Opracowanie Obrona - Informatyka przemysłowa

**Autor:** Michał Kapiczyński

# 1. Cechy charakterystyczne i zastosowania programowania obiektowego

**Programowanie obiektowe** - jest jednym z paradygmatów programowania, czyli sposobu patrzenia programisty na przepływ sterowania i wykonywania programu komputerowego. Zgodnie z nim programy definiuje się za pomocą obiektów, komunikujących się między sobą w celu wykonania określonego zadania.Obecnie jest to najpopularniejsza technika programistyczna.

Programowanie obiektowe jest potoczną, ogólnie przyjętą nazwą programowania zorientowanego obiektów.

Istnieje jeszcze:

**Object based design -** programowanie oparte na obiektach - paradygmat, który wykorzystuje jedynie pojęcie obiektu i związaną z tym hermetyzację kodu i ochronę pól,

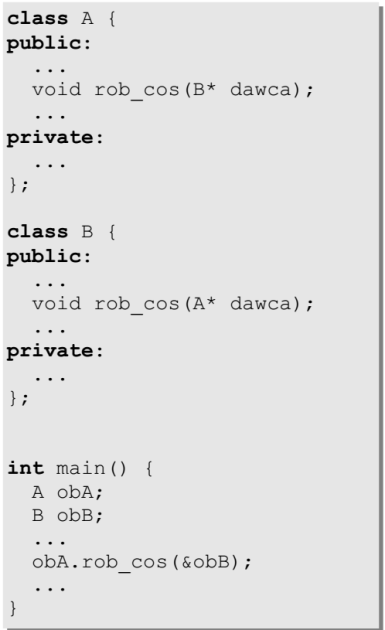
**Object orientated programming -**  programowanie zorientowane obiektowo - tutaj mamy do czynienia z czymś więcej - tworzenie hierarchii klas i obiektów, definicja ich wzajemnych zależności, zmiana zachowania klasy w zależności od typów itd.

**Obiekt -** materialny lub abstrakcyjny byt, który można wyizolować ze środowiska,

**Obiekt (bardziej programistycznie** - struktura danych, która występuje łącznie z operacjami dozwolonymi do wykonania na niej, Obiekt może być złożony tzn. składać się z innych obiektów.

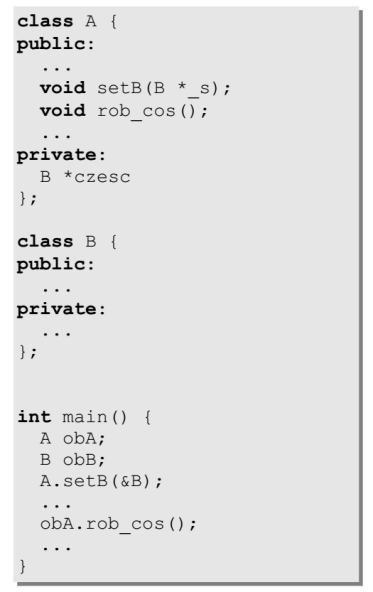
**Obiekt jest charakteryzowany poprzez:**

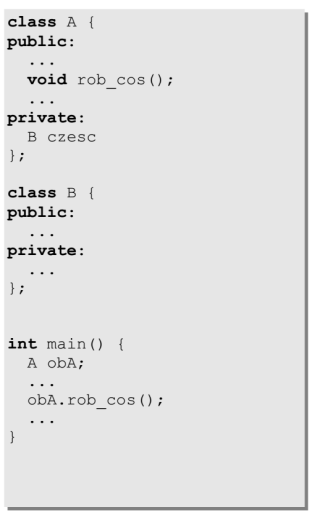
* tożsamość, która odróżnia go od innych obiektów (miejsce zajmowane przez ten obiekt w pamięci komputera),
* stan, który może zmieniać się w czasie (bez zmiany tożsamości obiektu). Stan obiektu w danym momencie jest określany przez aktualne wartości atrybutów i powiązań z innymi obiektami.
* zachowania do niego przypisane, tj. zestaw operacji, które wolno stosować na danym obiekcie,

**Atrybuty obiektów mogą być:**

* proste,
* złożone,
* wskaźnikowe,
* powtarzalne,
* opcjonalne,
* domyślne,
* pochodne (wyliczane),
* wspólne dla zestawu obiektów,
* inne obiekty.

**Relacje między obiektami:**

****

* **Skojarzenie -** obiekt typu A wykorzystuje obiekt typu B i/lub B wykorzystuje A w celu wykonania swoich zadań. A i B są tworzone i pamiętane zupełnie niezależnie.
* **Agregacja** - obiekt typu A zawiera w sobie obiekt typu B, ale oba są tworzone niezależnie od siebie. Obiekt typu A może składać się z wielu obiektów innych typów.
*  **Kompozycja** - obiekt typu A zawiera w sobie obiekt typu B i jest jego panem i władcą tzn. A zarządza czasem życia obiektu typu B - A tworzy i niszczy B. Obiekt B nie może istnieć (nie ma racjonalnego wyjaśnienia takiej możliwości) bez obiektu A.
* **Uogólnienie -** obiekt typu B dziedziczy po obiekcie typu A. A jest klasą nadrzędną, B - podrzędną.

**Przykład obiektu:** Samochód:

Jego atrybutami mogą być:

* pozycja,
* prędkość,
* stan paliwa,
* części składowe np. silnik, koła,

Czynności, które może wykonywać:

* start silnika,
* przemieszczanie się,
* przyspieszanie,
* hamowanie,
* ...

**Klasa**  to szablon obiektu:

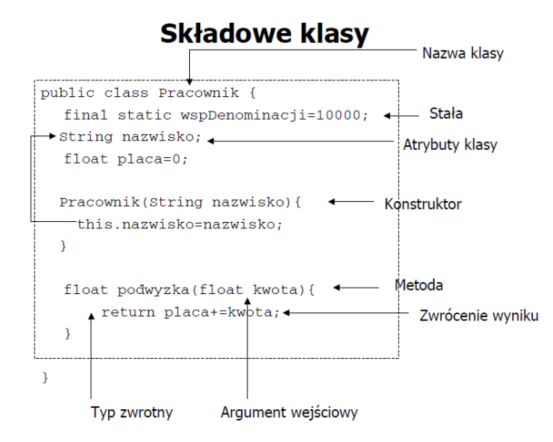
* Jest miejscem przechowywania (definiowania) tych własności grupy obiektów, które są niezmienne (co do definicji, a nie wartości) dla wszystkich członków grupy,
* Ogranicza kontekst, w którym odwołania do obiektu może być użyte w programie,
* Dobrze zbudowana klasa jest starannie wydzieloną abstrakcją pochodzącą ze słownictwa dziedziny danego problemu.
* Obejmuje pewien mały, dobrze określony zbiór zobowiązań, z których jest w stanie się w pełni wywiązać. Zapewnia oddzielenie specyfikacji abstrakcji od jej implementacji. Jest zrozumiała i prosta, a przy tym rozszerzalna i dająca się łatwo dostosować do potrzeb.

Struktura klasy:

* pola - atrybuty (różne wartości dla różnych instancji),
* metody - (takie same dla wszystkich).

Ochrona danych - definiowanie obszaru widoczności zmiennej lub funkcji:

* publiczne (public) - dostępne z dowolnego miejsca w programie,
* prywatne (private) - dostępne jedynie z wnętrza danej klasy,
* chronione (protected) - dostępne z wnętrza danej klasy i klas pochodnych.



**Podstawowe zasady programowania obiektowego:**

* Zasada dekompozycji (metoda walki ze złożonością) **-** rozdzielenie złożonego problemu na podproblemy, które można rozpatrywać i rozwiązywać niezależnie od siebie i od całości,
* Zasada abstrakcji (metoda walki ze złożonością) - budowa abstrakcyjnych struktur i operowanie na nich bez wnikania w ich wewnętrzną strukturę. Eliminacja, ukrycie lub pominięcie mniej istotnych szczegółów rozważanego przedmiotu lub mniej istotnej informacji. Wyodrębnienie cech wspólnych i niezmiennych dla pewnego zbioru bytów i wprowadzenie pojęć lub symboli oznaczających takie cechy,
* Zasada ponownego użycia - wykorzystanie wcześniej wytworzonych schematów, metod, wzorców, bibliotek, komponentów projektu itd.
* Zasada sprzyjania naturalnym ludzkim własnościom (cel nadrzędny obiektowej analizy) - dopasowanie modeli pojęciowych i modeli realizacyjnych systemów do wrodzonych ludzkich własności psychologicznych, instynktów oraz mentalnych mechanizmów percepcji i rozumienia świata.

**Cechy paradygmatu obiektowego:**

* Hermetyzacja

Ukrywanie implementacji, **enkapsulacja**. Zamknięcie pewnego zestawu bytów programistycznych w "kapsułę" o dobrze określonych granicach. Oddzielenie abstrakcyjnej specyfikacji tej kapsuły (obiektu, klasy, moduły etc.) od jej implementacji, ukrycie części informacji zawartej w tej kapsule dla operacji z zewnątrz obiektu. Hermetyzacja jest podstawową techniką abstrakcji tj. ukrycia wszelkich szczegółów danego przedmiotu lub bytu programistycznego, które na danym etapie rozpatrywania (analizy, projektowania, programowania) nie są istotne.

