Zawartość:

- diagramy UML

- gdzie i jak rysować

- diagram ERD

- diagram przypadków użycia

- diagram klas

- diagram aktywności

- diagram sekwencji

Diagram UML

Unified Modelling Language – graficzny system wizualizacji systemów/oprogramowania. Mówiąc prościej jest to standard diagramów, których zadaniem jest pokazanie jak działa i co oferuje dane oprogramowanie.

Rodzaje diagramów:

- diagram ERD – diagram pokazujący strukturę tabeli oraz relacje między nimi w bazie relacyjnej

- diagram klas – diagram pokazujący strukturę klas w aplikacji oraz relacje między nimi

- diagram przypadków użycia – diagram pokazujący funkcje aplikacji oraz aktorów (użytkowników)

- diagram aktywności – diagram pokazujący przebieg działania danej funkcji (mocno przypomina z założeń schemat blokowy stosowany do reprezentacji algorytmów)

- diagram sekwencji – pokazuje przepływ sterowania między klasami/funkcjami (która wywołuje którą/która komunikuje się z którą)

Po co to wszystko?

- diagram ERD to świetne narzędzie do tego by naświetlić komuś jak wygląda baza danych, zdecydowanie łatwiej będzie się zorientować z obrazka niż z opisu ustnego lub tekstowego

- diagram przypadków użycia idealnie nadaje się do pokazania klientowi opcji jakie zaoferuje mu oprogramowanie

- pozostałe diagramy mają charakter bardziej techniczny (bardziej powiązany z implementacją) przez co są bardziej przeznaczone dla programistów

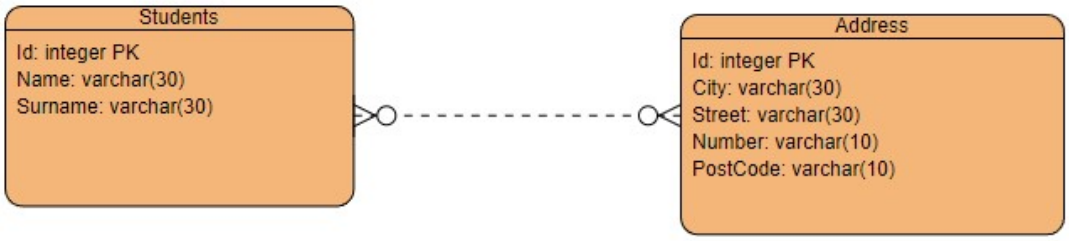
Jak tworzyć takie diagramy?

- bardzo dobrym portalem do tworzenia diagramów i rysunków (nie tylko UML) jest strona https://app.diagrams.net/

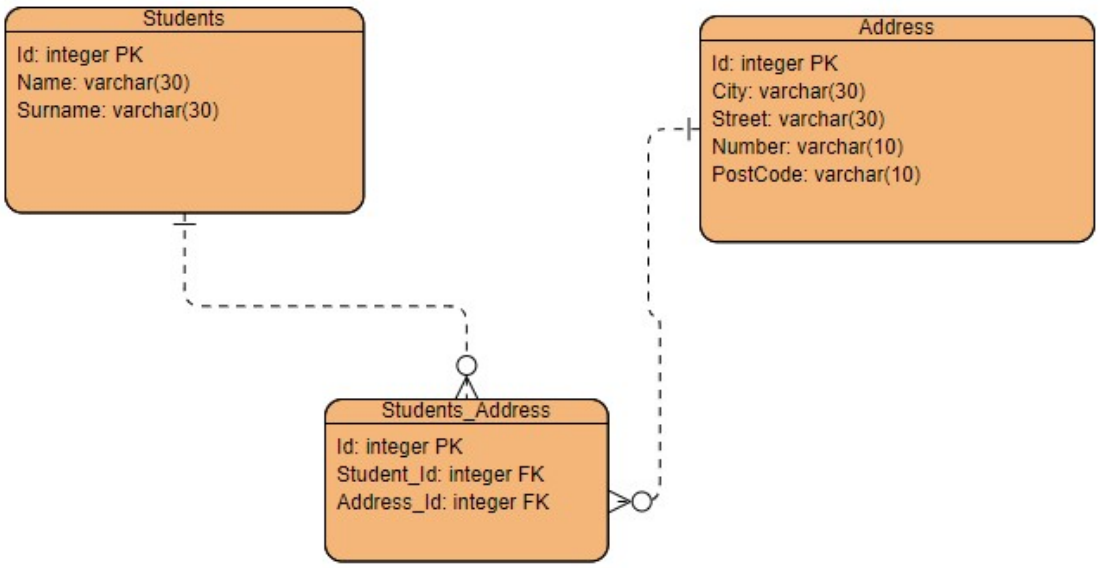
- tak stricte do UML mamy stronę https://online.visual-paradigm.com/

