UML- Unified Modeling Language
Ujednolicony Język Modelowania

- UML jest standardowym językiem do specyfikacji, wizualizacji, budowy i dokumentowania wszystkich artefaktów (wytworów) dowolnego systemu.
- UML jest językiem o szerokim zakresie zastosowań.
- UML nadaje się do opisu systemów programowych i nieprogramowych (tzw. systemów biznesowych) w różnych dziedzinach i branżach, np. w produkcji, bankowości, handlu elektronicznym itd.

UML- Unified Modeling Language
Ujednolicony Język Modelowania

- UML może być stosowany w całym procesie tworzenia systemu softwerowego, od gromadzenia wymogów, aż po implementację systemu.
- Spełnia zatem jako narzędzie oczekiwania inżynierii oprogramowania.
- Język ten obsługuje wielu producentów narzędzi, które sa standardami branżowymi; nie jest to zastrzeżony lub zamknięty język modelowania.

Historia języka UML

Lata 80-90:

- Różnorodne metodologie, techniki , różnorodne notacje, symbole, oznaczenia.

Lata 90-95:

- Wyróżniają się 3 metodologie:
  - Metoda Grady'ego Boocha Booch '93
  - Metoda Jamesa Rumbaugha OMT-2
  - Metoda Ivara Jacobsona OODE

Historia języka UML

1. Metoda Grady'ego Boocha Booch '93 (powstała z Booch '91) kładła nacisk na projektowanie i tworzenie systemów oprogramowania. Słaba w analizie.
2. Metoda Jamesa Rumbaugha OMT-2 (ang. Object Modeling Technique — technika modelowania obiektów, powstała z OMT-1) kładła nacisk na analize systemów oprogramowania. Słaba w projektowaniu.
3. Metoda Ivara Jacobsona OODE (ang. Object-Oriented Software Engineering — obiektowa technika programowania) kładła nacisk na modelowanie biznesowe i analize wymagań.

Historia języka UML

Lata 95-97:

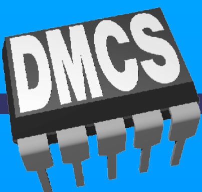
- James Rumbaugh, a po nim Ivar Jacobson dołączyli do Grady'ego Boocha w firmie Rational Software Corporation, aby połączyć swoje metody podejścia. Wyłonił się język UML 1.0.

Po 97:

- Standaryzacja, korekty, prace nad wersją 2.0

Najbardziej aktualna wersja specyfikacji UML jest dostępna w serwisie OMG http://www.omg.org.

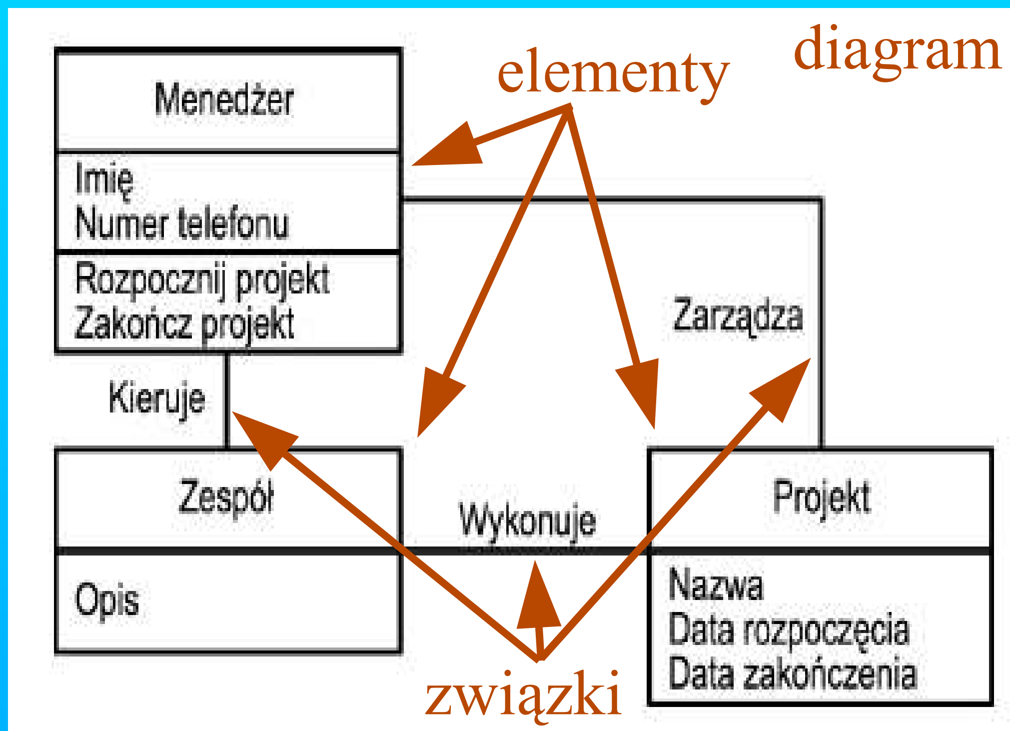
Object Management Group (OMG) — organizacja tworząca standardy powszechnie przyjmowane w przemyśle

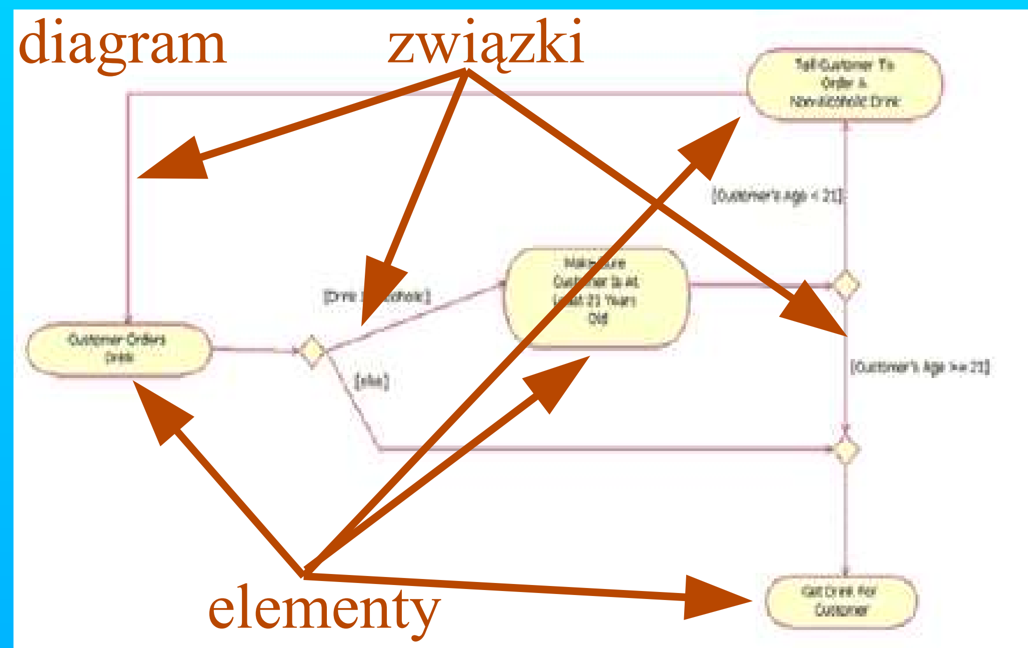


Na co położony zostanie nacisk w nauce języka UML

- Paradygmat obiektowy, ponieważ tworzy podstawy języka UML.
- Modelowanie strukturalne i behawioralne, ponieważ pozwalają zrozumieć wymogi stawiane systemowi i jego architekturę.

UML językiem programowania.





UML jest po prostu językiem wizualnym, służącym do modelowania i opisywania systemów za pomocą bloków konstrukcyjnych: elementów, związków między nimi i diagramów.

Schemat modelu systemu

Diagramy w UML.

Diagramy – schemat przedstawiający zbiór bytów. Najczęściej jest grafem, w którym wierzchołkami są elementy, a krawędziami związki.

Diagramy struktury
– statyczne aspekty systemu

- diagramy klas
- diagramy obiektów
- diagramy komponentów
- diagramy wdrożenia

Diagramy zachowania
– dynamiczne aspekty systemu

- diagramy przypadków użycia
- diagramy interakcji:
  - diagramy przebiegu
  - diagramy kooperacji
- diagramy stanów
- diagramy czynności

DMCS

Elementy w UML.1/5.

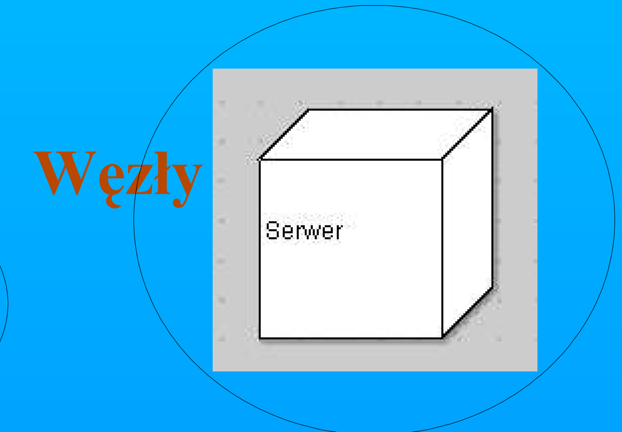
Strukturalne – wyrażone rzeczownikami.
Najbardziej statyczne elementy modelu.
Reprezentują składniki pojęciowe albo fizyczne.

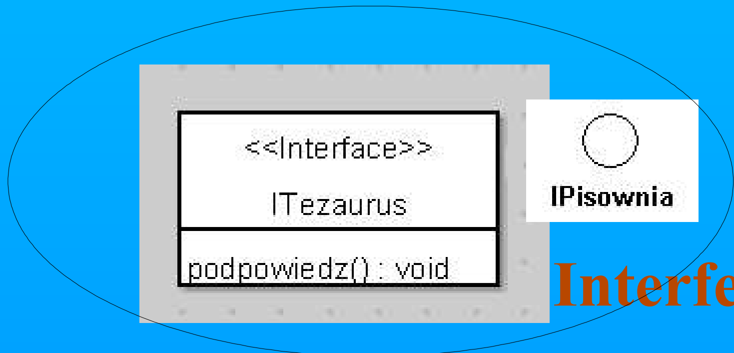
Przypadki użycia

Kooperacje

Klasy

Węzły





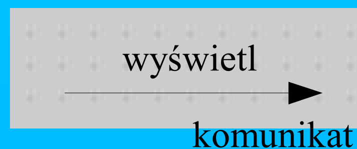
Interfejsy

Komponenty

Elementy w UML. 2/5.

Czynnościowe – dynamiczna część modelu w UML. Wyrażone czasownikami. Opisują zachowanie w czasie i w przestrzeni. Powiązane z elementami strukturalnymi.

Interakcja



komunikat

Zachowanie polegające na wymianiekomunikatów między obiektami.

- komunikaty
- ciągi akcji w odpowiedzi na komunikaty
- połączenia między obiektami

Maszyna stanowa

stan

Określa ciąg stanów jakie obiekt lub interakcja może przyjąć.

- stany
- przejścia między stanami
- zdarzenia powodujące przejścia
- czynności – odpowiedzi na zdarzenia

Elementy w UML. 3,4/5.

