# UML

# Spis treści



# Things

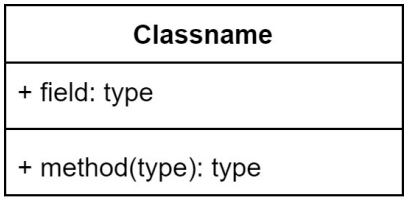
W dosłownym tłumaczeniu "rzeczy". Oznacza byty lub obiekty ze świata rzeczywistego. W obrębie **Things** możemy wyróżnić kilka kategorii, które są opisane poniżej. Przykłady użycia zostaną pokazane jak przejdziemy do przykładowych diagramów.

## Structural Things

Rzeczowniki, symbole opisujące statyczny wygląd modelu naszej aplikacji. Zalicza się do nich klasy, obiekty, interfejsy, komponenty.

### Class

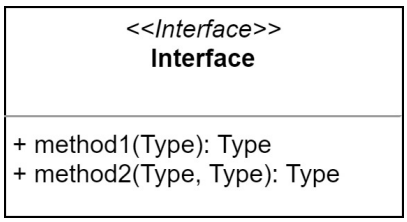
Reprezentuje klasę lub klasę abstrakcyjną. Wizualizacja jest następująca:




*Obraz 1. UML Class*

## Interface

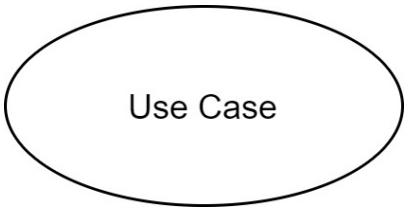
Zestaw operacji opisujący pewną funkcjonalność. Metody zdefiniowane w interface są implementowane gdy implementowany jest interface.




*Obraz 2. UML Interface*

## Use case

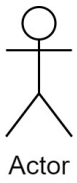
Przypadek użycia to zestaw działań, które należy wykonać aby osiągnąć jakiś zdefiniowany cel.




*Obraz 3. UML Use Case*

## Actor

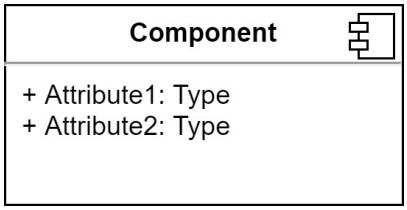
Używany razem z diagramami przypadków użycia (use case). Obiekt, który wchodzi w interakcję z systemem, może to być np. użytkownik.




*Obraz 4. UML Actor*

## Component

Przedstawia element systemu, który może być komponentem biznesowym lub fizycznym. Np. Silnik wyszukiwań lub Moduł do składania zamówień.



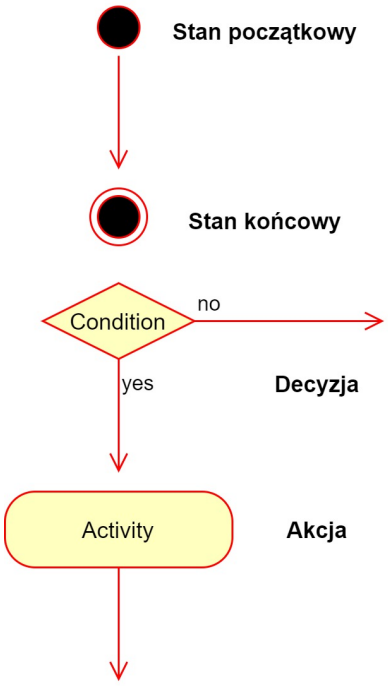

*Obraz 5. UML Component*

# Behavioral Things

Są to czasowniki zdefiniowane w modelu języka **UML**, które opisują dynamiczne aspekty diagramu, służą do przedstawiania zachowania systemu. Możemy wyróżnić m.in:

## Activity Diagram

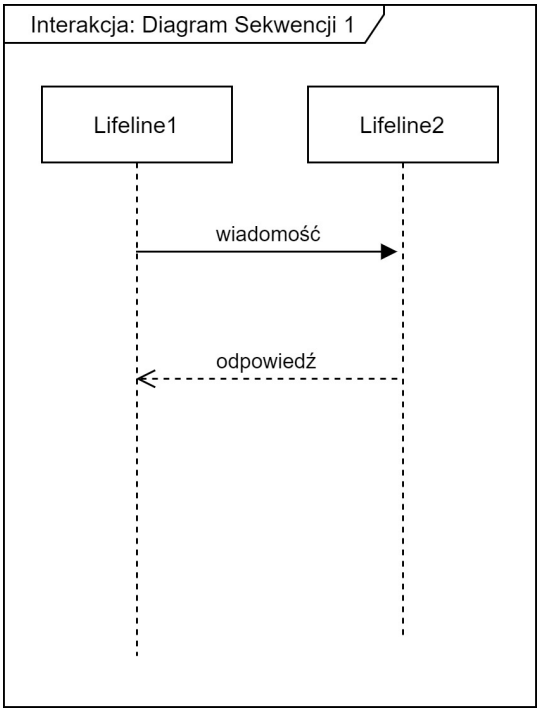
Diagram aktywności przedstawia czynności wykonywane przez różne części systemu. Definiuje sekwencję akcji jaką należy wykonać aby dojść od stanu początkowego do stanu końcowego. W skład takiego diagramu wchodzą elementy takie jak: stan początkowy, końcowy, okienko decyzyjne oraz oznaczenie akcji.




*Obraz 6. UML Activity Diagram*

## Interaction Diagram

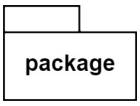
Diagram interakcji służy do wizualizacji przepływu informacji pomiędzy różnymi fragmentami systemu. Notacja składa się z elementów takich jak **Interakcja** i **Lifeline**. Interakcja jest definiowana jako zachowanie, które oznacza wymianę komunikatów między elementami w celem realizacji jakiegoś zadania. Lifeline odnosi się do linii wskazującej moment rozpoczęcia i zakończenia czasu trwania wiadomości.




*Obraz 7. UML Interaction Diagram*

# Grouping Things

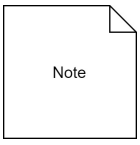
Są to paczki, które służą do grupowania powiązanych ze sobą elementów w jedną jednostkę organizacyjną.




*Obraz 8. UML Package*

# Annotational things

Są to notatki, które można zapisywać w modelu aby dodać istotne komentarze lub informacje. Wygląda jak karteczka samoprzylepna z zagiętym rogiem.




*Obraz 9. UML Note*

# Relationships

Relacje służą do tego, żeby pokazać w modelu jak jedna lub więcej encji/bytów są ze sobą powiązane. Dzięki temu możemy pokazać np. dziedziczenie albo kompozycję. Wyróżniamy 6 rodzajów relacji, które są wymienione poniżej. Przykłady użycia zostaną pokazane jak przejdziemy do przykładowych diagramów. Rysunki strzałek pojawiają się na końcu.

# Realization/Implementation

W przypadku tej relacji, jeden byt opisuje pewne zachowanie, które nie jest zaimplementowane, natomiast inne elementy implementują tę odpowiedzialność. Relacja ta jest używana do oznaczania implementacji interfejsów. Oznaczenie to linia przerywana z pustym grotem strzałki na końcu. Strzałka wskazuje na klasę nadrzędną.

# Generalization/Inheritance

Przedstawia związek pomiędzy klasą nadrzędną (**superklasą**), a klasą podrzędną (**subklasą**). Służy do określenia relacji dziedziczenia. Oznaczenie to linia prosta z pustym grotem strzałki na końcu. Strzałka wskazuje na klasę nadrzędną.

# Dependency

Relacja ta oznacza, że jedna klasa używa innej. Przykładowo jedna klasa używa innej w parametrze metody lub jako typ zwracany z metody. Przykładowo:

```
BigDecimal.valueOf(String);
```

`BigDecimal` w tym przykładzie jest zależne od `String`. Jeżeli jedna klasa ma zdefiniowaną drugą jako swoje pole, to nie jest to **Dependency** tylko **Association** (czyli kolejna relacja). Oznaczenie to linia przerywana ze strzałką po jednej stronie.

# Association

Relacja między dwiema oddzielnymi klasami, która łączy ze sobą dwie odrębne encje (byty). Relacja ta określa ile elementów bierze udział w relacji. Możemy tutaj nanieść ilość elementów biorących udział w interakcji. Typowy przykład w Javie to pole w klasie. Przykładowo klasa `Employee` może mieć pole:

```
String name;
```

Taka relacja jest wtedy nazywana **Asocjacją**.

**Association** jest jednocześnie generalizacją dwóch terminów jakimi są **Aggregation** oraz **Composition** (które są omówione niżej). Oznacza to, że **Aggregation** oraz **Composition** są podzbiorami **Asocjacji**. W obu przypadkach (**Aggregation** oraz **Composition**) relacja między klasami jest widoczna jako pole w klasie, różni je natomiast "właściciel", zaraz się to wyjaśni. Oznaczenie **Asocjacji** to linia ciągła, gdzie strzałki mogą się znaleźć po obu stronach.