Zapewnia, że obiekt nie może zmieniać stanu wewnętrznego innych obiektów w nieoczekiwany sposób. Każdy typ obiektu prezentuje innym obiektom swój interfejs, które określa dopuszczalne metody współpracy.

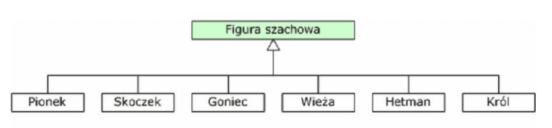
Pewne języki osłabiają to założenie, dopuszczając pewien poziom bezpośredniego (kontrolowanego) dostępu do wnetrza obiektu, ograniczając w ten sposób poziom abstrakcji. Przykładowo w niektórych kompilatorach C++ istnieje możliwość tymczasowego wyłączenia mechanizmu enkapsulacji i otwarcia w ten sposób dostępu do wszystkich pól i metody prywatnych.

* Dziedziczenie

To związek między klasami obiektów określający przekazywanie cech (definicji atrybutów, metod) z nadklasy do jej podklasy. Porządkuje i wspomaga polimorfizm i enkapsulację dzięki umożliwieniu definiowania i tworzenia specjalizowanych obiektów na podstawie bardziej ogólnych. Dla obiektów specjalizowanych nie trzeba redefiniować całej funkcjonalności tylko tę, której nie ma obiekt ogólny.

W typowym przypadku powstają grupy obiektów zwane klasami oraz grupy klas zwane drzewami. Odzwierciedlają one wspolne cechy obiektów.

Istnieje wiele rodzajów dziedziczenia np.: dziedziczenie oparte na klasach, na prototypach, wielokrotne dziedziczenie.



* Abstrakcja

Każdy obiekt w systemie służy jako model abstrakcyjnego "wykonawcy", które może wykonywać pracę, opisywać i zmieniać swój stan oraz komunikować się z innymi obiektami w systemie bez ujawniania, w jaki sposób zaimplementowano dane cechy. Ograniczenie zakresu cech manipulowanych obiektów wyłącznie do cech kluczowych dla algorytmu, a jednocześnie niezależnych od implementacji.

* Polimorfizm

Zdolność obiektów do różnych zachować w zależności od bieżącego kontekstu wykonania programu. Wywołane mogą być różne wersje tej samej funkcji. Pozwala to na rozszerzalność i łatwą modyfikację programu.

**Zastosowania programowania obiektowego:**

* Inżyniera oprogramowania - w języku modelowania UML do analizy oraz modelowania rzeczywistości. Podejście obiektowe ułatwia zrozumienie przez człowieka złożonych bytów informatycznych.
* Języki programowania - na programowaniu obiektowym bazują całe języki programowania takie jak np. język JAVA
* Komunikacja z relacyjnymi bazami danych - ORM (Object Rational Mapping) - sposób odwzorowania obiektów bazodanowych o relacyjnym charakterze poprzez obiektową architekturę systemu informatycznego,
* Biblioteki - programowanie obiektowe ułatwia pisanie reużywalnych bibliotek programistycznych, z których programista ma możliwość korzystania poprzez wystawiony interfejs bez wnikania w szczegóły implementacyjne,
* Komunikacja sieciowa - podejście obiektowe ułatwia komunikację sieciową między programami napisanymi w różnych językach programowania. Przykładem jest ujednolicony format przesyłania danych JSON oparty na podejściu obiektowym.
* Graficzne środowiska tworzenia oprogramowania - RAD (Rapid Application Development) - szybkie tworzenie aplikacji. Jest to ideologia i technologia polegająca na udostępnianiu programiście zestawu gotowych komponentów (obiektów), z których ten ma możliwość tworzenia złożonych programów.
* Współdziałanie systemów heterogenicznych - Przykładem zastosowania jest technologia **CORBA** (Common Object Request Broker Architecture) zapewniająca komunikację między obiektami pracującymi w heterogenicznych (różnorodnych) systemach komputerowych. Obiekty pełniące dowolne funkcje mogą być zaimplementowane w róznych językach programowania, na dowolnej platformie sprzętowej pod kontrolą różnych systemów operacyjnych.

Opis obiektów, a właściwie ich interfejsów znajduję się w specjalnym pliku IDL (Interface Definition Language), które jest kompilowany na kod zajmujący się kontrolą komunikacji w systemie.

Obiekty mają swoje adresy IOR (Interpretable Object Reference) będące kilkusetznakowymi adresami kodującymi wiele informacji m.in. o adresie komputera, programu, nr obiektu, informacje o kolejności zapisu bajów itd.

# Podział programowania obiektowego:

* oparte na klasach - definiowane są klasy, czyli typy zmiennych, a następnie tworzone są obiekty, czyli instancje tych typów
* oparte na prototypach - W tym podejściu nie istnieje pojęcie klasy. Nowe obiekty tworzy się w oparciu o istniejący już obiekt - prototyp, po którym dziedziczone są pola i metody. Dodatkowo można go rozszerzać o nowe. Typ ten spotykany w językach interpretowanych np. JavaScript.

# 2. Mechanizm dziedziczenia, ochrona pól - przykład implementacji

**Dziedziczenie** - to w programowaniu obiektowym mechanizm polegający na współdzieleniu funkcjonalności między klasami. Klasa może dziedziczyć po innej klasie, co oznacza, że oprócz swoich własnych atrybutów oraz zachowań, uzyskuje ("dziedziczy") także te pochodzące z klasy, po której dziedziczy.

Klasa dziedzicząca nazywana jest **klasą pochodną / potomną.**

Klasa, z której następuje dziedziczenie **klasą bazową.**

Klasy pochodne obok swoich własnych pól i metod posiadają również kompletny interfejs klasy bazowej.

W językach programowania z prototypowaniem (np. JavaScript) nie występuje pojęcie klasy, dlatego dziedziczenie zachodzi tam pomiędzy poszczególnymi obiektami.

Każdy obiekt o typie klasy pochodnej jest jednocześnie obiektem o typie klasy bazowej. Innymi słowy w C++ jeżeli mamy dwie klasy Figura i Elipsa, przy czym Elipsa dziedziczy po klasie Figura to wszędzie tam, gdzie spodziewamy się Figury geometrycznej lub wskaźnika do Figury możemy wstawić Elipsę.

W C++ dziedziczenie jest realizowane jako relacja zawierania. Innymi słowy jeśli zadeklarujemy, że klasa B dziedziczy po klasie A, to wówczas wewnętrznie będzie to zaimplementowane w ten sposób, że obiekt klasy B będzie w sobie zawierał obiekt klasy A. Przy czym jest to wewnętrzny mechanizm języka C++.

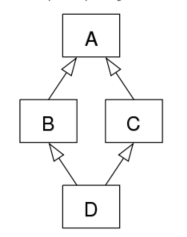
**Kolejność tworzenia obiektów klas bazowych i pochodnych:**

Generalnie podczas alokacji pamięci na obiekt klasy pochodnej najpierw alokowana jest pamięć na obiekt klasy bazowej, a dopiero później na obiekt klasy pochodnej. Dealokacja przebiega odwrotnie.

**Wielodziedziczenie -** nie wspominać, ale dobrze wiedzieć

Mechanizm umożliwiający klasie pochodnej dziedziczenie po więcej niż jednej klasie bazowej. Jest to cecha specyficzna danego języka np. w C++ wielodziedziczenie jest możliwe, a w Javie nie, aczkolwiek z Javą 8 wchodzi możliwość implementowania metod w interfejsach.

Z wielodziedziczeniem związany jest problem diamentu:



Klasa B i C dziedziczą po klasie A, czyli każda z nich ma swoją własną kopię klasy B. Jeśli klasa D dziedziczy zarówno po klasie B jak i C i na obiekcie klasy D spróbujemy wywołać metodę klasy A kompilator nie będzie wiedział, którą kopię metody wywołać (z tę z klasy B, czy z klasy C?).

**Rodzaje dziedziczenia**  - też nie wspominać, ale dobrze wiedzieć

W języku C++ wyróżnia się trzy rodzaje dziedziczenie: prywatne, chronione i publiczne. Jednak dziedziczenie prywatne i chronione są stosowane bardzo rzadko i jest to cecha specyficzna języka C++ (np. w Javie dziedziczenie prywatne i chronione nie istnieje).