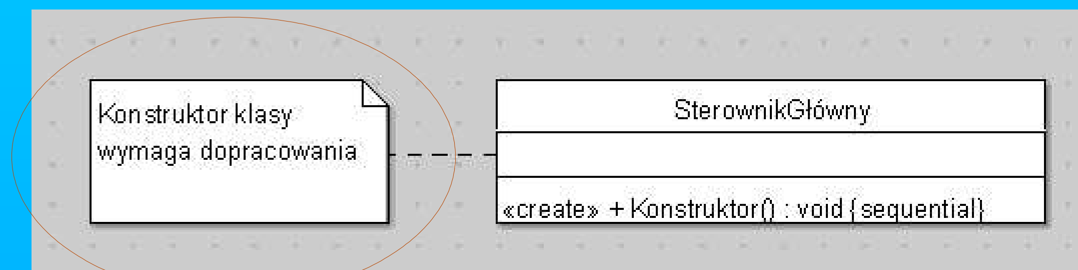
Grupujące – bloki na które może być dany model rozłożony. Rola organizacyjna.

Pakiety

- **Modele**
- **Pakiety**
- **Zręby**
- **Podsystemy** - rodzaje pakietów

- elementy strukturalne
- elementy czynnościowe
- inne pakiety

Komentujące – objaśnienia pisane w celu uwypuklenia lub zaznaczenia dowolnych składników systemu.



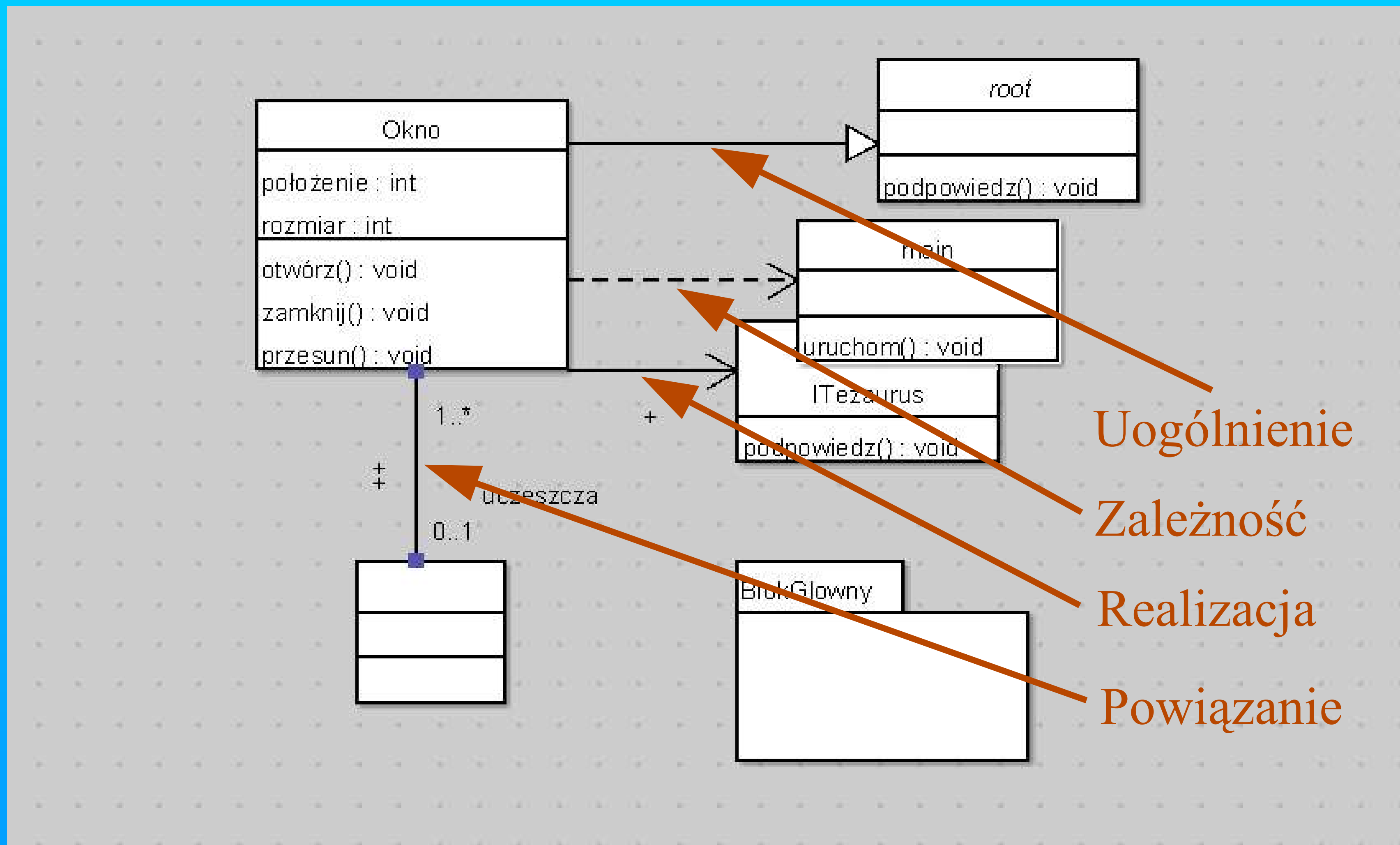
Notatka

- **Notatka**
- **Wymagania**

DMCS

Elementy w UML.5/5.

Związki – służą do łączenia elementów. Używane do budowy poprawnych modeli.

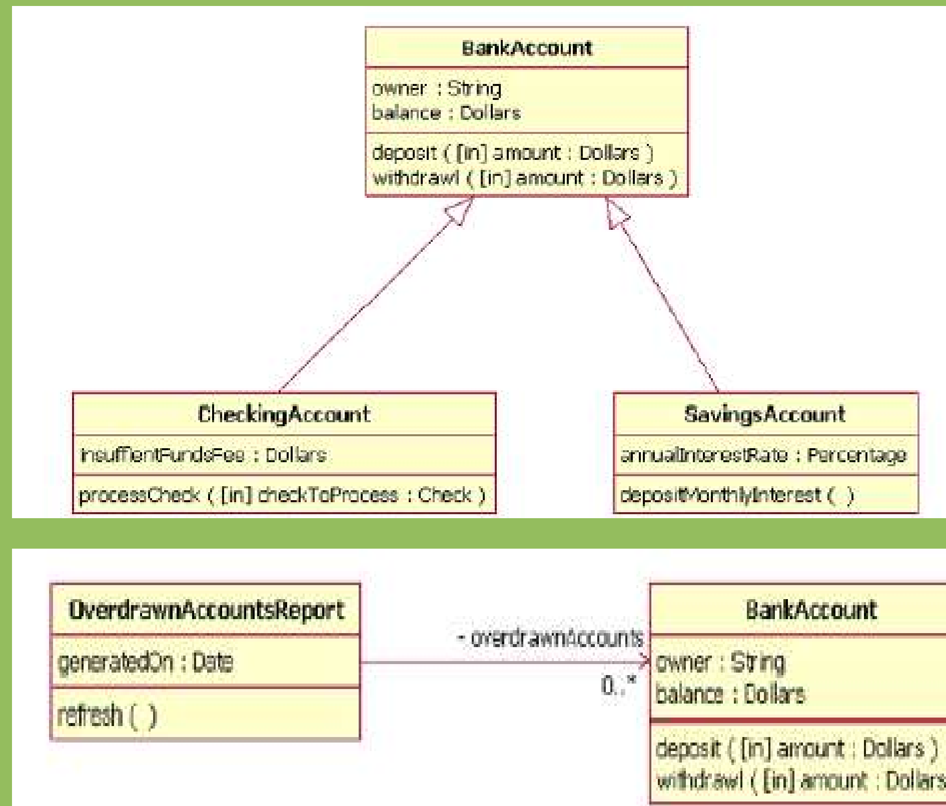


Właściwości.

Diagramy oraz elementy posiadają pewne właściwości. Właściwości są zbiorem cech charakterystycznych dla danego elementu. Mogą się składać z kilku podzbiorów. Właściwości definiowane są przez programistę. Posiadają one z kolei predefiniowane cechy, jak np. typ, widoczność itp.

Diagram klas

Class Diagram

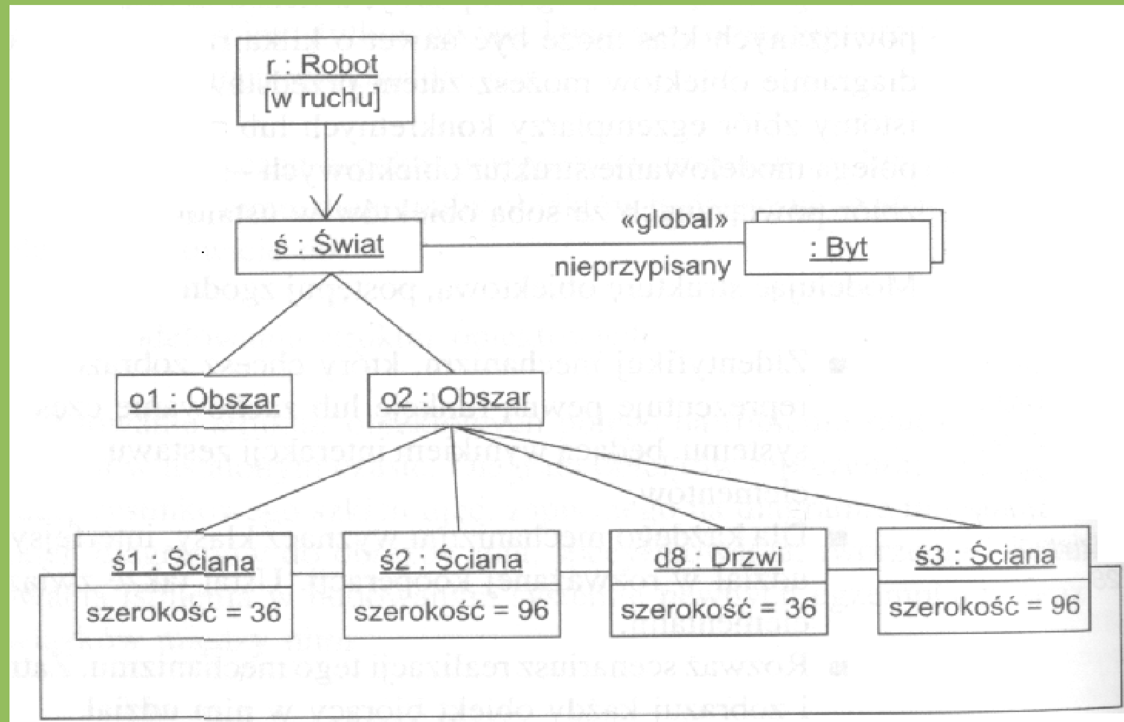


Klasy, obiekty, kooperacje, interfejsy, związki między nimi.

Diagram obiektów

Object Diagram

Wyobraża statyczny rzut pewnych egzemplarzy elementów występujących w diagramie klas.

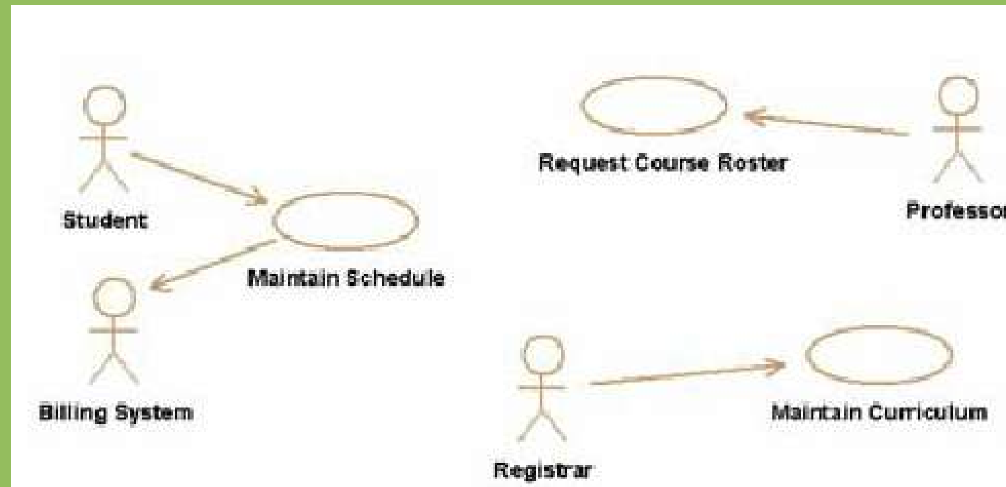


Obiekty, związki między nimi.