# Aggregation

Odnosząc się do stwierdzenia o właścicielu. **Agregacja** jest rodzajem relacji, w której jeden obiekt może istnieć oddzielnie bez drugiego, jako samodzielny byt. Przykładem może być koło i samochód. Koło może swobodnie istnieć bez samochodu. Oznaczenie to linia ciągła z pustym rombem na końcu.

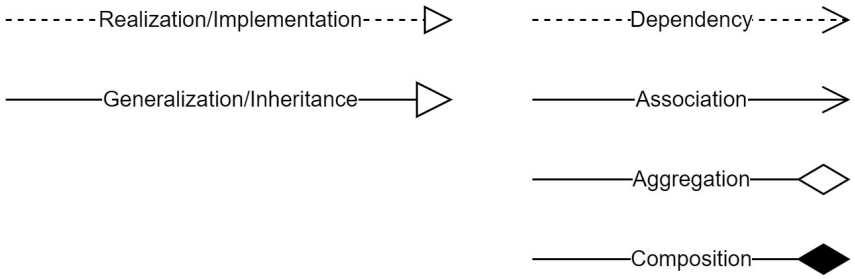
# Composition

**Kompozycja** jest bardzo podobna do **Agregacji**, przy czym różnica jest taka, że w przypadku kompozycji jeden obiekt nie może samodzielnie istnieć bez innego. Sytuacja taka jest nazywana "**death relationship**". Przykładem może być mieszkanie i blok, mieszkanie nie istnieje bez bloku. Zatem można powiedzieć, że blok posiada mieszkania.

Czyli podsumowując **Association**, **Aggregation** oraz **Composition**, bo te 3 najczęściej się ze sobą mylą. **Asocjacja** jest generalnym terminem, który określa, że jedna klasa korzysta z funkcji dostarczanych przez inną klasę. **Kompozycja** oznacza relację, gdzie jeden obiekt nie może samodzielnie funkcjonować bez innego obiektu. **Agregacja** natomiast pozwala na to, żeby jeden obiekt istniał samodzielnie, nie jest potrzebny do tego inny obiekt, który go "posiada".

Artykuły dotyczące **Association**, **Aggregation** oraz **Composition**: link1 oraz link2.

**Poniżej wizualizacja wszystkich strzałek:**




*Obraz 10. UML Relationships*

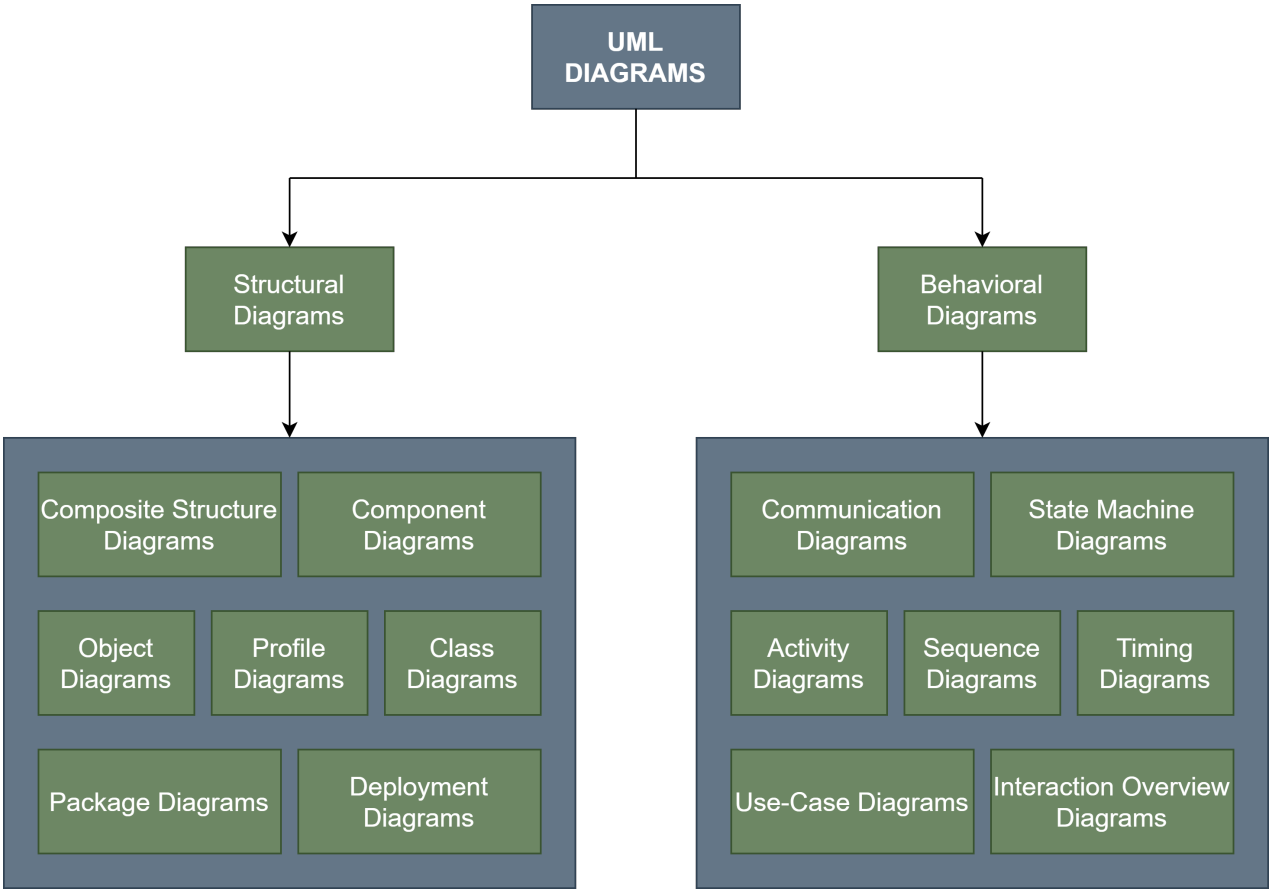
W lewej kolumnie umieszczona została grupa związana z "dziedziczeniem". W prawej kolumnie grupa związana z "zawieraniem". Przy czym prawa grupa została ułożona od relacji najsłabszej do najsilniejszej patrząc od góry. Przejdziemy do przykładów gdy będziemy omawiać konkretne diagramy.