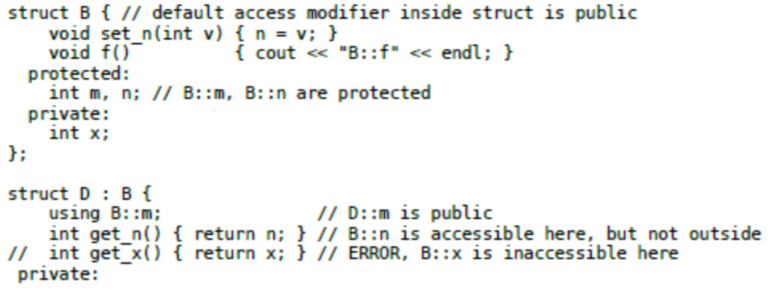
**Ochrona pól:**

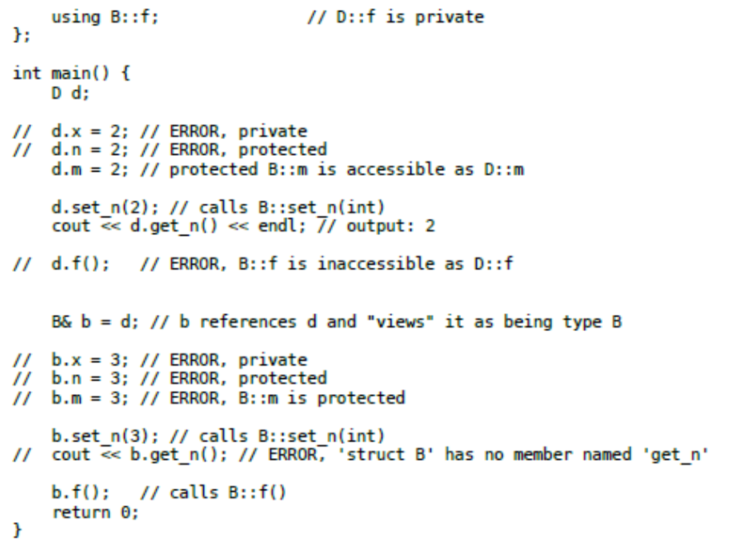
Jest to mechanizm umożliwiający programiście określanie, do których pól i metod będzie można odwołać sie spoza danej klasy. Zazwyczaj jest to realizowane za pomocą tzw. modyfikatorów dostępu:

* private - do pól i metod objętych tym modyfikatorem można odwołać się tylko i wyłącznie z metod tej klasy oraz z funkcji i klas zaprzyjaźnionych z tą klasą,
* protected - podobnie jak private, ale do pól i metod oznaczonych tym modyfikatorem mogą odwoływać się również klasy pochodne,
* public - pola i metody oznaczone tym modyfikatorem są dostępne w całym programie.

W teorii w programowaniu obiektowym w ogóle nie ma mowy o czymś takim jak pola publiczne (publiczne mogą być tylko metody), jednak większość języków (C++, Java) umożliwia programiście deklarację pól publicznych.

Przykład ochrony pól:





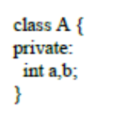
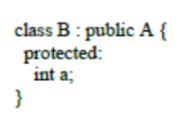
**Dziedziczenie, a zmiany modyfikatorów dostępu.**

Klasa pochodna dziedziczy wszystkie pola i modyfikatory klasy bazowej niezależnie od ich modyfikatorów dostępu, jednak:

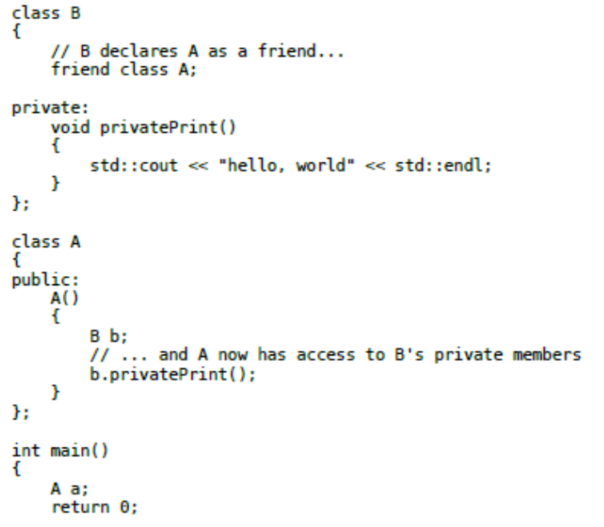
* zmienne i pola zadeklarowane w klasie bazowej jako **publiczne** będą publiczne w klasie pochodnej,
* zmienne i pola zadeklarowane w klasie bazowej jako **chronione**  będą chronione w klasie pochodnej i klasa pochodna będzie miała do nich dostęp,
* zmienne i pola zadeklarowane w klasie bazowej jako **prywatne** zostaną odziedziczone (tak samo jak wszystkie inne), jednak nie będzie do nich bezpośredniego dostępu. Można się do nich dostać tylko przez metody dostępowe odziedziczone z klasy bazowej.

Uwaga Wnuka:

Przyjmijmy, że mamy następujące dwie klasy:

W klasie bazowej i klasie pochodnej mamy zadeklarowaną zmienną **a** o tej samej nazwie, ale o innych modyfikatorach dostępu. Haczyk polega na tym, że ktoś może widząc takie coś pomyśleć, że umieszczona w klasie B deklaracja zmiennej chronionej **a** zmienia modyfikator dostępu zmiennej a odziedziczonej po klasie A. Nic takiego nie ma miejsca.

Dziedziczenie jest realizowane jako zawieranie, w związku z tym w klasie B znajdą się wszystkie zmienne klasy bazowej A wraz z modyfikatorami dostępu zgodnymi z zzasadami dziedziczenia modyfikatorów oraz dodatkowo w klasie B znajdzie się drugie pole o nazwie **a**, ale o modyfikatorze protected. Będziemy więc mieli jedno pole **a** typu int odziedziczone po klasie bazowej, do którego nie będziemy mieli bezpośredniego dostępu oraz drugie pole **a** o modyfikatorze dostępu protected.

**Klasy zaprzyjaźnione** (dodatek dobrze wiedzieć, że coś takiego istnieje)

W tym wypadku obiekt klasy B może wywołac prywatną metodę klasy A.

# 3. Polimorfizm - sposób działania, opisać na przykładzie

**Polimorfizm** jest pojęciem ściśle związanym z programowaniem obiektowym oraz dziedziczeniem. Jest to mechanizm pozwalający na definiowanie jednej metody w wielu postaciach oraz uzależnianie jej działania od typu obiektu, dla którego jest wywoływana.

Polimorfizm możemy określić jako wirtualizację operacji. Jest to możliwość dynamicznego (późnego, realizowanego w fazie wykonania) wiązania nazwy operacji do wielu implementacji (metod) tej operacji w różnych klasach pozostających w relacji dziedziczenia. Wiązaniu towarzyszy mechanizm wyboru konkretnej implementacji. Wybór implementacji zależy od nazwy metody oraz od typu dynamicznego tego obiektu, dla którego została wywołana operacja, a nie od typu zmiennej, wskazującej ten obiekt.

**Co nam daje?**

Dzięki niemu mamy pełną kontrolę nad wykonywanym programem, nie tylko w momencie kompilacji (**wiązanie statyczne**) ale także podczas działania programu (**wiązanie dynamiczne**) – niezależnie od różnych wyborów użytkownika.

Obiekt klasy pochodnej może być wskazywany przez wskaźnik typu klasy bazowej.

**Typem statycznym** obiektu wskazywanego przez wskaźnik jest typ tego wskaźnik, a **typem dynamicznym** obiektu wskazywanego przez wskaźnik jest typ na jaki dany wskaźnik wskazuje.

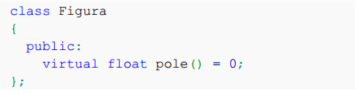
Mechanizm polimorfizmu jest możliwy dzięki metodom wirtualnym.

**Metoda wirtualna** jest to funkcja składowa klasy poprzedzona słowem kluczowym **virtual**, której sposób wywołania zależy od typu dynamicznego wskaźnika, a nie od typu statycznego.

**Metoda wirtualna** to taka metoda, której ciało może zostać przykryte w klasach dziedziczących po klasie bazowej. W ten sposób możemy zdefiniować zupełnie inne zachowania klas dziedziczących po klasie bazowej.

Specyficznym przykładem metody wirtualnej jest metoda czysto wirtualna, która w ogóle nie posiada ciała. Klasa posiadająca taką metodę staje się klasą abstrakcyjną. Oznacza to tyle, iż nie jest możliwe stworzenie obiektu tej klasy. Klasa taka służy jedynie temu, aby zdefiniować pewnego rodzaju interfejs i jest przeznaczona jedynie po to, by po niej dziedziczyć.

W przykładzie poniżej o ile mogą istnieć konkretne figury będące kwadratami, kołami itp. to nie powinien istnieć żaden obiekt klasy Figura ponieważ jest to jedynie abstrakcyjny byt. Natomiast dziedziczenie po tej klasie i jej rozszerzanie powoduje, że stworzymy już konkretną figurę geometryczną. Metodę czysto wirtualną w języku C++ deklaruje się tak:



Taka deklaracja metody wirtualnej uniemożliwia stworzenie jakiegokolwiek obiektu klasy *Figura* oraz zmusza do określenia metody *float pole()* na jednym z poziomów z dziedziczenia. Nie jest możliwe pominięcie takiej implementacji.