Diagram przypadków użycia

Use Case Diagram

Statyczne aspekty perspektywy przypadków użycia. Wyznaczanie i modelowanie systemu.



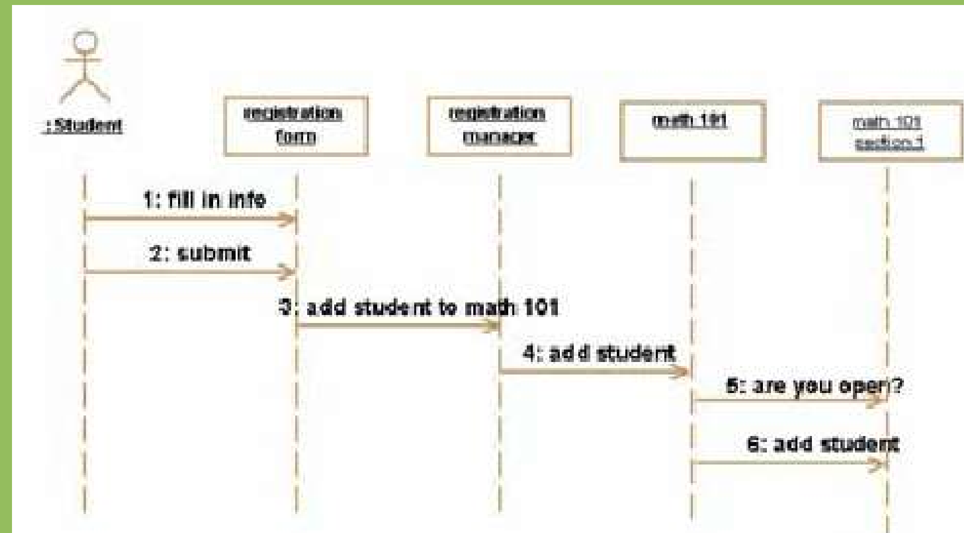
Przypadki użycia, aktorzy, zależności, uogólnienia, powiązania.

DMCS

Diagram przebiegu

Sequence Diagram

Obrazuje kolejność wysyłania komunikatów w czasie.

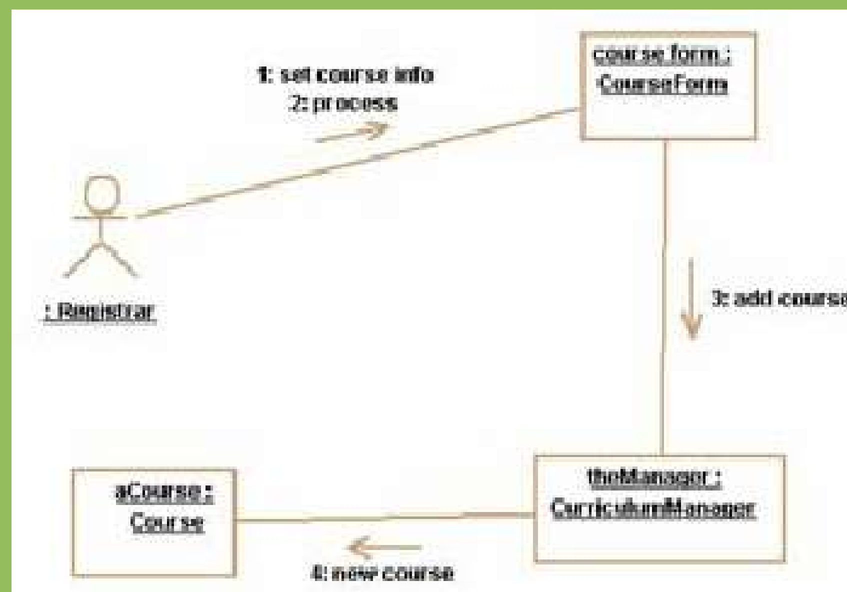


Obiekty, wiązania, komunikaty.

Diagram kooperacji

Collaboration Diagram

Strkturalna organizacja obiektów, które wysyłają i odbierają komunikaty.

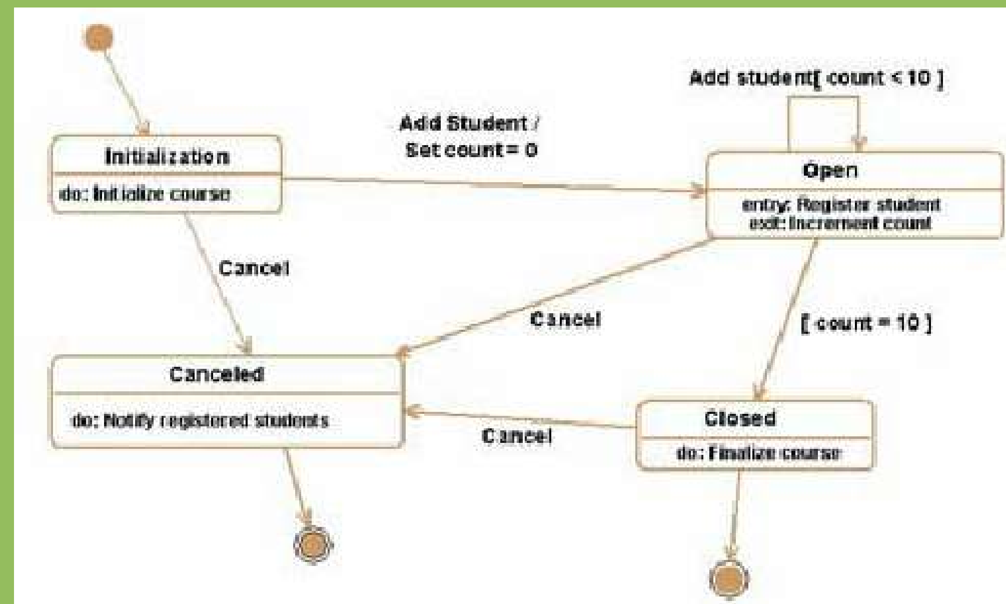


Obiekty, wiązania, komunikaty.

Diagram stanów

Statechart Diagram

Przedstawia maszynę stanową.
Opisuje reakcje obiektów na ciągi zdarzeń.
Modelowanie zachowania interfejsów, klas i kooperacji.
Projektowanie systemów interakcyjnych.

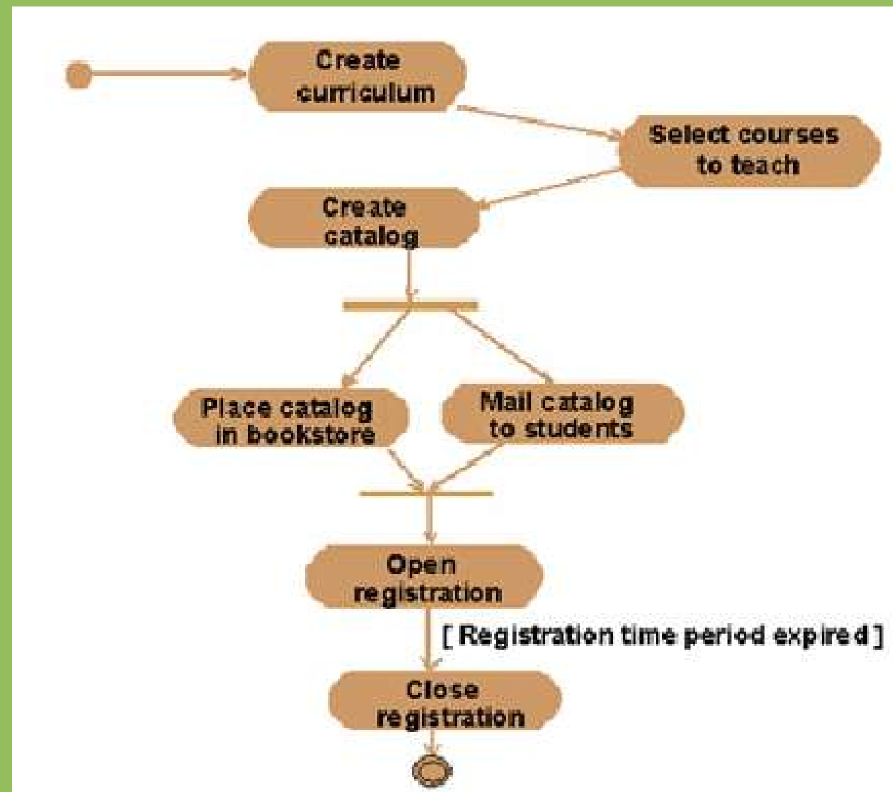


Stany zwykłe i złożone, przejścia ze zdarzeniami i akcjami.

Diagram czynności

Activity Diagram

Szczególny przypadek diagramu stanów. Obrazuje przepływ sterowania od czynności do czynności między obiektami. Modelowanie funkcji systemu.

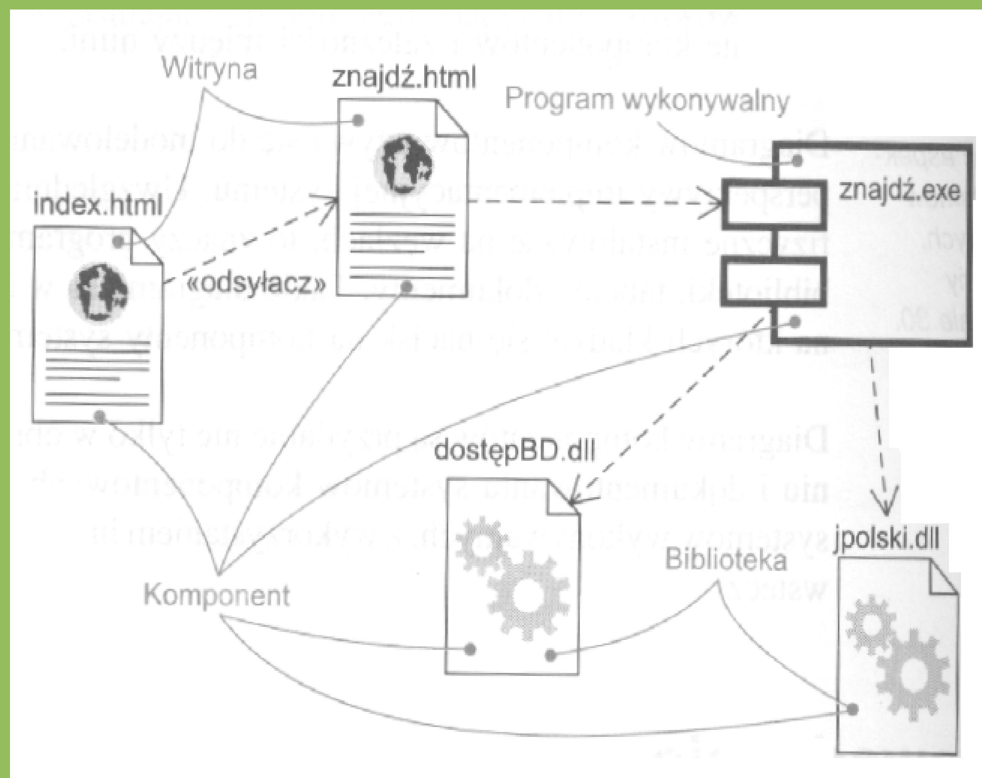


Stany akcji, stany czynności, przejścia, obiekty.