DIAGRAM ERD

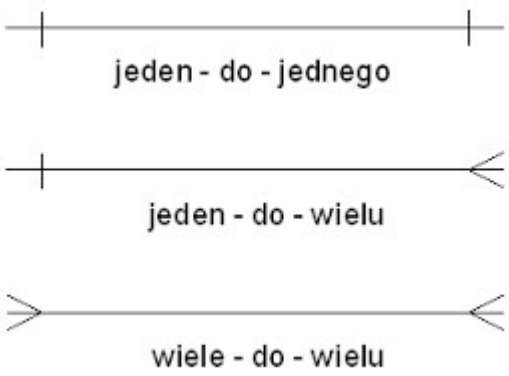
Pokazuje strukturę bazy danych czyli tabele (i ich strukturę) oraz relacje między nimi



Prosta sprawa. Wymieniamy kolumny i ich typy. Powyższa relacja wiele do wielu tak naprawdę wyglądać będzie tak:



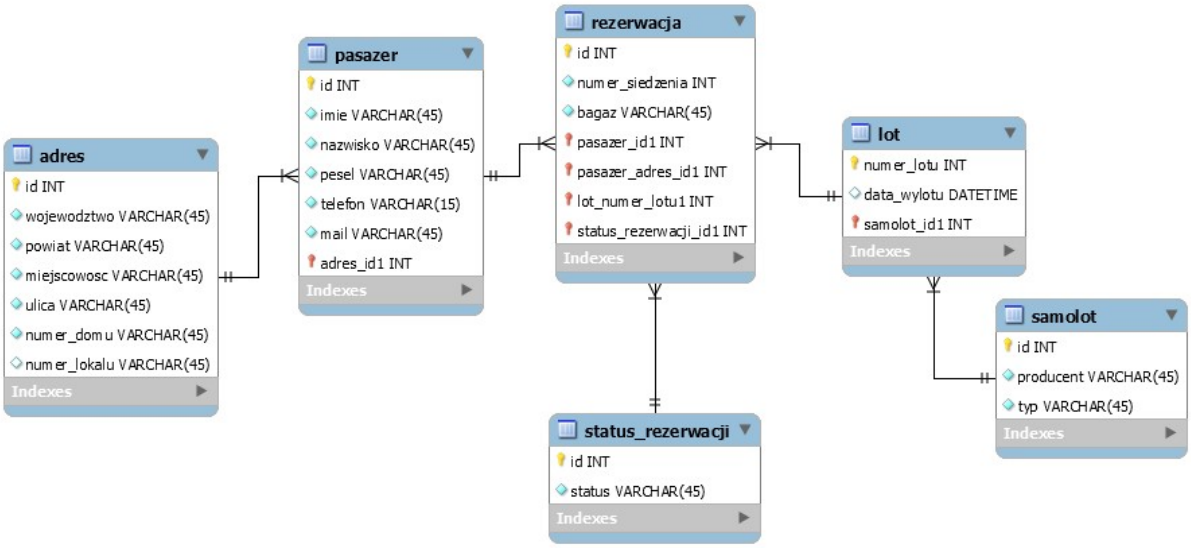
Rodzaje relacji:



Oczywiście kierunek jest istotny.

Dobrym narzędziem do rysowania diagramów ERD jest MySQL Workbench.