Metody wirtualne:

* Definiowane przy wykorzystaniu słowa kluczowego **virtual** (przy implementacji słowa **virtual** nie używamy),
* Przy wywołaniu metody wirtualnej zostanie przeszukana tablica funkcji wirtualny przynależąca do danego obiektu i wyszukana wersja najbliższa w hierarchii dziedziczenia,
* Metody wirtualne w danej hierarchii powinny mieć pełną zgodność. Nie może się różnić liczba parametrów metody. Wartość zwracana może się różnić, ale tylko jeśli w klasie pochodnej metoda zwraca klasę dziedziczącą po typie wartości zwracanej przez metodę klasy bazowej.
* Funkcja zaczyna zaczynać się jak wirtualna w momencie pierwszego pojawienia się słowa **virtual**,
* Zachowanie wirtualne może (ale nie musi) skończyć się po pierwsyzm wsytąpieniu funkcji bez **virtual** w hierarchii dziedziczenia,
* Rodzaj dziedziczenia nie wpływa na zachowanie sie funkcji wirtualnych (zmienia się jedynie ich widoczność),
* Metody statycznie nie mogą być wirtualne,
* W przypadku argumentów domyślnych wykorzystana będzie wartość odpowiadająca wersji funkcji według typu wskaźnika, a nie dynamicznego typu (związane z określaniem wartości domyślnych na etapie kompilacji),
* Konstruktor nie może być wirtualny,
* Destruktor powinien być wirtualny ze względu na niebezpieczeństwo wycieków pamięci jeśli w klasie została zadeklarowana jakakolwiek metoda wirtualna. Deklaracja destruktora powinna znajdować się w klasie bazowej.

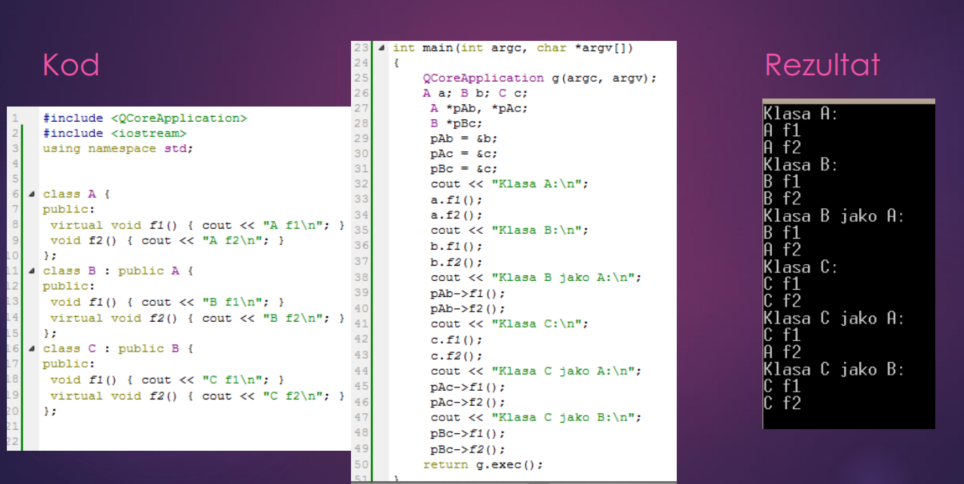
Polimorfizm uzyskujemy dzięki:

* Niejawnemu przekształceniu typów wskaźników do klas pochodnych na wskaźniki do klas typu podstawowego,
* Mechanizmowi funkcji wirtualnych,
* Operacji rzutowania *dynamic\_cast* oraz operatorom *typeid*

**Polimorfizm kosztuje** - Klasy **polimorficzne** zajmują więcej miejsca w pamięci, ponieważ kompilator automatycznie dodaje do nich wskaźnik vptr wskazujący na tablicę vtab. Dla każdej klasy musi istnieć osobny wskaźnik i osobna tablica. Tablica jest generowana automatycznie i zawiera wskaźniki do funkcji, wygenerowane przez kompilator.

W języku JAVA wszystkie metody są wirtualne.

**Przykład polimorfizmu C++**



Klasa B jako A:

* metoda *f1()* zostanie związana dynamicznie z typem na, który wskaźnik pAb będzie wskazywał (typ klasy B).
* metoda *f2()* nie jest zadeklarowana jako wirtualna zatem zostanie związana statycznie z typem wskaźnika w fazie kompilacji - stąd wywołanie metody *f2()* na wskaźniku pAb wywołuje metodę z klasy A

Klasa C jako A taka sama sytuacja.

Klasa C jako B:

* metoda *f1()* była zadeklarowana jako wirtualna w klasie A, zatem zostanie związana dynamiczna i wywołana implementacja z klasy C,
* metoda *f2()* została zadeklarowana jako wirtualna w klasie B, więc również zostanie związana dynamicznie i wywołana implementacja z klasy C

Jawne wywołanie funkcji bazowej jest możliwe poprzez poprzedzenie metody dwukropkiem i nazwą klasy.

# 4. Pojęcie zbioru rozmytego, definicja i interpretacja funkcji przynależności.

**Zbiorem rozmytym A** w pewnej niepustej przestrzeni **X**, co zapisujemy nazywamy zbiór par:

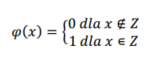
gdzie:

μA(x) jest **funkcją przynależności** zbioru rozmytego A. Funkcja ta każdemu elementowi przyporządkowuje jego stopień przynależności do zbioru rozmytego A, przy czym można wyróżnić 3 przypadki:

* μA(x) = 1 - oznacza to pełną przynależność do zbioru rozmytego A, tzn. ,
* μA(x) = 0 - oznacza to brak przynależności elementu x do zbioru rozmytego A, tzn.
* 0< μA(x) < 1 - oznacza to częściową przynależnośc elementu x do zbioru rozmytego A.

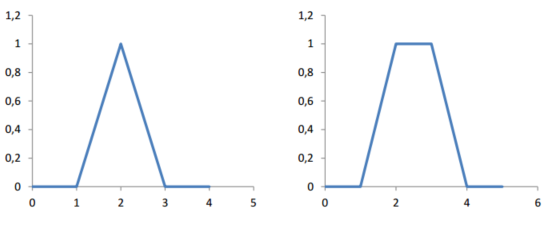
**Funkcja przynależności** jest uogólnieniem funkcji charakterystycznej określonym na zbiorach rozmytych.

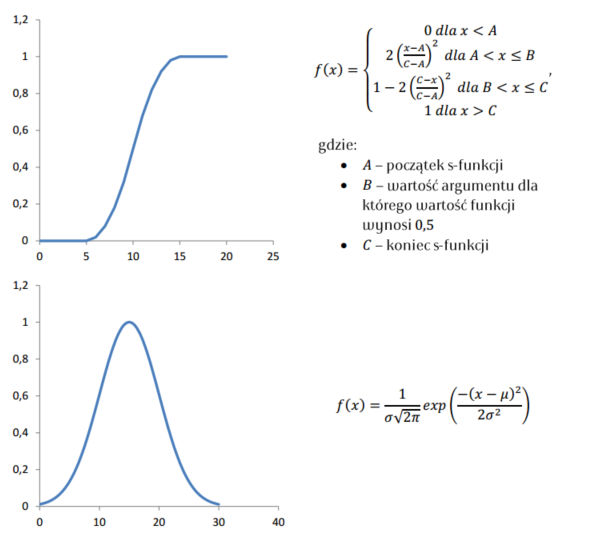
Funkcja charakterystyczna zbioru Z to funkcja φ(x), która elementom przestrzeni X przyporządkowuje wartości 0 i 1:

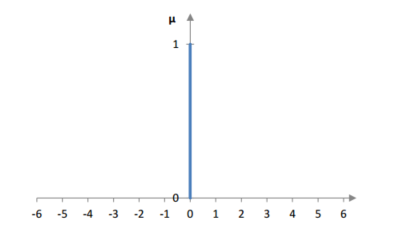


Funkcja charakterystyczna określa, czy dany element x przestrzeni X należy, czy nie należy do zbioru Z, a funkcja przynależności określa dodatkowo stopień przynależności do tego zbioru.

W teorii zbiorów rozmytych istnieją różne funkcje przynależności. Najczęściej stosowane są funkcje trójkątne, trapezowe, singleton, s-funkcje oraz funkcje Gaussa (dwie ostatnie ze względu na to, że są różniczkowalne).



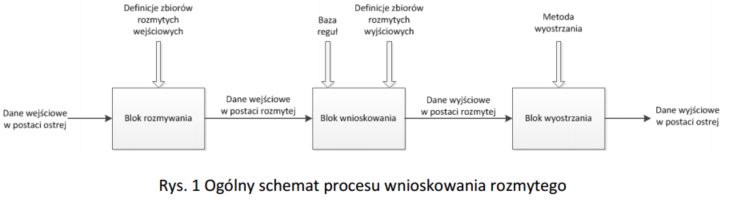




# 5. Elementy składowe wnioskowania rozmytego.

**Wnioskowanie rozmyte**, podobnie jak wnioskowanie w logice klasycznej polega na ocenie prawdziwości zdania logicznego (**wniosku** lub inaczej **konkluzji**) na podstawie prawdziwości innych zdań logicznych (**przesłanek)**. Różnica polega na tym, że w logice klasycznej zdanie może być prawdziwe lub fałszywe, a w logice rozmytej mówimy o stopniu prawdziwości zdania. Zdanie dotyczy zazwyczaj przynależności danej liczbowej do określonego zbioru.