Diagram komponentów

Component Diagram

Obrazuje uporządkowanie komponentów i zależności między nimi. Ściśle wiąże się z diagramem klas, ponieważ zwykle każdemu komponentowi przyporządkowane są pewne klasy, interfejsy i kooperacje.

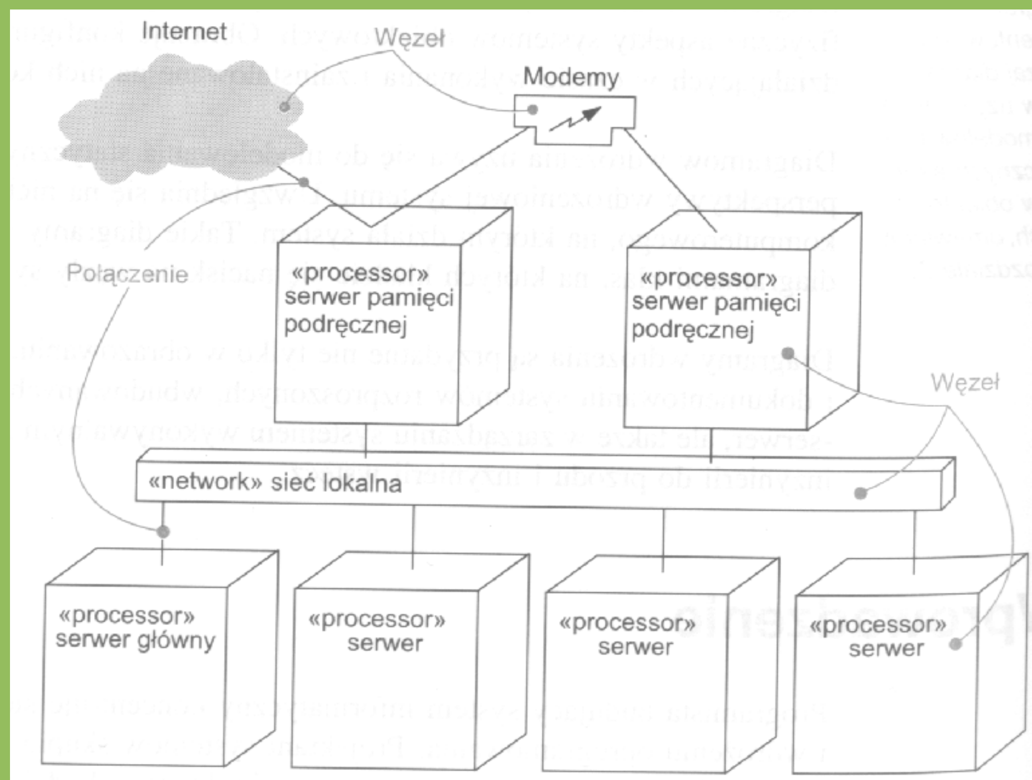


Komponenty, interfejsy, zależności, uogólnienia, powiązania i realizacje.

Diagram wdrożenia

Deployment Diagram

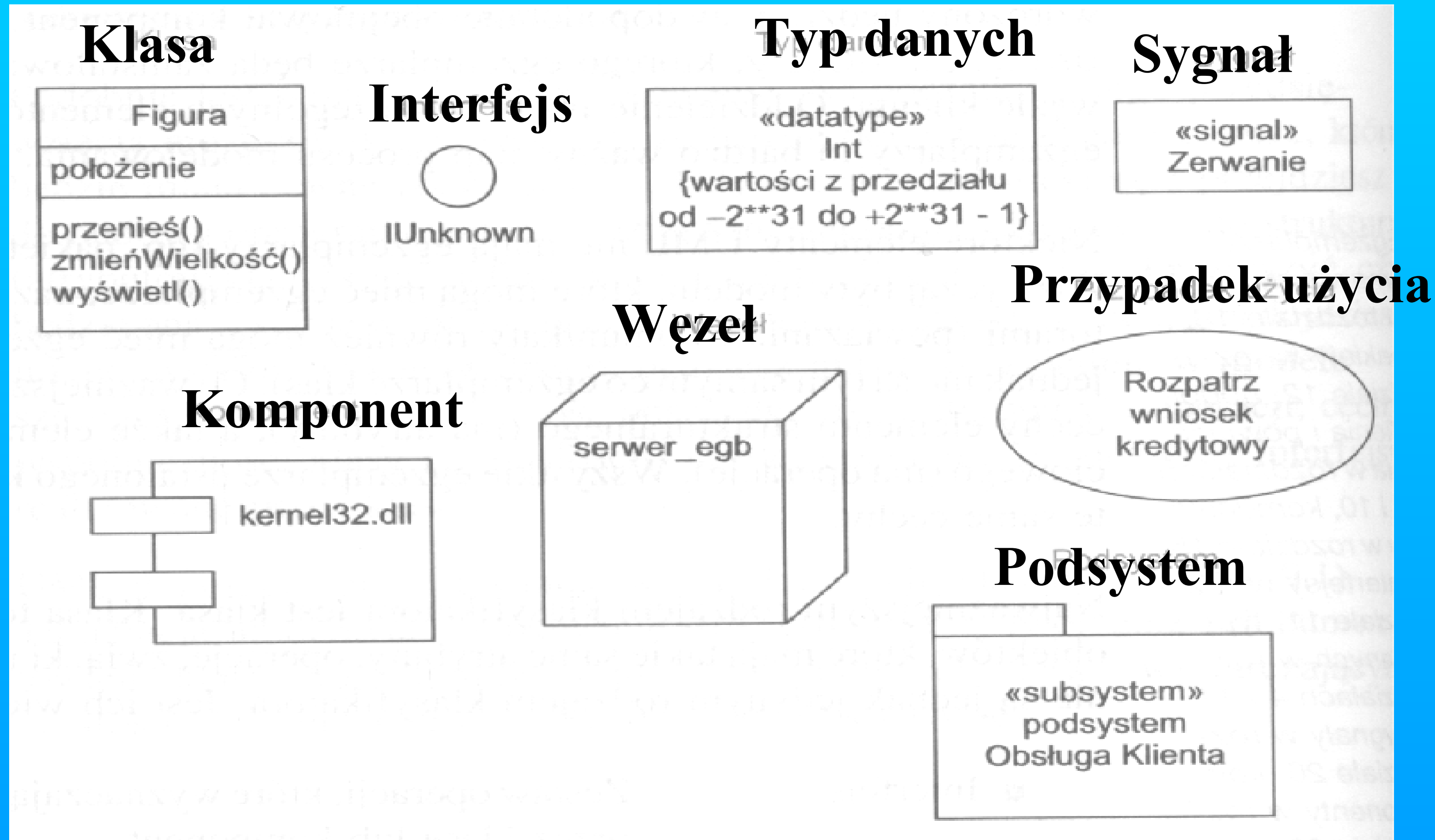
Obrazuje konfigurację poszczególnych węzłów w czasie wykonania i zainstalowane na nich komponenty.
Wiąże się z diagramem komponentów, ponieważ zwykle każdy węzeł zawiera co najmniej jeden komponent.



Węzły, zależności, powiązania.

Czym są klasyfikatory?

Klasyfikatory to byty modelu mogące posiadać egzemplarze. Przedstawiane są za pomocą bloków konstrukcyjnych.



DMCS

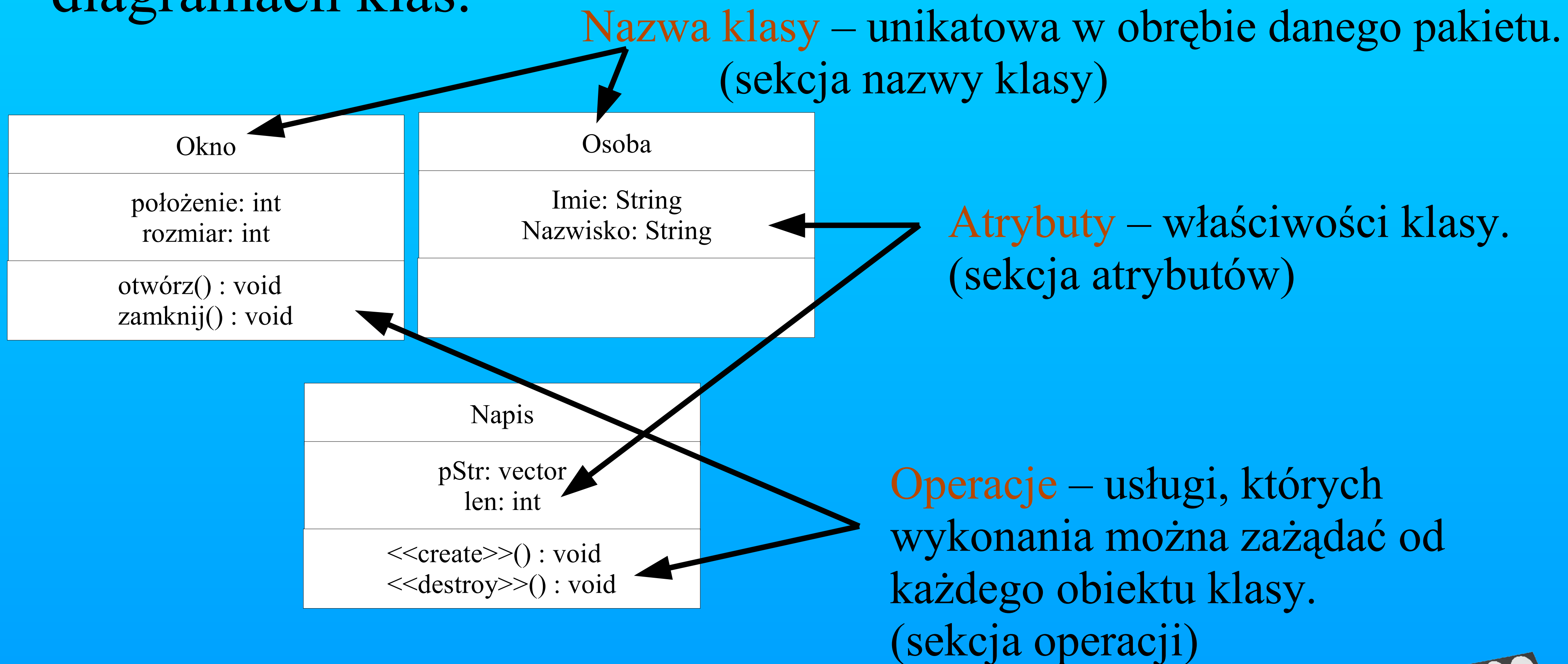
Podstawowe elementy występujące w diagramie klas

Diagramy klas używane są do modelowania statycznych aspektów perspektywy projektowej. Wiąże się z tym w głównej mierze modelowanie słownictwa systemu. Diagramy klas stanowią bazę wyjściową dla diagramów komponentów i wdrożenia.
Diagramy klas przedstawiają zbiory klas, interfejsów, kooperacji oraz związki między nimi.

Na kolejnych slajdach przedstawione zostaną elementy, które mogą wystąpić w diagramach klas.