# Diagrams

Diagramy są graficzną implementacją, wizualizacją fragmentów oprogramowania. W standardzie **UML** wyróżnia się 14 różnych typów diagramów, natomiast my skupimy się na pokazaniu w praktyce tylko kilku najczęściej używanych. Każdy diagram ma swój zestaw symboli. Symbole te zostały pokazane wcześniej. Każdy diagram przedstawia inną perspektywę systemu, pozwala spojrzeć na system z innego kąta. Diagramy podzielone są na 2 kategorie:

- Structural Diagram
- Behavioral Diagram

Oczywiście jak możesz się domyślić, w skład każdej kategorii wchodzi pewna ilość diagramów.




*Obraz 11. UML Diagram types*

# Structural Diagram

Diagramy strukturalne służą do przedstawienia statycznego widoku systemu, do pokazania jego struktury. Schemat struktury ma za zadanie przedstawić różne elementy/obiekty w systemie lub aplikacji. Możemy wyróżnić diagramy strukturalne takie jak:

- Class diagram - diagram Klas
- Object diagram - diagram Obiektów
- Profile diagram - diagram Profili
- Package diagram - diagram Paczek
- Component diagram - diagram Komponentów
- Deployment diagram - diagram Wdrożenia
- Composite Structure diagram - diagram struktur złożonych

# Behavioral Diagram

Diagramy behawioralne mają za zadanie przedstawić zachowanie, funkcjonowanie systemu. Są to diagramy, które wyrażają elementy dynamiczne systemu. Służą one do wizualizacji przepływu danych pomiędzy różnymi elementami systemu lub aplikacji. Diagramy te służą do pokazania interakcji pomiędzy fragmentami systemu wraz ze sposobem przepływu tych danych. Możemy wyróżnić diagramy behawioralne takie jak:

- Activity diagram - diagram Aktywności
- State machine diagram - diagram Maszyny Stanów
- Use case diagram - diagram Przypadków Użycia
- Sequence Diagram - diagram Sekwencji
- Communication Diagram - diagram Komunikacji
- Interaction Overview Diagram - diagram Przeglądu Interakcji
- Timing Diagram - diagram Czasowy

Stosując diagramy **UML** możemy pokazać plan i strukturę aplikacji lub systemu. Można powiedzieć, że **Things**, **Relationships** i **Diagrams** to trzy bloki składowe składni języka **UML**. Things to byty, które odwzorowujemy ze świata rzeczywistego. Relacje są używane do opisania związku pomiędzy jednym lub kilkoma elementami. Natomiast konkretne rodzaje diagramów służą do wyrażenia różnych aspektów systemu informatycznego. Pozwalają na niego spojrzeć z różnej perspektywy.

Mając już omówione podstawowe elementy konstrukcyjne wymienionych diagramów, możemy przejść do omówienia kilku najbardziej popularnych na praktycznych przykładach.