Wnioskowanie rozmyte odbywa się według poniższego schematu:

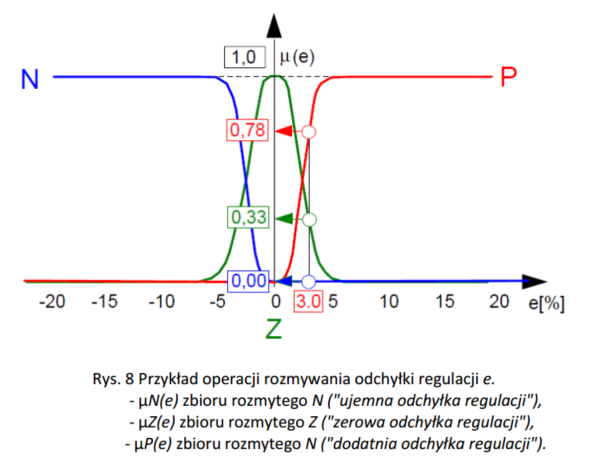


**Blok rozmywania**

Rozmywaniem nazywamy przekształcenie wartości ostrych wielkości wejściowej (np. modelu) do wartości rozmytych. Każdej wartości ostrej w określonej przestrzeni rozważań (czyli określonym zbiorze) zostają przyporządkowane zbiory wartości funkcji przynależności do określonych zbiorów rozmytych. **Zbiorem rozmytym** nazywamy zbiór uporządkowanych par - wartość ostra x, wartość funkcji przynależności μ(x). Funkcja przynależności określa stopień przynależności elementu do danego zbioru.

W ramach rozmywania w pierwszej kolejności należy nazwać wejścia. Następnie należy zaprojektować zbiory rozmyte. Odbywa się to poprzez przyporządkowanie wartości określonym w poprzednim kroku zmiennym słownym (lingwistycznym). Końcową fazą rozmywania wejścia jest określenie zbioru wartości funkcji przynależności dla każdej wartości nazwy lingwistycznej na podstawie wartości ostrej wejścia oraz stworzonych zbiorów rozmytych.

Przykład - rozmywanie odchyłki regulacji



**Blok wnioskowania**

**Wnioskowaniem** nazywamy wyznaczanie wyjścia o charakterze rozmytym na podstawie nieostrego wejścia oraz bazy reguł między wejściem, a wyjściem.

**Baza reguł** jest zbiorem implikacji, mówiących o przynależności zmiennej wyjściowej do danego zbioru rozmytego w zależności od przynależności zmiennych wejściowych do odpowiednich zbiorów rozmytych. Reguły mają postać IF ... THEN ...

Mówimy, że baza reguł jest zupełna jeśli definiuje wszystkie relacje wejście-wyjście. Liczba reguł zależy od liczby wejść, wartości lingwistycznych im przypisanych i liczby wyjść. Przesłanki (reguły) mogą być złożone. Wówczas poszczególne elementy przesłanki są łączone spójnikami AND lub OR, ponadto używa się w zapisie reguł operatora zaprzeczenia NOT. W anaalogiczny sposób następnik może być również złożony.

Oprócz bazy reguł do przeprowadzenia procesu wnioskowania niezbędne są **funkcje przynależności zmiennych wyjściowych**  do ich zbiorów rozmytych.

Wnioskowanie może być oparte na pojedynczej regule bądź ich złożeniu.

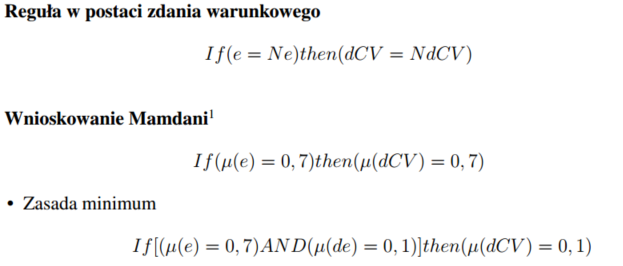
Wnioskowanie na pojedynczej regule odbywa się zgodnie z implikacją **Mamdaniego**. Stopień spełnienia następnika reguły nie może przekroczyć stopnia spełnienia przesłanki. Stopień spełnienia przesłanki złożonej jest wyznaczany zgodnie z zasadą minimum (dla AND) lub maksimum (dla OR). Zasada minimum/maksimum mówi o tym, że stopień spełnienia w tym wypadku przesłanki jest równy minimalnej/maksymalnej wartości funkcji przynależności jednej ze składowych przesłanki. Dla operatora NOT stopień spełnienia przesłanki w postaci NOT(x = A) jest równy (1-μA).

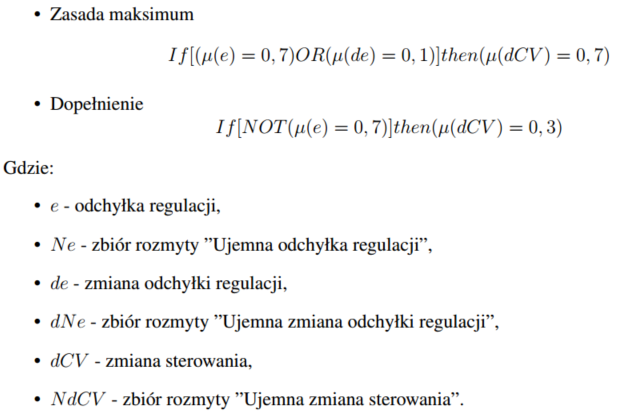
Stopień w jakim spełniony jest wniosek możemy nazwać **poziomem zapłonu reguły**.

W trakcie wnioskowania na podstawie jednej reguły ważne jest odnalezienie reguły, która została spełniona w największym stopniu. Wnioskowanie odbywające się w ten sposób jest proste i nie wymaga dużych nakładów obliczeniowych.

Możliwe jest także wnioskowanie skalowane, polegające na wyznaczeniu stopnia spełnienia przesłanki zgodnie z wyżej podanymi zasadami i przeskalowaniu zbioru rozmytego następnika o wyznaczony poziom (mnożenie każdej wartości funkcji przynależności wniosku przez stopień spełnienia przesłanki).

Z kolei wnioskowanie na zbiorze reguł odbywa się na zasadzie utworzenia sumy mnogościowej wniosków cząstkowych, które są wyznaczane na podstawie pojedynczych reguł.

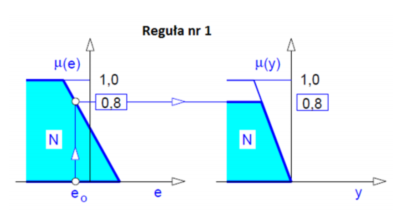


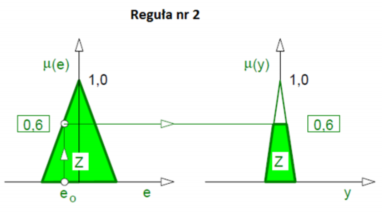


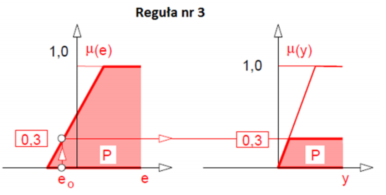
**Blok wyostrzania**

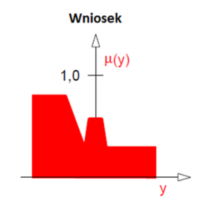
**Wyostrzaniem** nazywamy przekształcenie zbioru rozmytego do wartości ostrej wielkości wyjściowej.

Przykład dla jednego wejścia rozmytego i trzech funkcji przynależności:



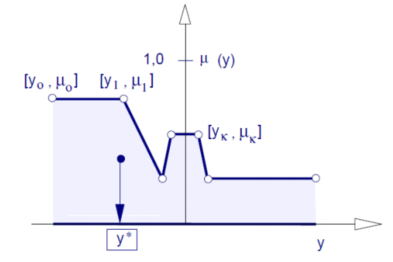






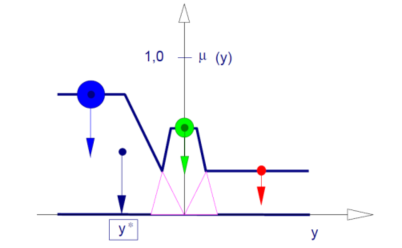
Odwzorowanie wzoru rozmytego w wartość ostrą nie jest jednoznaczne, tzn. istnieje wiele metod wyostrzania:

* **metody obszarowe**
  + metoda środka ciężkości (COG)

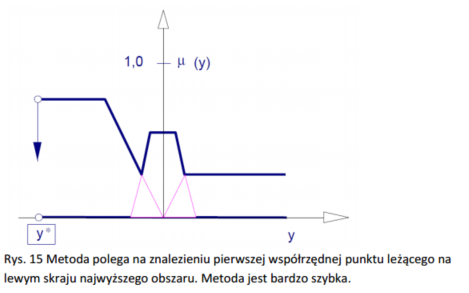


Polega na znalezieniu pierwszej współrzędnej środka ciężkości obszaru pod wykresem wniosku rozmytego.