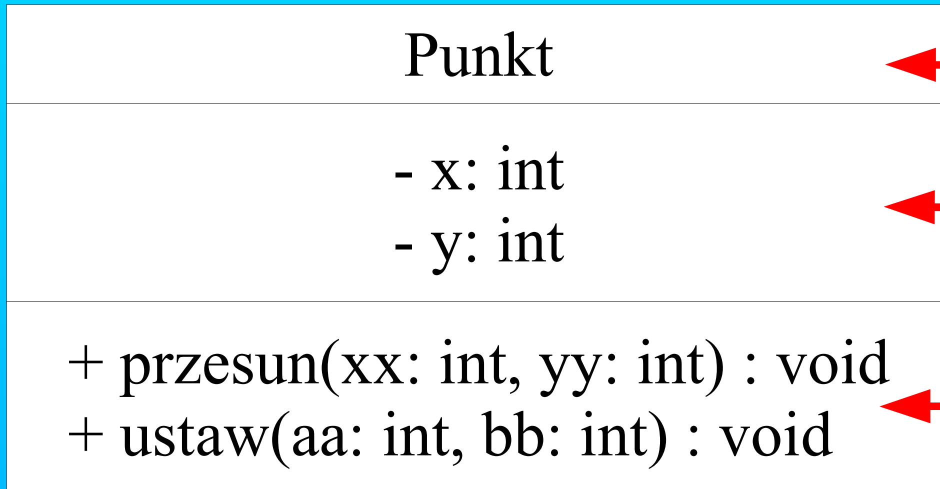
Czym są klasy?

Klasa to opis zbioru obiektów, które mają takie same atrybuty, związki i znaczenie. Klasy definiuje się w diagramach klas.



Właściwości klas

UML



Nazwa klasy

Atrybuty

Operacje

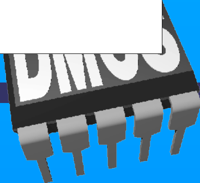
C++

```
class Punkt
{
    private:
        int x;
        int y;
    public:
        void przesun(int xx, int yy);
        void ustaw(int aa, int bb);
};
```

Nazwa klasy

Atrybuty

Deklaracje metod



Obiekty

UML

P1 : Punkt

Nazwa Obiektu

C++

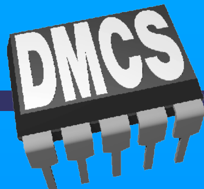
Punkt2 : Punkt

```
Punkt p1, punkt2, p;

Punkt PktCentralny;
```

Pkt : Punkt

: Punkt

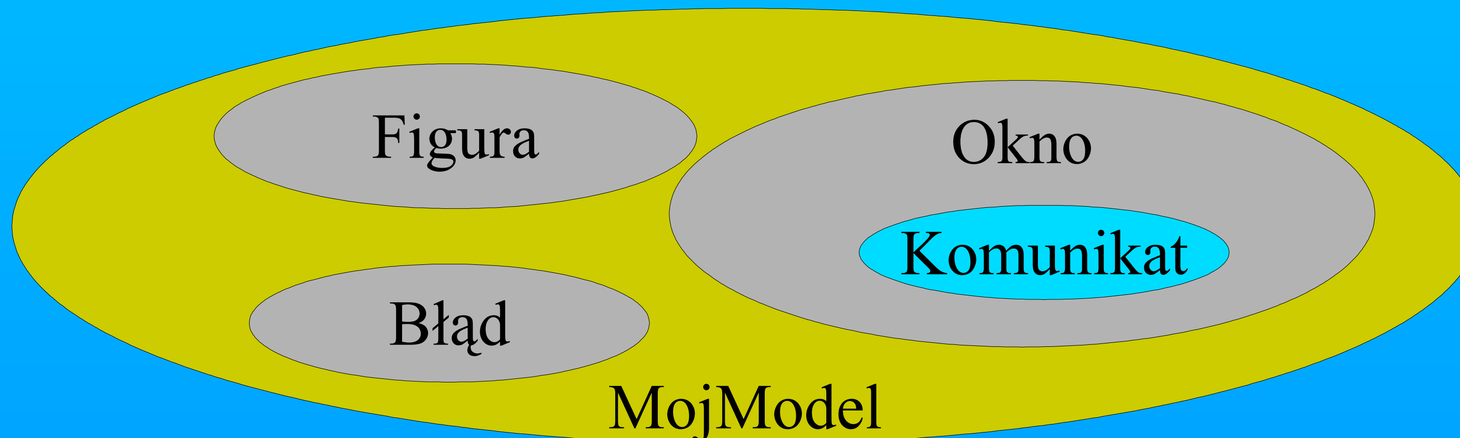
Obiekt, który nie ma nazwy jest obiektem anonimowym



Nazwa klasy

Nazwa jest napisem. Wyróżnia klasę spośród innych klas.

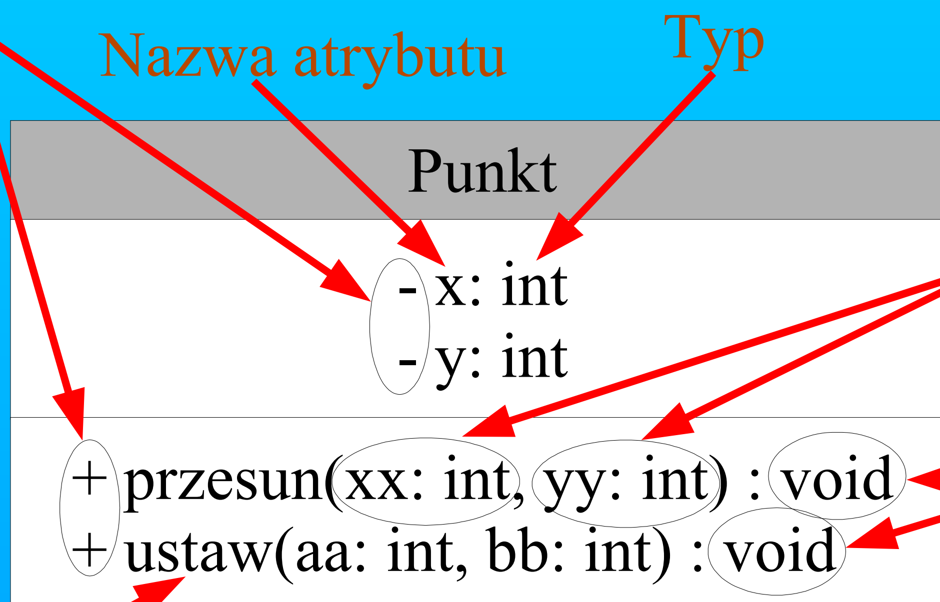
Nazwa ścieżkowa	Nazwa prosta
MojModel::Okno	Klient
MojModel::Okno::Komunikat	Okno
MojModel::Figura	Błąd



Atrybuty i operacje

Atrybut jest nazwaną właściwością klasy. Określa zbiór wartości jakie można przypisać do poszczególnych egzemplarzy tej klasy.

Widoczność:
- private
protected
+ public



Nazwa operacji

Operacja jest implementacją usługi, której wykonania moża zażądać od każdego obiektu klasy.

Atrybuty klasy

Ogólna defnicja atrybutu:

[widoczność] nazwa [liczebność] [:typ] [=wartość_początkowa][{określenie właściwości}]

[] - opcjonalnie

położenie
+ położenie
położenie: Punkt

Początek: *Element
nazwa[0..1]: String
położenie: Punkt = (0,0)
id: Integer {frozen}

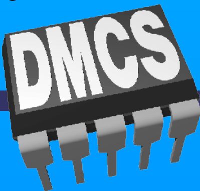
Atrybuty klasy

[widoczność] nazwa [liczebność] [:typ] [=wartość_początkowa][{określenie właściwości}]

public – (publiczny) Każdy zewnętrzny klasyfikator, który ma dostęp do danego klasyfikatora, ma także dostęp do takiego składnika. Na diagramie oznaczany +

protected – (chroniony) Każdy potomek klasyfikatora ma także dostęp do takiego składnika.
Na diagramie oznaczany #

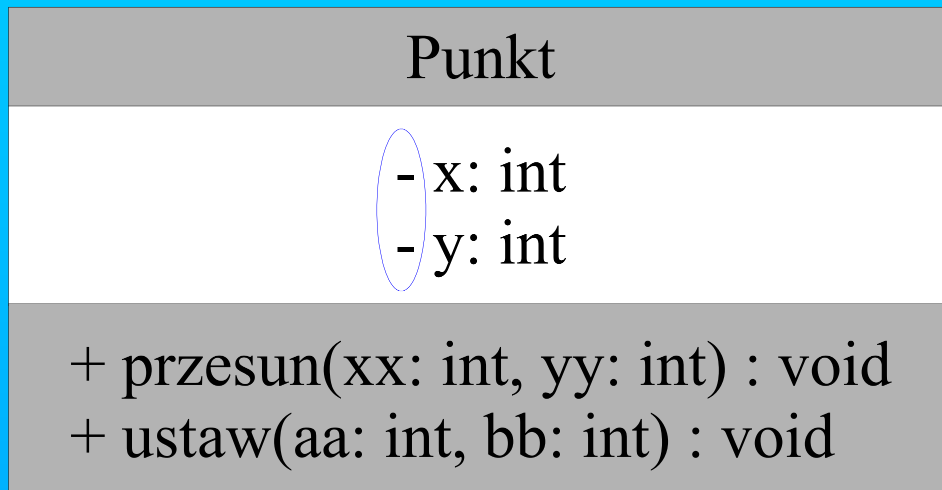
private – (prywatny) Tylko rozważany klasyfikator ma dostęp do takiego składnika. Na diagramie oznaczany -



Prywatność

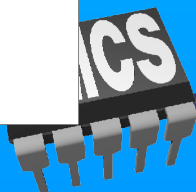
[widoczność] nazwa [liczebność] [:typ] [=wartość_początkowa][{określenie właściwości}]

UML



C++

```
class Punkt
{
    private:
            int x;
            int y;
    public:
            void przesun(int xx, int yy);
            void ustaw(int aa, int bb);
};
.....
Punkt p1;
int k;
p1.x =14;
k = p1.y;
```



Publiczność

UML

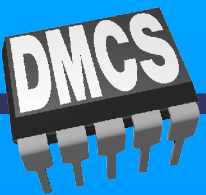
Punkt
- x: int - y: int
+ przesun(xx: int, yy: int) : void + ustaw(aa: int, bb: int) : void

C++

```
class Punkt
{
    private:
        int x;
        int y;
    public:
        void przesun(int xx, int yy);
        void ustaw(int aa, int bb);
};
.....
Punkt p1;
int k = -5;
p1.przesun(12,4);
p1.ustaw(k,26);
```

Chronienie

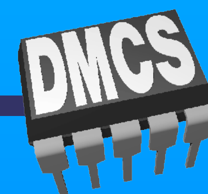
Omówione zostanie po uogólnieniach



Atrybuty klasy

[widoczność] nazwa [liczebność] [:typ] [=wartość_początkowa][{określenie właściwości}]

changeable – Nie ma ograniczeń co do modyfikacji wartości atrybutu.
addOnly – W wypadku atrybutów o liczebności większej niż jeden można dodawać nowe wartości, ale raz dodana wartość nie może być usunięta lub zmieniona.
frozen – Wartość atrybutu nie może być zmieniona po zainicjowaniu obiektu (= const w języku C/C++).



Atrybuty klasy

[widoczność] nazwa [liczebność] [:typ] [=wartość_początkowa][{określenie właściwości}]

liczebność – Liczba kardynalna określająca liczbę egzemplarzy atrybutu.
wartość początkowa – Wartość nadawana atrybutowi w momencie utorzenia obiektu.