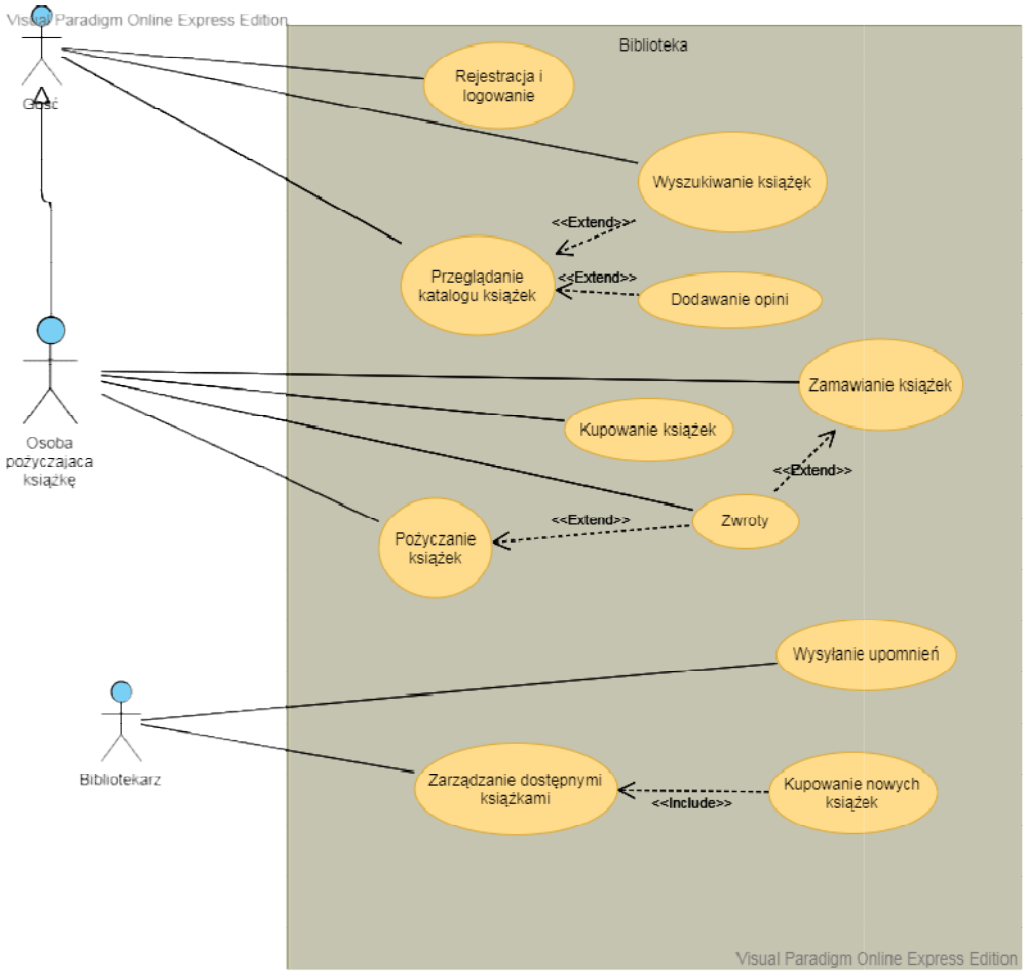
Przykład:



To taki typowy zrzut ekranu z MySQL Workbench. Kolor kropki przy polu bierze się z tego czy dane pole może być NULLem czy nie bądź czy jest kluczem czy nie.

DIAGRAM PRZYPADKÓW UŻYCIA:

Diagram taki zawiera listę funkcjonalności, powiązania między tymi funkcjami oraz aktorów.