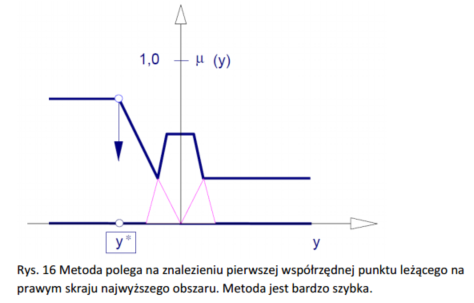
* + metoda środka sum (COS)
  + metoda środka największego obszaru (COLA)
* metody wysokościowe
  + metoda wysokości (HM)

Wyjście jest średnią ważoną wysokości zbiorów rozmytych. Metoda prostrza od metod obszarowych.

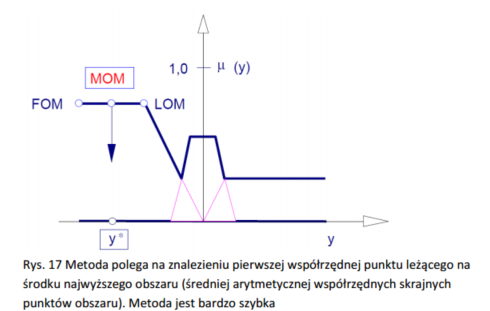
* + metoda wysokości "pierwszy z największych" (FOM)



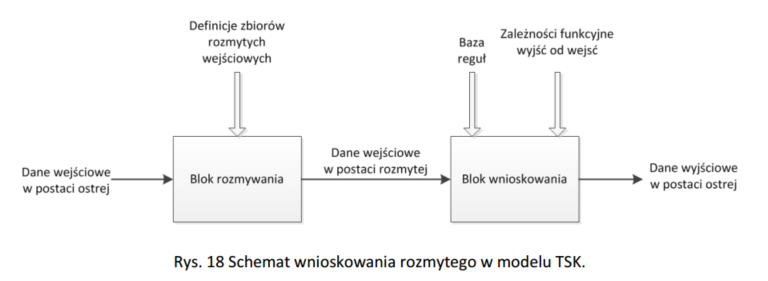
* + metoda wysokości "ostatni z największych" (LOM)



* + metoda wysokości "środkowy z największych" (MOM)



**Model TSK (Takagi-Sugeno-Kang'a)**



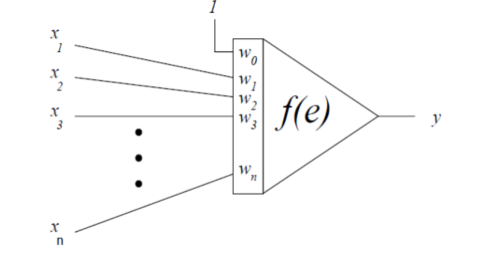
W tym modelu wniosek (reguła) jest w postaci ostrej. Nie ma zatem bloku wyostrzania. Reguły jednak też są w postaci IF...THEN...

Wnioskiem z pojedynczej reguły jest zależność funkcyjna zmiennej wyjściowej (z konkluzji) od zmiennych wejściowych (z przesłanek). Do przeprowadzenia procesu wnioskowania jest zatem niezbędna znajomość zależności funkcyjnych zmiennej wyjściowej od wejściowej. Wnioskiem ogólnym z kilku reguł jest średnia ważona konkluzji z pojedynczych reguł, gdziaktye wagami są poziomy aktywacji pojedynczych reguł.

# 6. Typy sieci neuronowych i ich zastosowanie.

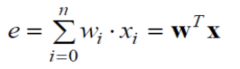
**Sztuczne sieci neuronowe** to struktury, składające się z prostych jednostek obliczeniowych (sztucznych neuronów), przetwarzających dane, komunikujących się między sobą i pracujących równolegle. Powstały na gruncie wiedzy o działaniu systemu nerwowego istot żywych i stanowią próbę wykorzystania zjawisk zachodzących w systemach nerwowych przy poszukiwaniu nowych rozwiązań technologicznych.

**Sztuczny neuron** można rozpatrywać jako specyficzny przetwornik sygnałów.



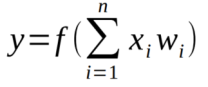
**Podstawowe elementy składowe neuronu:**

* n wejść neuronu wraz z wagami wi (wektor wag w oraz wektor sygnałów wejściowych x),
* pobudzenie **e** neuronu:



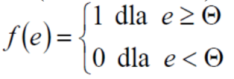
* funkcja aktywacji (przejścia),
* jeden sygnał wyjściowy **y**,

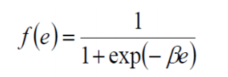
Zatem formuła opisująca działanie neuronu:



Jak wynika z powyższego wzoru działanie neuronu jest bardzo proste. Sygnały wejściowe x1, x2 ... xn zostają ponożone przez odpowiednie wagi w1, w2 ... wn, a następnie otrzymane wartości są sumowane. W wyniku tych operacji powstaje sygnał poddawany działaniu funkcji aktywacji **f**. O właściwościach neuronu decydują wagi oraz rodzaj i parametry funkcji aktywacji. Wagi są dobierane w procesie uczenia sieci.

**Podstawowe typy funkcji aktywacji:**

* liniowa
* nieliniowe - ciągłe, nieciągłe, unipolarne i bipolarne,
* funkcja skoku jednostkowego, progowa (McCulloch i Pitts),



* funkcja sigmoidalna:
* funkcja tangens hiperboliczny

**Uczenie sieci**

* Z nauczycielem

Uczenie z nauczycielem polaga na tym, że sieci podaje się przykłady poprawnego działania, które powinna ona porem naśladować w swoim bieżącym działaniu. Przykład należy rozumieć w ten sposób, że nauczyciel podaje konkretne sygnały wejściowe i wyjściowe, pokazując jaka jest wymagana odpowiedź sieci dla pewnej konfiguracji danych wejściowych. Mamy do czynienia z parąwartości - przykładowym sygnałem wejściowym i pożądanym (oczekiwanym) wyjściem, czyli wymaganą odpowiedzią sieci na ten sygnał wejściowy. Zbiór przykładów zgromadzonych w celu ich wykorzystaniu w procesie uczenia sieci nazywa się zwykle ciągiem uczącym. Zatem w typowym procesie uczenia sieć otrzymuje od nauczyciela ciąg uczący i na jego podstawie uczy się prawidłowego działania, stosując jedną z wielu znanych dziś strategii uczenia.

* Bez nauczyciela

Obok opisanego wyżej schematu uczenia z nauczycielem występuje też szereg metod tak zwanego uczenia bez nauczyciela (albosamouczenia sieci). Metody te polegają na podawaniu na wejście sieci wyłącznie szeregu przykładowych danych wejściowych, bez podawania jakiejkolwiek informacji dotyczącej pożądanych czy chociażby tylko oczekiwanych sygnałów wyjściowych. Odpowiednio zaprojektowana sieć neuronowa potrafi wykorzystać same tylko obserwacje wejściowych sygnałów i zbudować na ich podstawie sensowny algorytm swojego działania - najczęściej polegający na tym, że automatycznie wykrywane są klasy powtarzających się  sygnałów wejściowych i sieć uczy się (spontanicznie, bez jawnego nauczania) rozpoznawać te typowe wzorce sygnałów.

**Można wyróżnić kilka rodzajów sieci neuronowych:**

* sieci jednokierunkowe,
* sieci rekurencyjne (Hopfielda),
* sieci samoorganizujące się (Kohena),
* sieci radialne (RBF).

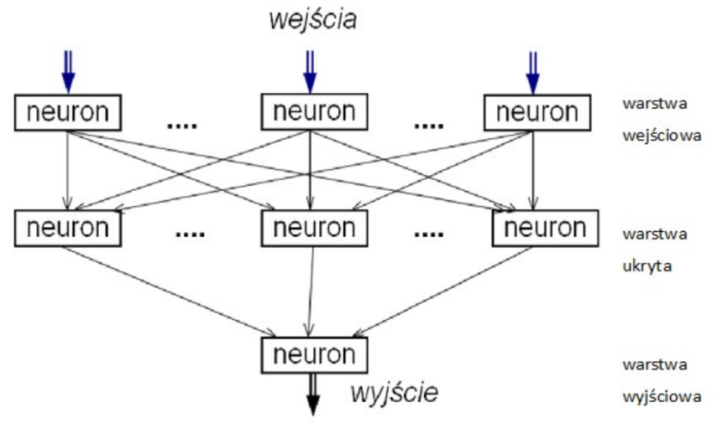
**1. Sieci jednokierunkowe**

Przepływ sygnałów odbywa się w jednym kierunku - od wejścia do wyjścia. Zwykle neurony ułożone są w warstwach, a powiązania dotyczą tylko neuronów w sąsiednich warstwach.