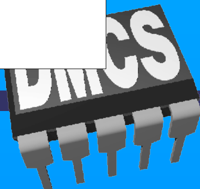
DMCS

Atrybuty klasy

[widoczność] nazwa [liczebność] [:typ] [=wartość_początkowa][{określenie właściwości}]

```
Lista[0..*] : Osoba
portKonsoli[2..*] : Port
pulpitSterujacy[1] : Pulpit
pojemność[0..5]: Jednostka

nazwisko : String = (“”)
ilość : long = (0)
tablicaInt[2]: Int = (2,4)
```

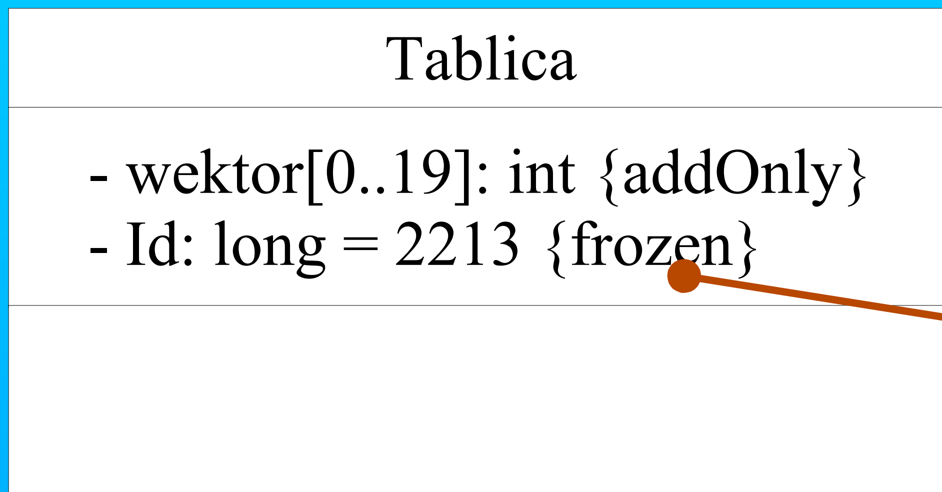
```
class Klasa
{
	private:
		String	nazwisko;
		int wiek;
		int tablica[20];
	public:
		Klasa():
				nazwisko(“Anonim”),
				wiek(20)
				{
					tablica[0]=0;
				}
};
```



Atrybuty – określenie właściwości

[widoczność] nazwa [liczebność] [:typ] [=wartość_początkowa][{określenie właściwości}]

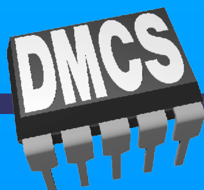
UML



C++

```
class Tablica
{
    private:
        int wektor[20];
        const long id;
    public:
        Tablica():id(2213L){}
};
.....
Tablica tab;
```

W języku C++ nie ma odpowiednika określenia właściwości addOnly



Właściwości operacji klasy

Ogólna defnicja operacji:

[widoczność] nazwa [(lista-parametrów)] [:typ_wyniku] [{określenie właściwości}]

wyświetl
+ wyświetl
ustaw(n: Nazwa, s: String)
pobierzID: Integer
wyzeruj() {guarded}

Uwaga!
Widoczność operacji definiowana jest tak samo jak widoczność atrybutów.

DMCS

Parametry operacji klasy

[widoczność] nazwa [(lista-parametrów)] [:typ_wyniku] [{określenie właściwości}]

W sygnaturze operacji może być dowolna liczba parametrów (odzielane są one wówczas przecinkiem), a może nie być ich wcale. Deklaracja każdego z parametrów ma postać:

[tryb] nazwa : typ [= wartość_domyślna]

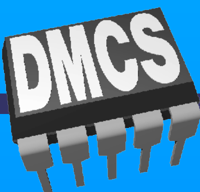
```
default(n: Nazwa = ”no name”, s: String, i int = 5)
next(out n: int, in p: parameter)
```

Parametry operacji klasy - tryb

[widoczność] nazwa [(lista-parametrów)] [:typ_wyniku] [{określenie właściwości}]

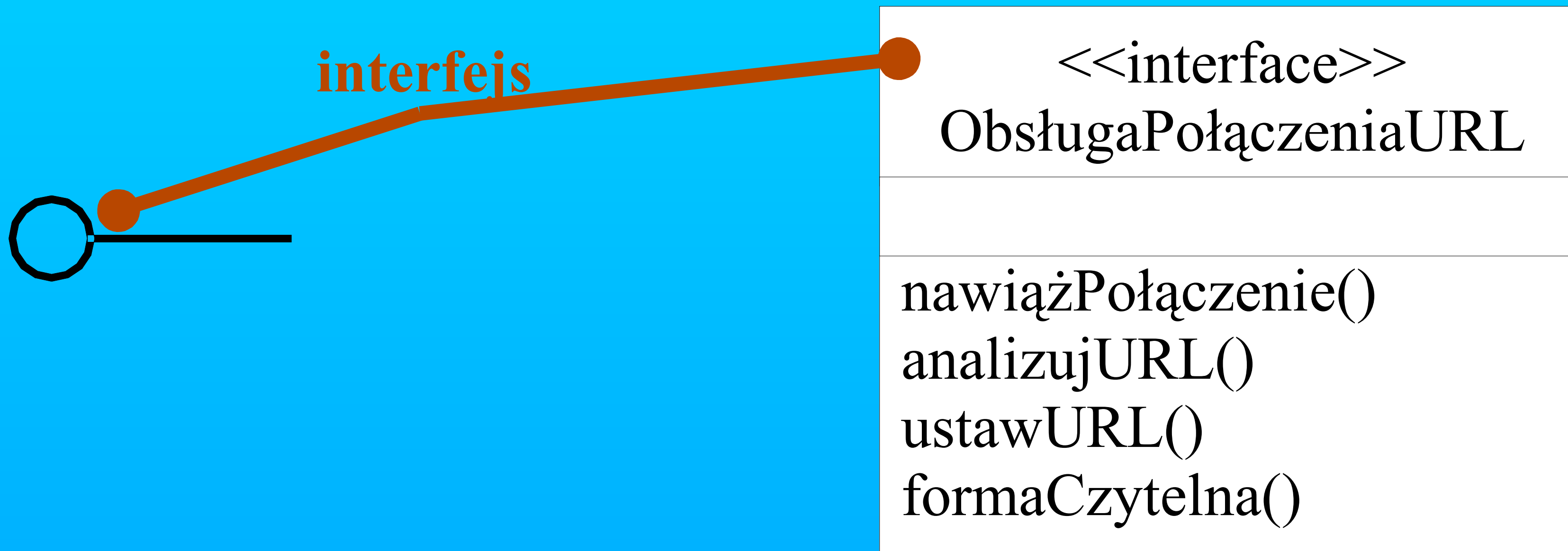
[tryb] nazwa : typ [= wartość_domyślna]

in – Parametr wejściowy; nie może być modyfikowany.
out – Parametr wyjściowy; może być modyfikowany w celu przekazania informacji wywołującemu.
inout – Parametr wejściowy; może być modyfikowany.
return – Parametr zwracany przez operacje.



Interfejsy

Interfejs to zestaw operacji które wyznaczają usługi oferowane przez klasę lub komponent.



Interfejs nie posiada argumentów!
Interfejs definiowany jest jako stereotypowana klasa. Na digramach może przedstawiony za pomocą ww klasyfikatorów.

Dziedziczenie

Wydarzenie
{root}

korzeń

przodek
(nadklasa)

uogólnienie

Prozaiczne

Wypadek

Katastrofa

potomek
(podklasa)

liść

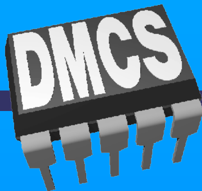
KotNaDrzewie
{leaf}

WypadekDrogowy

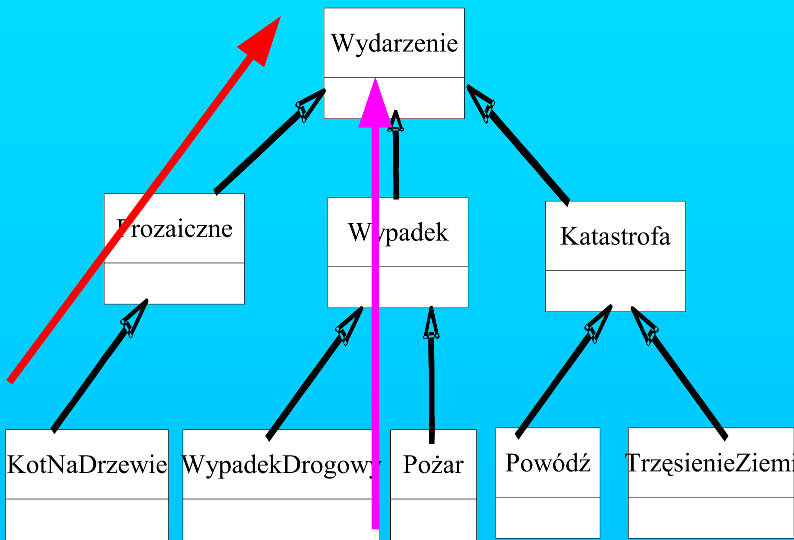
Pożar

Powódź
{leaf}

TrzęsienieZiemi



Dziedziczenie C++




```
class Wydarzenie
{

};
class Prozaiczne : public Wydarzenie
{

};
class KotNaDrzewie : public Prozaiczne
{

};
```

```
class Wydarzenie
{

};
class Wypadek : public Wydarzenie
{

};
class WypadekDrogowy : public Wypadek
{

};
class Pożar : public Wypadek
{

};
```

Dziedziczenie, polimorfizm

uogólnienie – związek między elementem ogólnym (przodkiem) a pewnym specyficznym jego rodzajem (potomkiem). Stosowane do oznaczenia dziedziczenia. Uogólnienie polega na tym, że potomek może wystąpić wszędzie tam, gdzie jest spodziewany jego przodek a nie na odwrót. Potomek dziedziczy wszystkie właściwości przodka, w szczególności atrybuty i operacje. Uogólnienie może posiadać nazwę.

Wyróniono 1 stereotyp i 4 ograniczenia dla uogólnienia. Zwykłe uogólnienie bez dodatków wystarcza do modelowania większości związków dziedziczenia.

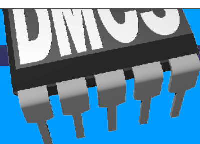
Dziedziczenie, polimorfizm

polimorfizm – polega na tym, że operacja potomka mająca tę samą sygnaturę co operacja przodka jest ważniejsza (ma pierszeństwo).
root (korzeń) – klasa bez przodków z conajmniej jednym potomkiem. Klasa nie może mieć przodków.
(leaf) liść – klasa bez potomków. Klasa nie może mieć potomków.

