowanych powiązań między elementami diagramu. Na rysunku 16 przedstawiono zrzut ekranu z wykonanej aplikacji, w której zaprojektowano przykładowy system w formie diagramu klas UML.



Rysunek 23. Diagram klas UML wykonany w aplikacji  
Źródło: opracowanie własne

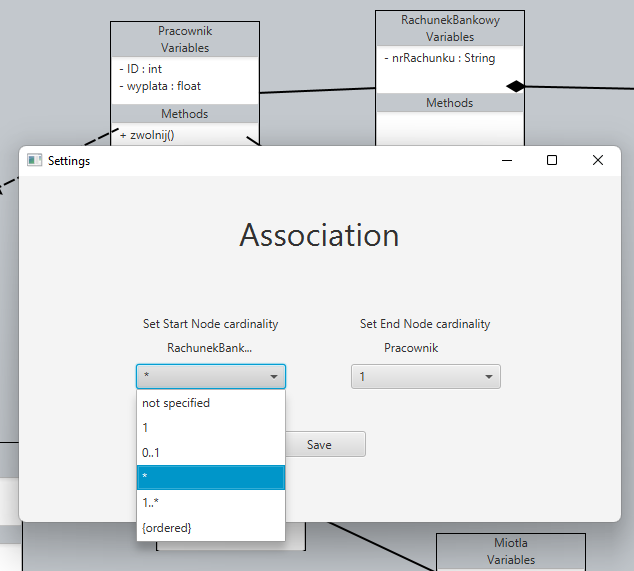
Na panelu bocznym, po lewej stronie, znajdują się przyciski odpowiedzialne za umieszczanie na ekranie klas oraz relacji, a przy pomocy pól menu na pasku u góry ekranu, możliwe jest wykonanie zapisu oraz odczytu danych, a także wykonanie zrzutu ekranu.

Otwarcie okna z danymi klasy dokonuje się poprzez podwójne kliknięcie na wybrany element diagramu. Na rysunku 17 przedstawiono wygląd okna z informacjami o klasie Pracownik, 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”.



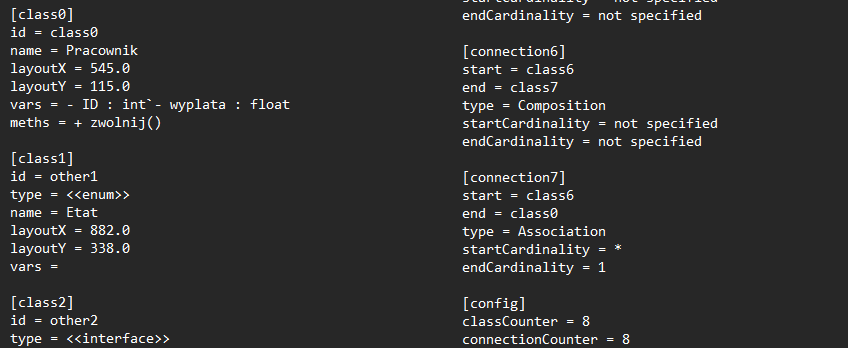
Rysunek 24. Okno z informacjami o klasie Pracownik  
Źródło: opracowanie własne

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”.



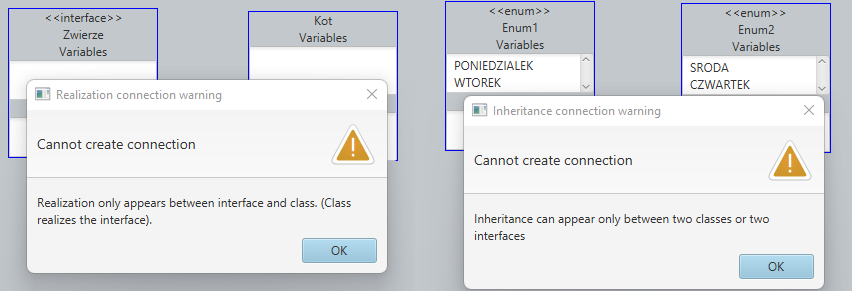
Rysunek 25. Okno z informacjami o związku między klasą Pracownik a RachunekBankowy  
Źródło: opracowanie własne

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 26. Zawartość pliku z zapisanymi danymi diagramu klas przedstawionego na rysunku 16  
Źródło: opracowanie własne

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 27. Informacje o próbie zamodelowania niepoprawnego połączenia między elementami  
Źródło: opracowanie własne

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

Przed programem postawiono dwa cele. Aplikacja powinna umożliwić projektowanie diagramów klas UML oraz kontrolować i uniemożliwiać tworzenie niepoprawnych połączeń między elementami (klasami).

Wszystkie założenia przedstawione w rozdziale trzecim zostały spełnione. Przy pomocy programu, możliwe jest zamodelowanie systemu w postaci diagramu klas UML, a tworzone niepoprawne połączenia między elementami diagramu zostaną wykryte, zgłoszone użytkownikowi oraz usunięte. Do projektowania, osoba wykorzystująca program ma do dyspozycji trzy typy elementów oraz sześć różnych relacji, w tym asocjację w trzech różnych wariantach. Elementami na diagramie można poruszać przy pomocy myszy oraz usuwać i edytować. Aby zapamiętywać i odtwarzać pracę, program wyposażony został w funkcje zapisu i odczytu pliku z danymi utworzonych diagramów oraz wykonania i zapisania zrzutu okna aplikacji w formie pliku graficznego .png.

# Bibliografia

1. R. Miles, K. Hamilton,UML 2.0 Wprowadzenie, Helion S.A., 2007, Gliwice
2. A. M. Fernandez-Saez, D. Caivano, M. Genero, M. R. V. Chaudron, On the Use of UML Documentation in Software Maintenance: Results from a Survey in Industry, IEEE, 2015, Ottawa
3. P. Langer, T. Mayerhofer, M. Wimmer, G. Kappel, On the Usage of UML: Initial Results of Analyzing Open UML Models, Gesellschaft für Informatik eV, 2014, Bonn
4. T. Stark, Y. Sedelmaier, D. Landes, J. Mottok, C. Wolff, Insights in Students’ Problems During UML Modeling, R. Reuter, IEEE 2020, Porto
5. K. Siau, P.-P. Loo, Identifying Difficulties in Learning UML, Information Systems Management 2006, Abingdon
6. The Java Language Specification, https://docs.oracle.com/javase/specs/jls/se8/jls8.pdf [dostęp 14.12.2022]
7. <https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2008-r2-and-2008/cc731332(v=ws.11)>
8. John Smith ”Jewelsea”, https://stackoverflow.com/questions/19748744/javafx-how-to-connect-two-nodes-by-a-line [dostęp: 13.12.2022]