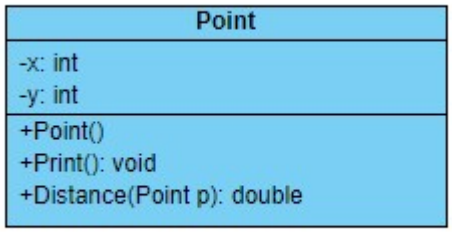
Powyżej znajduje się przykładowy diagram przedstawiający funkcjonalności aplikacji do obsługi biblioteki:

- Widzimy po lewej stronie aktorów
  - Każdy aktor ma przypisane funkcje, do których ma dostęp
  - Zastosowano mechanizm dziedziczenia gdzie aktor dziedziczący dostaje wszystkie funkcjonalności, do których dostęp miał aktor z którego dziedziczymy
  - Dziedziczenie oznaczamy strzałką z psutym grotem wskazującym aktora bazowego
- Widzimy listę funkcji
  - Pomiędzy funkcjami znajdują się również strzałki
  - Strzałka „Extend” oznacza, że dana funkcja rozszerza inną (ważny jest kierunek strzałki)
  - Strzałka „Include” oznacza, że dana funkcja zawiera się w innej (ważny jest kierunek strzałki)

DIAGRAM KLAS

Diagram klas zwiera strukturę klas w projekcie jak i relacje między nimi. Mowa tu zarówno o klasach typowo modelowych (klasy, które pełnią rolę foremki na dane) oraz klasach z logiką (klasy, które zawierają funkcje). Diagram klas często wykorzystywany jest do zaprezentowania wzorców projektowych.

Klasa:



Pierwsze co rzuca się w oczy to oddzielenie pól klasy od jej metod. Druga kwestia to symbole oznaczające dostępność + (dla elementów publicznych), - (dla elementów prywatnych), # (dla elementów chronionych), ~ (dla elementów internal czyli takich dostępnych tylko w danym projekcie (C#)).

Trzecia kwestia to konwencja zapisu.

-x: int

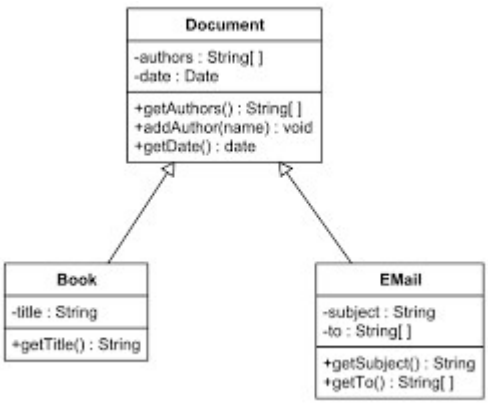
Czyli najpierw umieszczamy specyfikator dostępu a później po dwukropku typ wyrażenia.

W przypadku metod wygląda to podobnie:

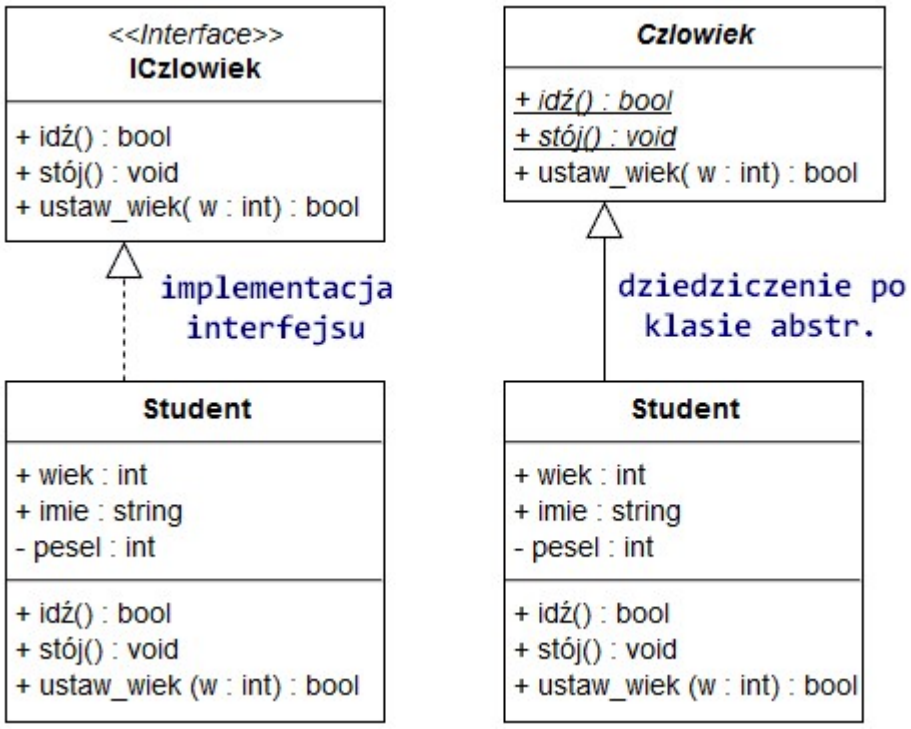
+Print(): void

Gdzie po dwukropku umieszczamy typ zwracany.

Dziedziczenie w przypadku diagramu klas:



Jak widać na obrazku zastosowano tu podobną konwencję jak w przypadku diagramu użycia a mianowicie strzałkę z pustym grotem wskazującą klasę bazową. Książka dziedziczy z dokumentu. Tak samo mail dziedziczy z dokumentu.



Interfejsy w C# oznacza się za pomocą <<Interface>> oraz nazwy zaczynającej się od I. Klasy abstrakcyjne bardzo często oznacza się kursywą.

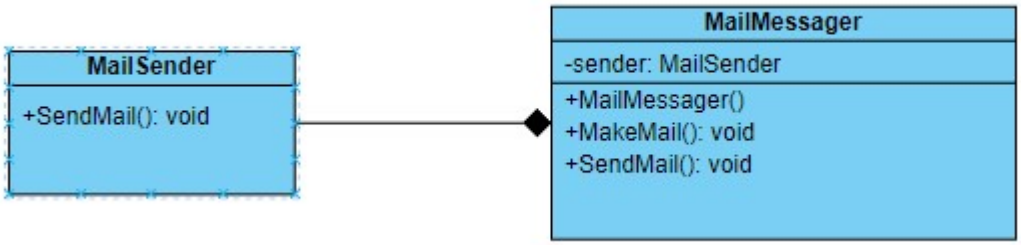
Zależności:

Zależność to przypadek, w którym klasa A korzysta z funkcjonalności klasy B. Aby to było możliwe klasa A posiada w swoim polu instancję klasy B.

Zależność twarda (bez stosowania zasady wstrzykiwania zależności)

```
class MailSender
{
    1 reference
    public void SendMail() { }
}

0 references
class MailMessager
{
    private MailSender sender = new MailSender();

    0 references
    public void MakeMail() { }
    0 references
    public void SendMail() { sender.SendMail(); }
}
```

Jej diagram:



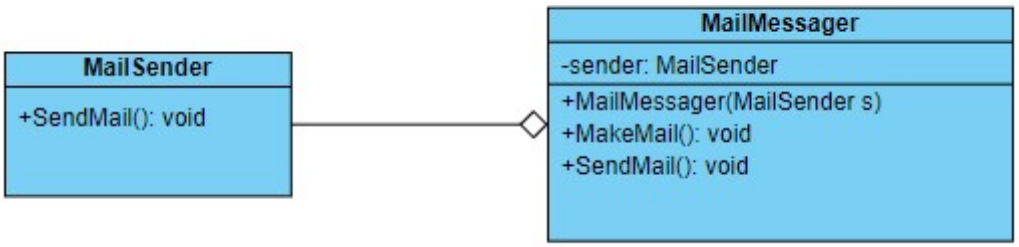
Zależność czyli obiekt klasy, z której korzystamy jest tutaj umieszczony w polu „na sztywno”. W momencie tworzenia instancji MailMessager nie jesteśmy w stanie zmienić tej zależności.

Więcej na ten temat pisałem w opisie Dependency Injection oraz wzorców projektowych.