Dziedziczenie, polimorfizm

```
// Dziedziczenie
class Wydarzenie
{

};
class Wypadek : public Wydarzenie
{

};

void funkcja(Wydarzenie *pW);

int main(void)
{
	Wypadek wyp;

	funkcja(&wyp);
	return 0;
}
```

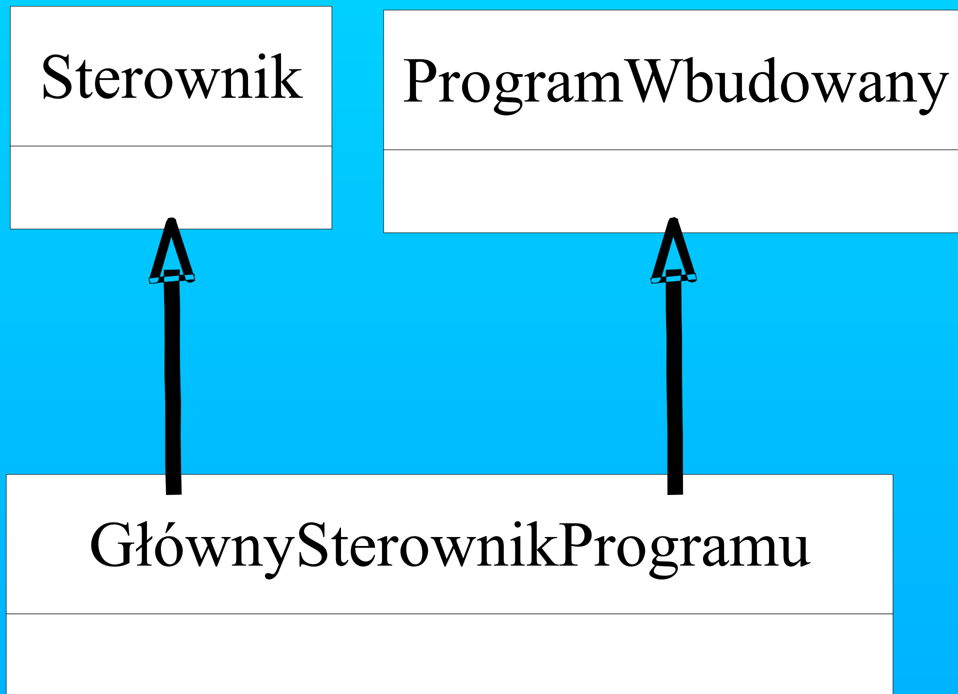
```
// Polimorfizm
class Wydarzenie
{ public:
	void wyswietl(){ cout << "operacja wydarzenia"; }
};
class Wypadek : public Wydarzenie
{ public:
	void wyswietl(){ cout << "operacja wypadku"; }
};

int main(void)
{
	Wypadek wyp;

	wyp.wyswietl();

	return 0;
}
```



Funkcje wirtualne C++

```
class Wydarzenie
{	public:
		void virtual wyswietl(){ cout << "operacja wydarzenia"; }
};
class Wypadek : public Wydarzenie
{	public:
		void wyswietl(){ cout << "operacja wypadku"; }
};
class Katastrofa : public Wydarzenie
{	public:
		void wyswietl(){ cout << "operacja katastrofy"; }
};
int main(void)
{
	Wypadek wyp;
	Katastrofa kat;
	Wydarzenie *pWyd = &kat;
	Wydarzenie &refWyd = wyp;

	pWyd->wyswietl();
	refWyd.wyswietl();

	return 0;
}
```

Wielodziedziczenie




```
class Sterownik
{

};
class ProgramWbudowany
{

};

class GłównySterownikProgramu:
public Sterownik, public ProgramWbudowany
{

};
```

wielodziedziczenie – (dziedziczenie wielobazowe) klasa ma wiele przodków.

Abstrakcyjność

klasa abstrakcyjna

Ikona {root}
położenie: Punkt
wyświetl() pobierzID(): integer {leaf}

operacja abstrakcyjna

operacja konkretna

klasa abstrakcyjna

IkonaProstokątna
wysokość : integer szerokość : integer

IkonaDowolna
Brzeg: ZestawOdcinków
leżyWewn(p : Punkt) : Boolean

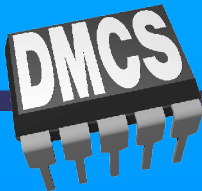
operacja polimorficzna

Przycisk
wyświetl()

klasa konkretna

PrzyciskOK {leaf}
wyświetl()

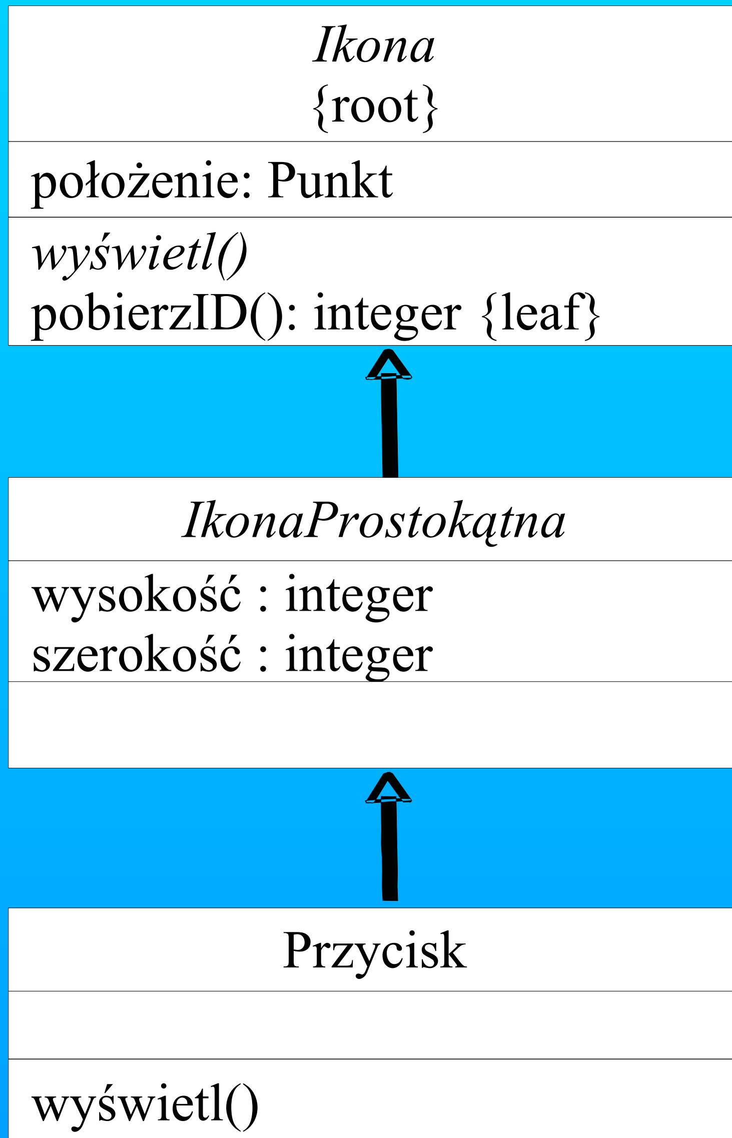
klasa liść



Abstrakcyjność

Abstrakcyjność klas – klasy, które nie mogą mieć bezpośrednich egzemplarzy są klasami abstrakcyjnymi. Klasy abstrakcyjne oznaczane są nazwami pisanymi kursywą.
Abstrakcyjność operacji – operacja nie posiadająca implementacji. Jej implementację muszą zapewnić potomkowie. Operacje abstrakcyjne oznaczane są nazwami pisanymi kursywą.
Operacja liść – operacja nie jest polimorficzna. Nie może być zastąpiona w potomkach. Operacje liście oznaczane są napisem leaf umieszczonym w jej metce.

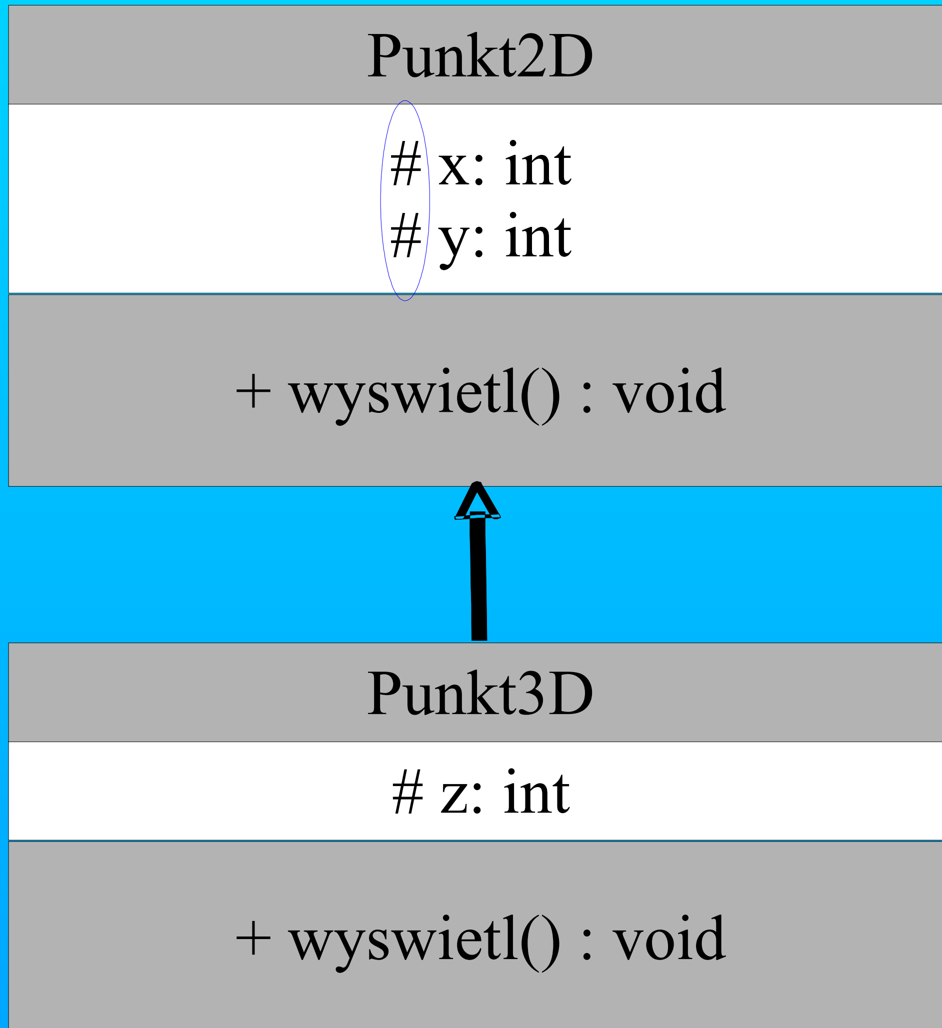
Abstrakcyjność C++




```
class Ikona
{ public:
      Punkt polozenie;
      void virtual wyświetl()=0;
};

class IkonaProstokatna : public Ikona
{ public:
      int wysokosc, szerokosc;
};

class Przycisk : public IkonaProstokatna
{ public:
      void wyświetl(){cout<< “ikona prostokatna” ;}
};
```

Chronienie

UML

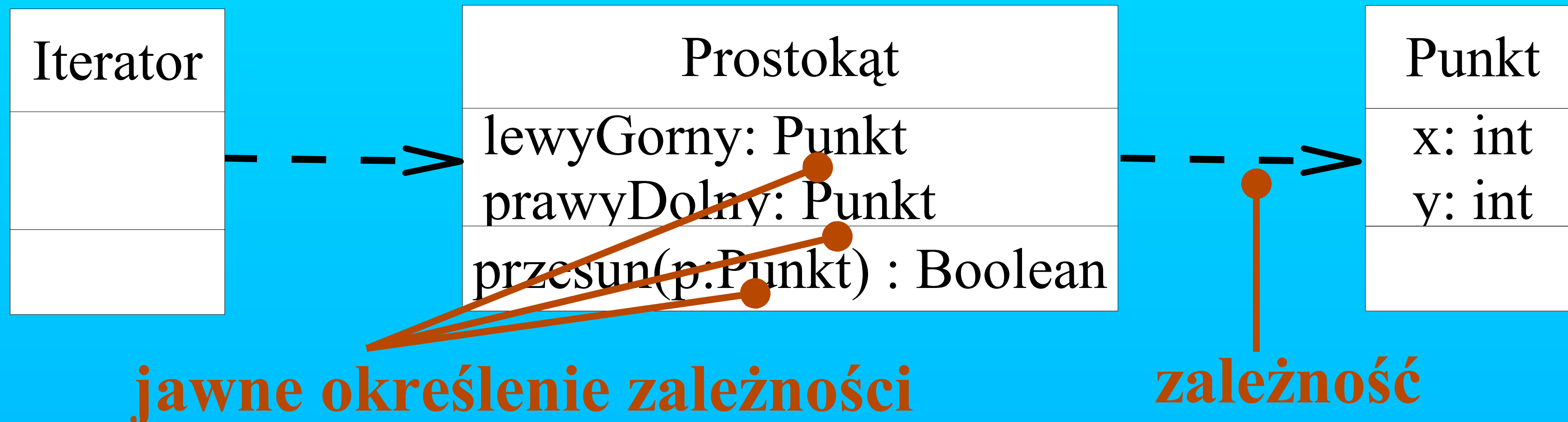


C++

```
class Punkt2D
{     protected:
            int x;
            int y;
      public:
            void wyswietl()
      {cout << x << ' ' << y; }
};
class Punkt3D: public Punkt2D
{     protected:
            int z;
      public:
            void wyswietl()
      {cout << x << ' ' << y << ' ' << z; }
};
.....
Punkt2D p2D;
p2D.x=5;
p2D.y=0;
```

(the last two lines are struck through on the slide)

Zależność



zależność – jest to związek użycia między elementami. Klasa Prostokąt zależy od klasy Punkt. Punkt nie musi nic "wiedzieć" o prostokącie. Zmiany wprowadzone w klasie Punkt mogą mieć wpływ na klasę Prostokąt. Wyróżniono 17 stereotypów dla tego typu związku. Zwykła zależność bez dodatków wystarcza do modelowania większości związków użycia.