Wyróżnia się w niej następujące warstwy:

* warstwę wejściową,
* warstwy ukryte,
* warstwę wyjściową.

**Perceptron wielowarstwowy**

****

W warstwie wejściowej następuje normalizacja sygnałów, a w warstwie wyjściowej wyliczana jest wartość sygnału wyjściowego z sieci (neuron sigmoidalny lub o liniowej funkcji aktywacji). Warstwy ukryte tworzą najczęściej neurony o sigmoidalnej funkcji aktywacji. Każdy neuron kolejnej warstwy otrzymuje sygnały wyjściowe wszystkich neuronów warstwy wcześniejszej i przekazuje swój sygnał do wszystkich neuronów kolejnej warstwy. W obrębie tej samej warstwy neurony nie mogą się łączyć ze sobą. Ilość warstw ukrytych i ilość neuronów w warstwie ukrytej zależy od charakteru zadania. Do modelowania procesów przemysłowych zazwyczaj stosuje się perceptrony z jedna warstwą ukrytą.

Największym zainteresowaniem cieszy się sieć jednokierunkowa, wielowarstwowa o neuronach typu sigmoidalnego, zwana perceptronem wielowarstwowym.

Uczenie sieci polega na modyfikacji wag. Podczas uczenia dąży się do minimalizacji odpowiednio zdefiniowanej funkcji celu, której argumentem jest wektor szukanych wag wejść neuronów.

Do uczenia sieci najczęściej stosuje się metodę propagacji wstecznej. Algorytm propagacji wstecznej (nazwa tego algorytmu wynika z kolejności sygnałów obliczania sygnałów błędu, która przebiega w kierunku odwrotnym niż przechodzenie sygnałów przez sieć, to znaczy od warstwy wyjściowej poprzez warstwy ukryte w kierunku warstwy wejściowej) określa strategię doboru wag przy wykorzystaniu gradientowych metod optymalizacji. Podstawę algorytmu stanowi funkcja celu, definiowana jako suma kwadratów różnić między aktualnymi wartościami sygnałów wyjściowych, a wartościami zadanymi.

Najskuteczniejsze są gradientowe metody uczenia sieci, których podstawą działania jest znajomość gradientu funkcji celu. Opierają się na rozwinięciu w szereg Taylora funkcji celu w najbliższym sąsiedztwie znanego aktualnie rozwiązania.

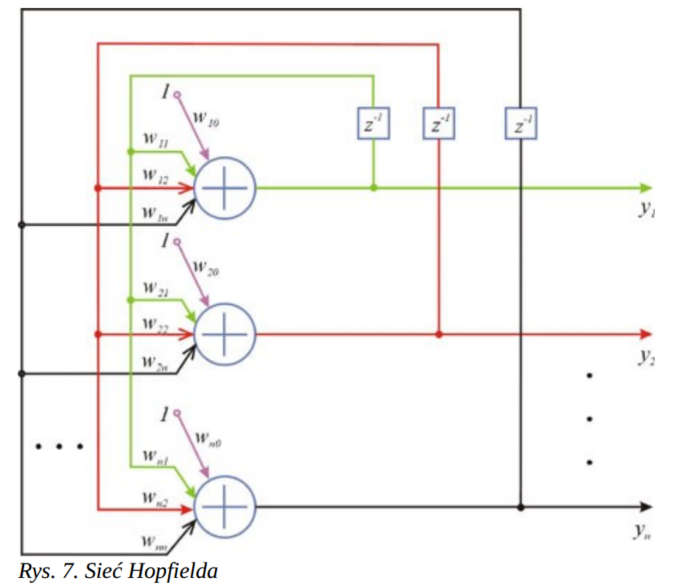
Zastosowania sieci jednokierunkowych:

* budowanie prostych układów logicznych (np. XOR),
* aproksymacja dowolnej nieliniowej, ale statycznej funkcji,
* aproksymacja zależności wielkości wyjściowych od wejściowych, gdy zależność ta jest trudna do opisu matematycznego na podstawie zależności fizycznych, ale mamy doświadczalne wartości wejść i wyjść do nauki sieci (diagnostyka procesów, walidacja czujników)

**2. Sieci rekurencyjne (Hopfielda)**

Różnią się od sieci jednokierunkowych występowaniem sprzężenia zwrotnego między warstwami wyjściowymi lub ukrytymi i wejściowymi. Podstawową cechą wyróżniającą te sieci są zależności dynamiczne na każdym etapie działania. Zmiana stanu jednego neuronu przenosi się w skutek masowego sprzężenia zwrotnego na całą sieć, wywołując stan przejściowy, kończący się określonym stanem ustalonym, ogólnie innym niż poprzedni.

Jest to sieć jednowarstwowa o regularnej budowie, składająca się z wielu neuronów połączonych każdy z każdym. Wagi sieci są symetryczne tzn. . Neurony mają funkcję aktywacji typu signum, przyjmującą wartości ze zbioru {-1,1). Podczas uczenia sieć Hopfielda modyfikuje swoje wagi w zależności od wektora uczącego. W trybie odtworzeniowym wagi nie ulegają modyfikacjom, natomiast sygnał wejściowy pobudza sieć, która poprzez sprzężenie zwrotne wielokrotnie przyjmuje na swoje wejście sygnał wyjściowy, aż do ustabilizowania się odpowiedzi.



Uczenie sieci rekurencyjne odbywa się według **reguły Hebba**, zgodnie z którą wagi są modyfikowane według zależności:



gdzie: **N** - ilość wyjść, **M** - ilość wektorów uczących, **i** - nr wektora uczącego

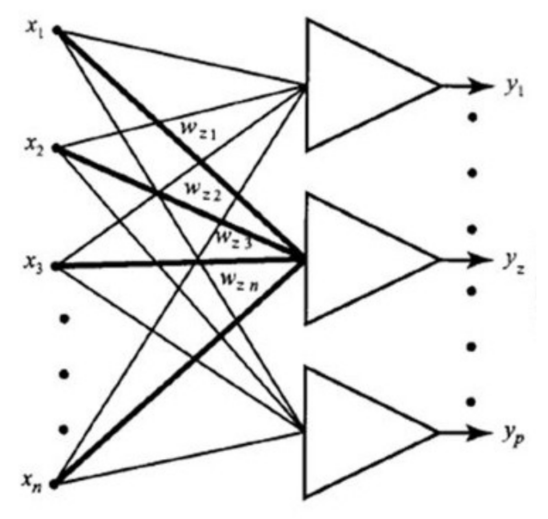
Zastosowanie:

* jako pamięć asocjacyjna, tzn. ma za zadanie odtworzenie pewnego wzorca na podstawie zaszumionego sygnału wejściowego. Stąd jest stosowana np. do klasyfikacji i rozpoznawania obrazów.
* optymalizacja przy użyciu specjalizowanej struktury obwodu (sieć realizująca programowanie liniowe i kwadratowe, sieć rozwiązująca problem komiwojażera, podział grafu na 2 części - zapewnia minimalną liczbę łączeń między częściami),
* przetwarzanie sygnałów - przetworniki A/C. transformacja Fouriera, przetwarzani i dekompozycja sygnałów

**3. Sieci samoorganizujące się (Kohena)**

Sieć, której uczenie odbywa się "bez nauczyciela". Ciąg uczący składa się jedynie z wartości wejściowych.

Sygnał wejściowy trafia na wejście wszystkich neuronów. Tutaj następuje wyznaczenia miary podobieństwa (odległości w sensie dobranej metryki) sygnału wejściowego **x** od wszystkich wektorów wag. Następnie wyłania się zwycięzcę, czyli neuron o wagach o najmniejszej odległości od wektora wejściowego i modyfikuje się jego wagi oraz wagi jego sąsiadów. Wielkość korekty zależy od odległości wektora wejściowego od wektora wag zwycięskiego neuronu. Ten algorytm uczenia nazywa się *Winner Takes All*. Jest to przykład uczenia bez nauczyciela.