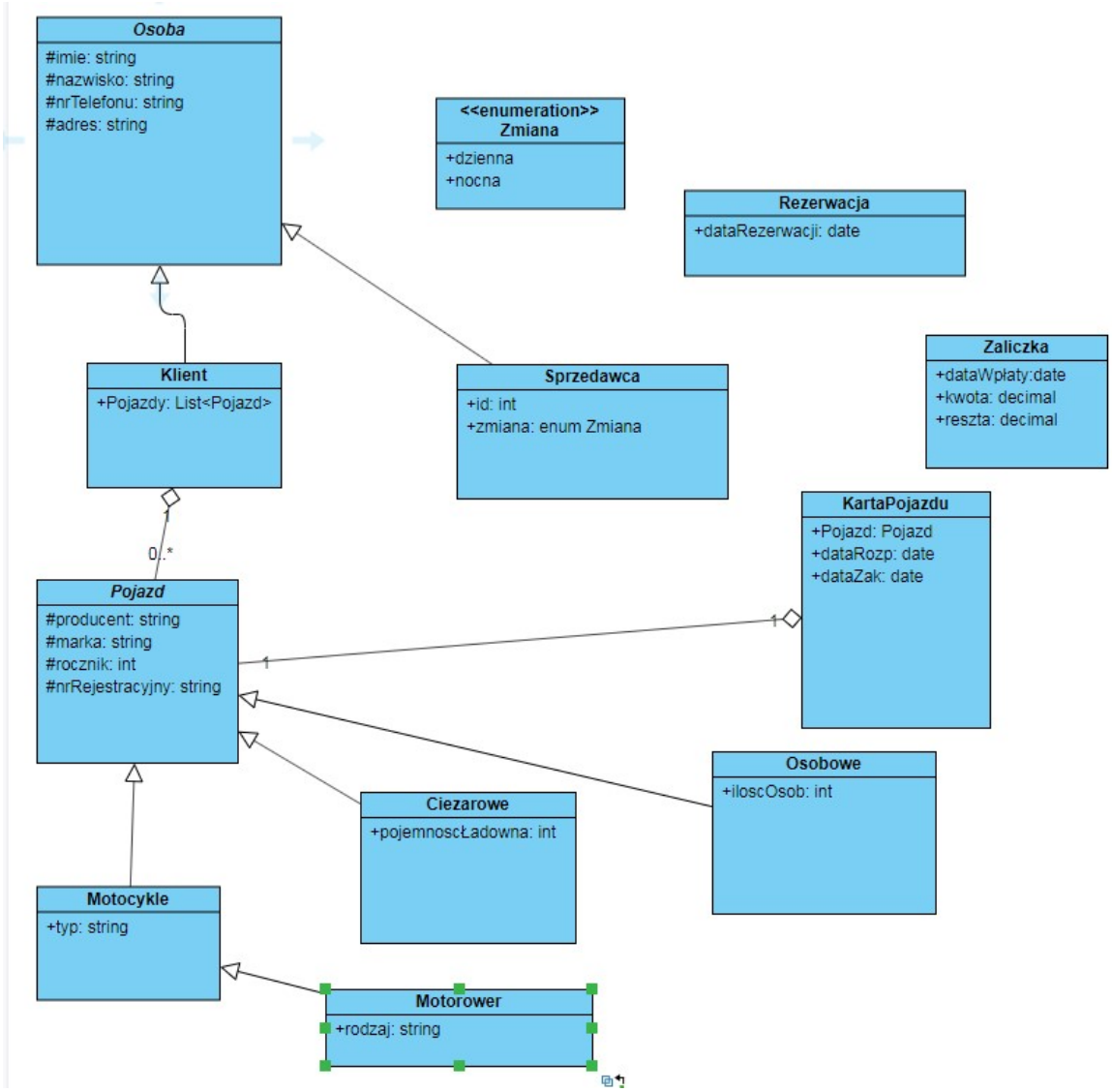
Zależność miękka (stosująca zasadę wstrzykiwania zależności)

```
class MailSender
{
    1 reference
    public void SendMail() { }
}

1 reference
class MailMessager
{
    private MailSender sender;

    0 references
    public MailMessager(MailSender sender)
    {
        this.sender = sender;
    }

    0 references
    public void MakeMail() { }
    0 references
    public void SendMail() { sender.SendMail(); }
}
```

Oraz jej diagram:



Zastosowano tu wstrzykiwanie zależności do pola poprzez konstruktor. Takie podejście jest bardziej mobilne. Różnicę widać w nagłówku konstruktora oraz łączącej klasy „strzałce”.



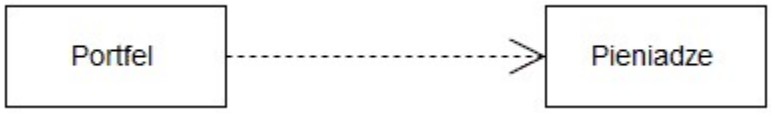
Czasami w przypadku zależności pojawia się coś takiego jak multiplikator. Na powyższym schemacie można go zauważyć między klasą Klient a Pojazd. Klasa Klient ma pole z listą pojazdów. Naturalne jest, że lista u konkretnego klienta może przechowywać 0 lub więcej pojazdów (obiektów). W drugą stronę jest inaczej. Każdy pojazd ma jednego właściciela.

Zależności w formie parametrów:

Czasami zdarza się, że metoda klasy używa obiektu innej klasy przyjmując go po prostu w parametrze jak w poniższym fragmencie kodu (nie przypisuje go do pola na dłużej).

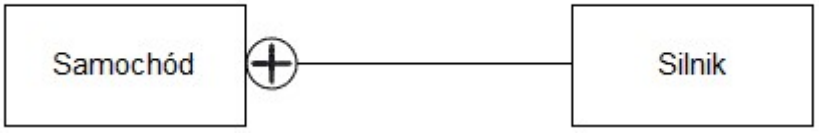
```
public class Portfel
{
    public void Dodaj(Pieniadze pieniadze) { }
}
```

Metoda Dodaj klasy Portfel używa obiektu klasy Pieniadze. I na tym relacja się kończy. Takie zależności oznaczamy tak:



* diagram jest uproszczony – bez pól i metod

W przypadku diagramów klas została jeszcze jedna kwesta. Chodzi o klasy w klasie:



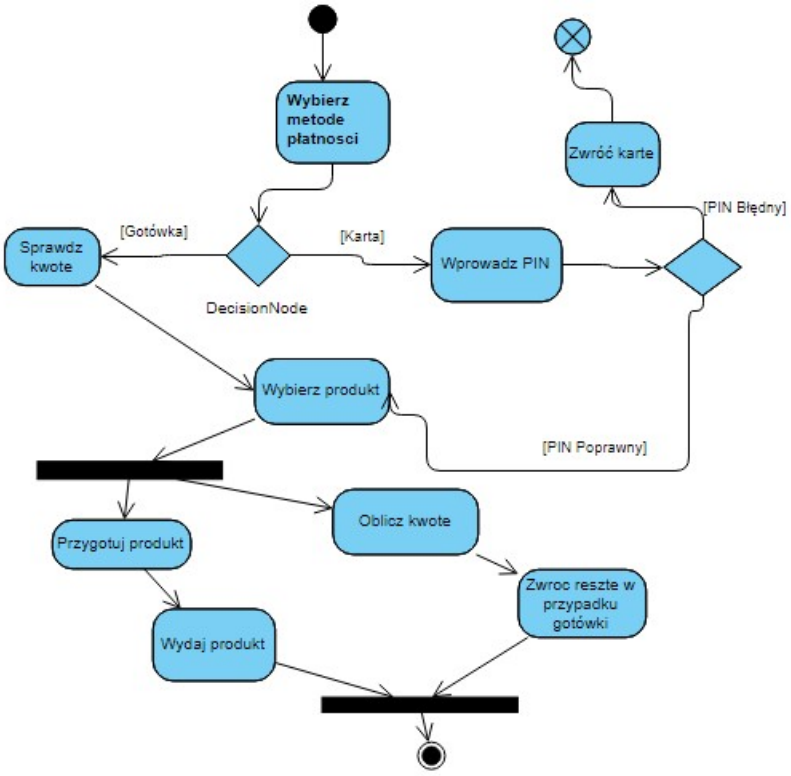

```
public class Samochod
{
    private string _marka;

    private class Silnik
    {
        private int _moc;
    }
}
```

Klasa Silnik widoczna jest do użycia tylko wewnątrz klasy Samochód (mało popularna zagrywka szczerze mówiąc).