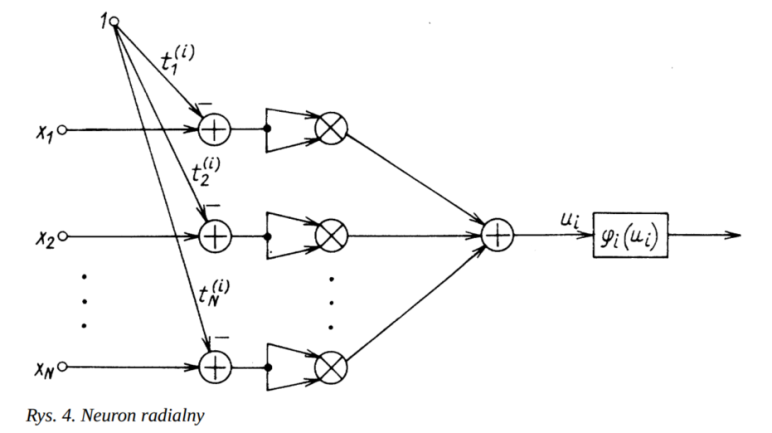


Zastosowanie:

* grupowanie danych,
* kompresja danych - odwzorowanie topologii danych wejściowych poprzez dużo mniejszą liczbę wag neuronów tworzących sieć,
* przetwarzanie mowy
* prognozowanie (np. obciążeń systemu elektroenergetycznego),

**4. Sieci radialne**

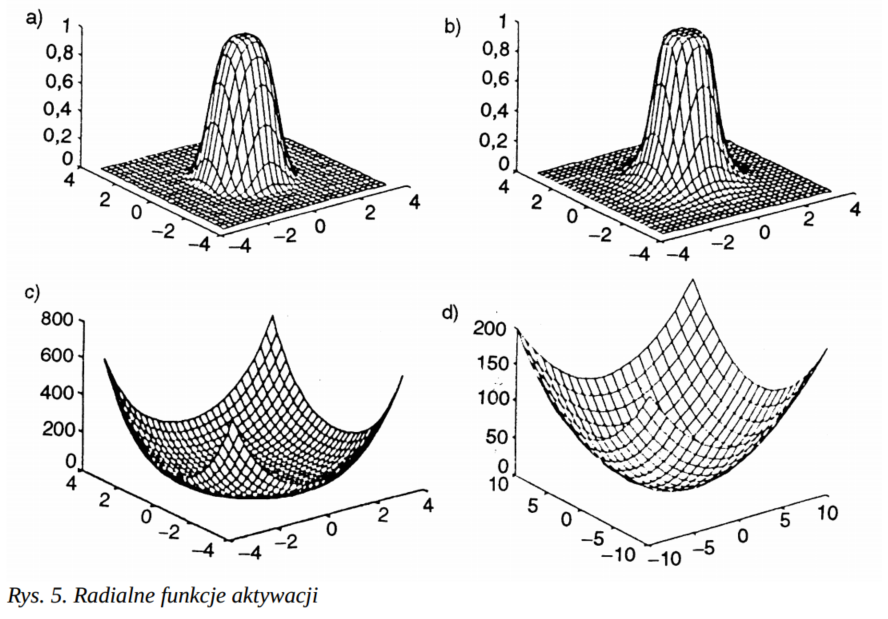
Zawierają neurony o radialnych funkcjach aktywacji.



Funkcje aktywacji realizują odwzorowanie:

gdzie: - to norma euklidesowa.

Funkcja φ() jest nazywana radialną funkcją bazową. Jej wartość zmienia się radialnie wokół środka t. Oznacza to, że argument funkcji aktywacji jest różnica między wektorem wyznaczonym przez wartości sygnałów wejściowych i wektorem współczynników t.



W porównaniu do sieci perceptronowych, sieci radialne zawierają znacznie więcej neuronow. W realizacji programowej działają wolniej niż sieci perceptronowe. Stosują inny sposób przetwarzania danych, przez co następuje skrócenie procesu uczenia się. Mają z góry ustalona architekturę składającą się z trzech warstw: wejściowej, ukrytej i wyjściowej. Nauron warstwy wyjściowej realizuje operację ważone sumy sygnałów. W warstwie ukrytej można stosować różne funkcje bazowe.

Uczenie składa się z dwóch kroków:

* Dobór położenia oraz kształtu funkcji bazowych wybranymi metodami (dobór losowy, samoorganizacja, metoda wstecznej propagacji błędu).
* Dobór wag neuronu wyjściowego.

Zastosowanie:

Uczenie się tych sieci bywa bardzo wolne, ale działanie po procesie uczenia bardzo szybkie. Wykorzystywane są przykładowo do:

* aproksymacja funkcji nieliniowych (jak w jednokierunkowych),
* przewidywanie np. przewidywanie trendów ekonomicznych na podstawie danych z długiego okresu czasu,
* monitoring np. monitoring dźwięków wydawanych przez silniki samolotów i pociągów,
* kontrola procesów chemicznych,
* kompresja danych,.

# 7. Cechy charakterystyczne i zastosowania systemów czasu rzeczywistego

**System czasu rzeczywistego** (ang. real-time system) to urządzenie techniczne, którego wynik i efekt działania jest zależny od chwili wypracowania tego wyniku. Istnieje wiele różnych definicji naukowych takiego systemu. Ich wspólną cechą jest zwrócenie uwagi na równoległość w czasie zmian w środowisku oraz obliczeń realizowanych na podstawie stanu środowiska. Z tego wyściugu dwóch stanów: zewnętrznego i wewnętrznego wynikają kryteria ograniczające czas wypracowania wyniku.

Systemy czasu rzeczywistego najczęściej są tworzone w oparciu o komputery, ale nie jest to konieczne. Można tym pojęciem określić np. pneumatyczny regulator.

**System operacyjny czasu rzeczywistego** (ang. real-time operating system - RTOS) to komputerowy system operacyjny, który został opracowany tak, aby spełnić wymagania narzucone na czas wykonywania zadanych operacji. Systemy takie stosuje się jako elementy komputerowych systemów sterowania pracujących w reżimie czasu rzeczywistego.

Ogólnie można przyjąć założenie, że zadaniem systemu operacyjnego czasu rzeczywistego oraz oprogramowania pracującego pod jego kontrolą i całego sterownika komputerowego jest wypracowanie odpowiedzi (np. sygnałów sterujących kontrolowanym obiektem) na skutek wystąpienia pewnych zdarzeń (zmiana sygnałów z czuników sterownika). Biorąc to pod uwagę podstawowym wymogiem dla systemów operacyjnych czasu rzeczywistego jest określenie najgorszego najdłuższego czasu, po jakim urządzenie komputerowe wypracuje odpowiedź po wystepianiu zdarzenia. Ze względu na to kryterium systemy operacyjne czasu rzeczywistego dzielą się na:

* **Twarde** - takie, dla których znany jest najdłuższy czas odpowiedzi oraz wiadomo, ze nie zostanie on przekroczony,
* **Miękkie** - takie, które starają się odpowiedzieć najszybciej jak to możliwe, ale nie wiadomo jest jaki może być najgorszy czas odpowiedzi,

**Najczęstsze schematy działania RTOS'ów:**

* Wywołane zdarzeniami (Event-driven) - przełączają wykonywanie procesu, gdy nadejdzie proces o wyższym priorytecie - szeregowanie priorytetami. Głównie dla systemów miekkich.
* Dzielące czas (Time-sharing) - przełączające zadania w równych odstępach czasowych kontrolowanych przez zegar procesora i algorytm szeregowania,

W systemach dzielących czas kluczową kwestią jest opracowanie (wybranie) odpowiedniego algorytmu szeregowania (scheduling) oraz sposobu podziału czasu procesora. Należy określić, któremu z procesów należy przydzielić procesor oraz na jak długi czas, aby wszystkie wykonywane procesy spełniały zdefiniowane dla nich ograniczenia czasowe.

**Algorytm szeregowania** (scheduler - planista) - to algorytm rozwiązujący jedno z najważniejszych zagadnień informatyki - jak rozdzielić czas procesora i dostęp do innych zasobów między zadania o te zasoby konkurujące.

Najczęściej algorytm szeregowania jest implementowany jako część wielozadaniowego systemu operacyjnego. Oprócz systemów operacyjnych dotyczy w szczególności także serwerów bazodanowych.

**Używane najczęściej algorytmy szregowania:**

* **FIFO** - często stosowany algorytm, jeden z prostszych w realizacji. Daje dobre efekty w systemach ogólnego przeznaczenia. Zadanie wykonuje się aż nie zostanie wywłaszczone przez siebie lub inne zadanie o wyższym priorytecie.
* **Planowanie rotacyjne** (round-robin) - Znane również jako algorytm karuzelowy. Każde z zadań otrzymuje kwant czasu, po spożytkowaniu którego zostaje wywłaszczone i ustawione na końcu kolejki.
* **Planowanie sporadyczne** - zadania otrzymują tak zwany "budżet czasu". Ten algorytm pomaga pogodzić wykluczające się regułu dotyczące szregowania zadań okresowych i nieokresowych. Wciąż nie jest implementowany przez wiele systemów, jednak znalazł się w standardzie POSIX.

**Najbardziej znane systemy operacyjne czasu rzeczywistego:**

* LynxOS,
* OSE,
* QNX twardy,
* RTLinux twardy,
* VxWordks twardy,
* Windows CE - system miękki,
* MacOS - system miękki

**Systemy czasu rzeczywistego znajdują zastosowane:**

* W przemyśle do nadzorowania procesów technologicznych,
* Do nadzorowania eksperymentów naukowych,
* W urządzeniach powszechnego użytku jak sterowniki układów ABS i ESP czy wtrysku paliwa do silników samochodowych, bądź też w urządzeniach gospodarstwa domowego,
* W medycynie, w lotnictwie, zastosowaniach wojskowych i komicznych