DIAGRAM AKTYWNOŚCI

Diagram pod względem zastosowania bardzo przypomina schemat blokowy algorytmów, ale wizualnie od niego odbiega przez zastosowanie innych kształtów i figur do poszczególnych zadań.

Diagram aktywności pokazuje przebieg pewnego procesu/funkcji.

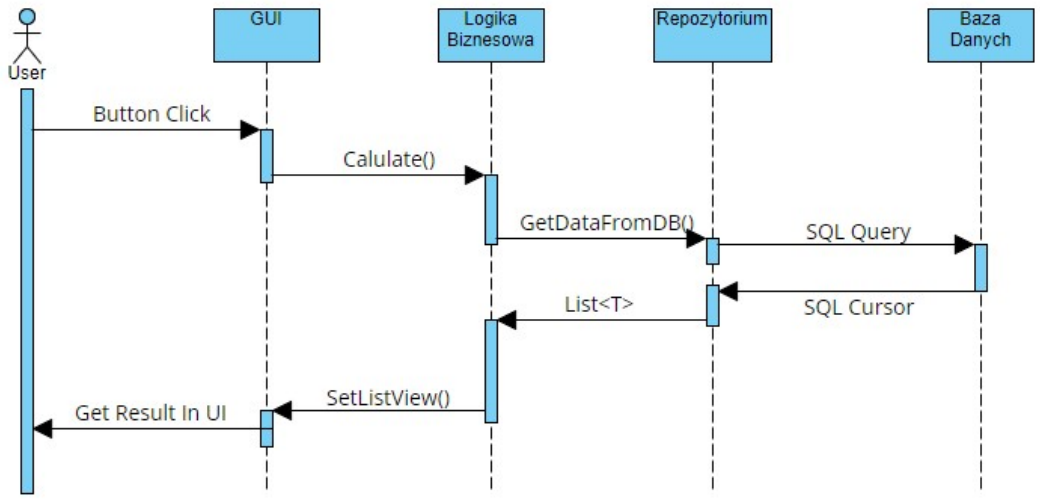


Tak wygląda proces opłacania produktu w aplikacji. Nic szczególnego. Jedyna nowość względem schematu blokowego algorytmów to możliwość zrównoleglenia (wykonywania kilku rzeczy na raz). Jak widać na schemacie pojawiły się dwie poziome kreski. Pierwsza to rozwidlenie na kilka ścieżek. Druga natomiast działa odwrotnie. Nie można pójść dalej z bloku, który łączy kilka ścieżek do momentu aż wszystkie ścieżki tam dotrą.

Inną nowością jest blok startu i końca.

DIAGRAM SEKWENCJI

Diagram sekwencji pokazuje przepływ sterowania między elementami systemu. Takim elementem może być klasa lub coś bardziej ogólnego jak na poniższym przykładzie:



Mamy tutaj pięć elementów. Ogólne sterowanie przebiega z góry do dołu zgodnie ze strzałkami. Niebieski pasek oznacza aktywność danego elementu.

Użytkownik klika przycisk na stronie. Przycisk ten odwołuje się do jakiejś metody klas logiki biznesowej. Metoda ta do wykonania zadania potrzebuje trochę danych z bazy. Prosi więc klasę repozytorium (taki wzorzec) lub ORMa (biblioteka do obiektowego operowania na bazach relacyjnych) o dane. Repozytorum lub ORM wykonują zapytanie o dane do bazy i zwracają je do pytającego czyli do logiki biznesowej. Tak wykonuje obliczenia i ustawia UI dla użytkownika.

Oczywiście ten diagram jest mocno uproszczony. Zazwyczaj te ścieżki są dużo bardziej poplątane i zagmatwane.