Powiązanie

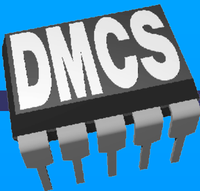
Biblioteka | 1 | zawiera ▶ | * | Książka

nazwa, kierunek nazwy, liczebność, kierunek nawigacji

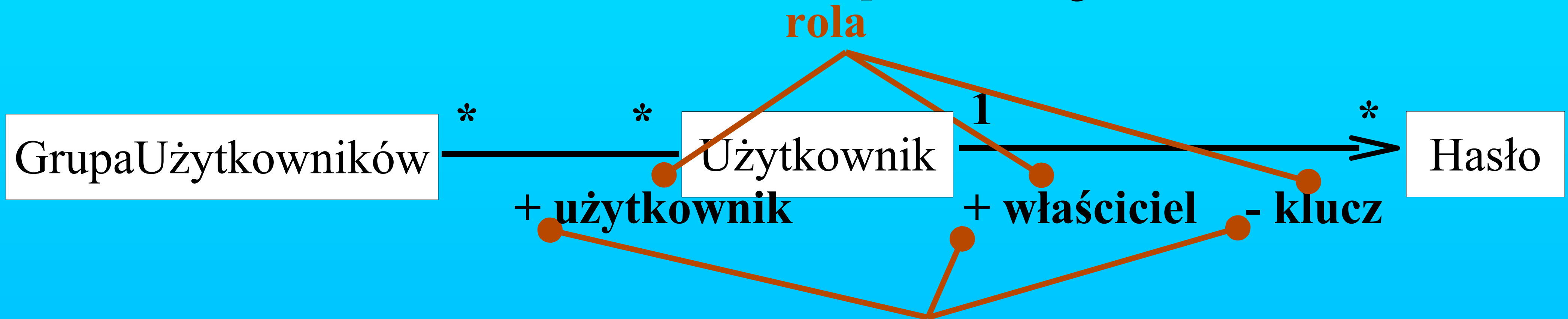
Osoba — pracownik — pracuje dla — pracodawca — Przedsiębiorstwo

rola, powiązanie

powiązanie – jest to związek strukturalny, który wskazuje, że obiekty jednego elementu są połączone z obiektami innego.
Klasa Biblioteka ma powiązanie jeden-do-wielu z klasą Książka. Mając Bibliotekę można wyznaczyć jej wszystkie książki. Mając Książkę można wyznaczyć do jakiej biblioteki ona należy.
Powiązanie wyraża "równorzędność" lub "starszeństwo" egzemplarzy.
Dostępne są 4 podstawowe dodatki do powiązań: nazwa, rola, liczebność (przy każdym końcu) oraz agregacja.
Rola to zachowanie bytu w określonym kontekście.



Widoczność powiązania

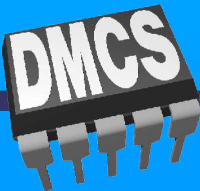


widoczność powiązania – określa widoczność obiektów biorących udział w powiązaniu na zewnątrz. Widoczność jest określona i oznaczana analogicznie jak w przypadku klas.

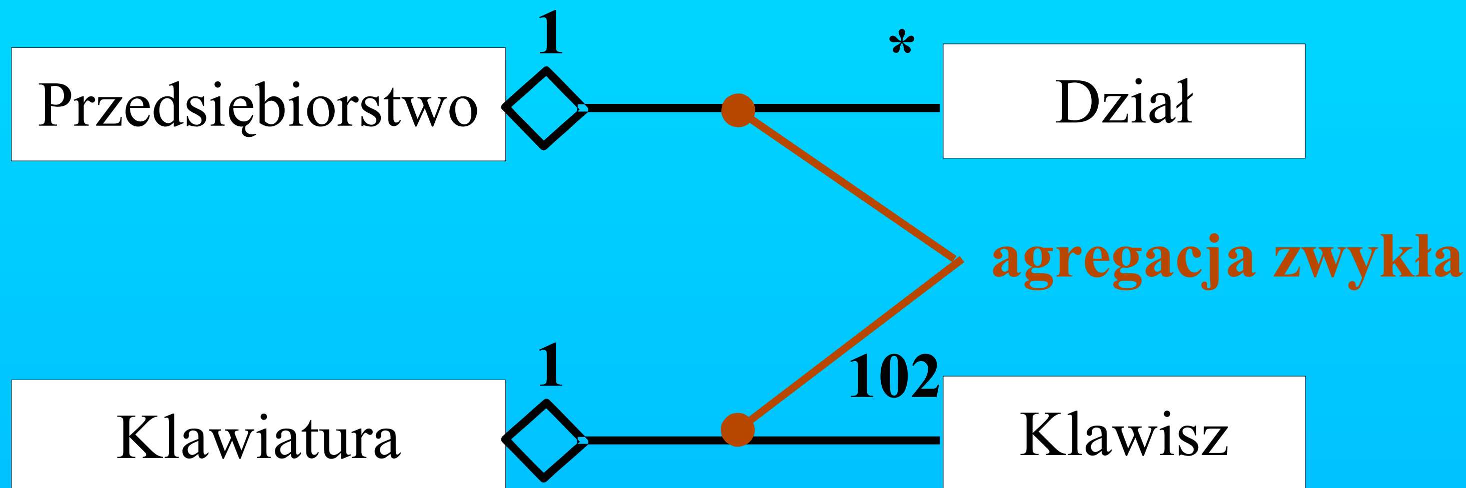
public – wartość domyślna. Obiekty powiązania są widoczne na zewnątrz.

private – Obiekty powiązania są niewidoczne dla obiektów nie biorących udziału w powiązaniu.

protected – Obiekty powiązania są niewidoczne dla obiektów nie biorących udziału w powiązaniu, z wyjątkiem potomków klasy z przeciwnego końca.

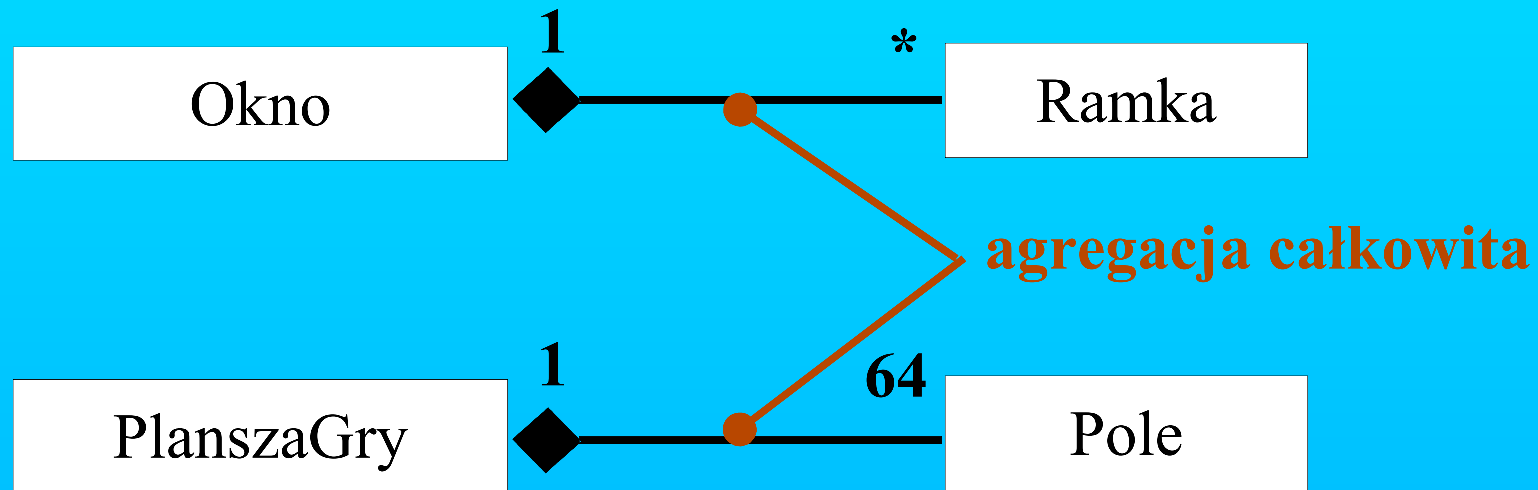


Agregacja zwykła



Zazwyczaj powiązanie jest związkiem strukturalnym równorzędnych partnerów (nie wyróżniamy wówczas kierunku nawigacji). Oznacza to że powiązane klasy znajdują się na tym samym poziomie pojęciowym, czyli żadna nie jest ważniejsza.
Szczególnym przypadkiem ww powiązania jest agregacja. Stusujemy ją wówczas, gdy chcemy wyrazić związek "całość składająca się z części". Np. Przedsiębiorstwo składa się z Działów.
Uwaga! Część może należeć do kilku całości i "żyć" w czasie niezależnie od całości.

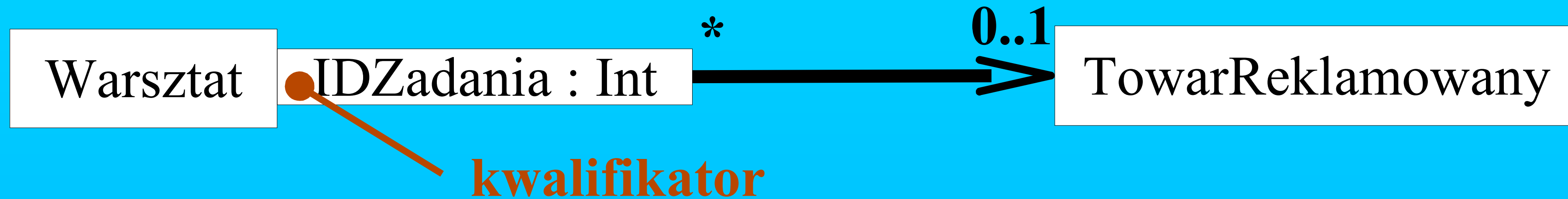
Agregacja całkowita



agregacja całkowita – oznacza, że część biorąca udział w agregacji może należeć tylko do jednej całości oraz, że o czasie życia części decyduje całość. Całość musi zadbać o stworzenie i zniszczenie części. (Czas życia części <= czasowi życia całości).

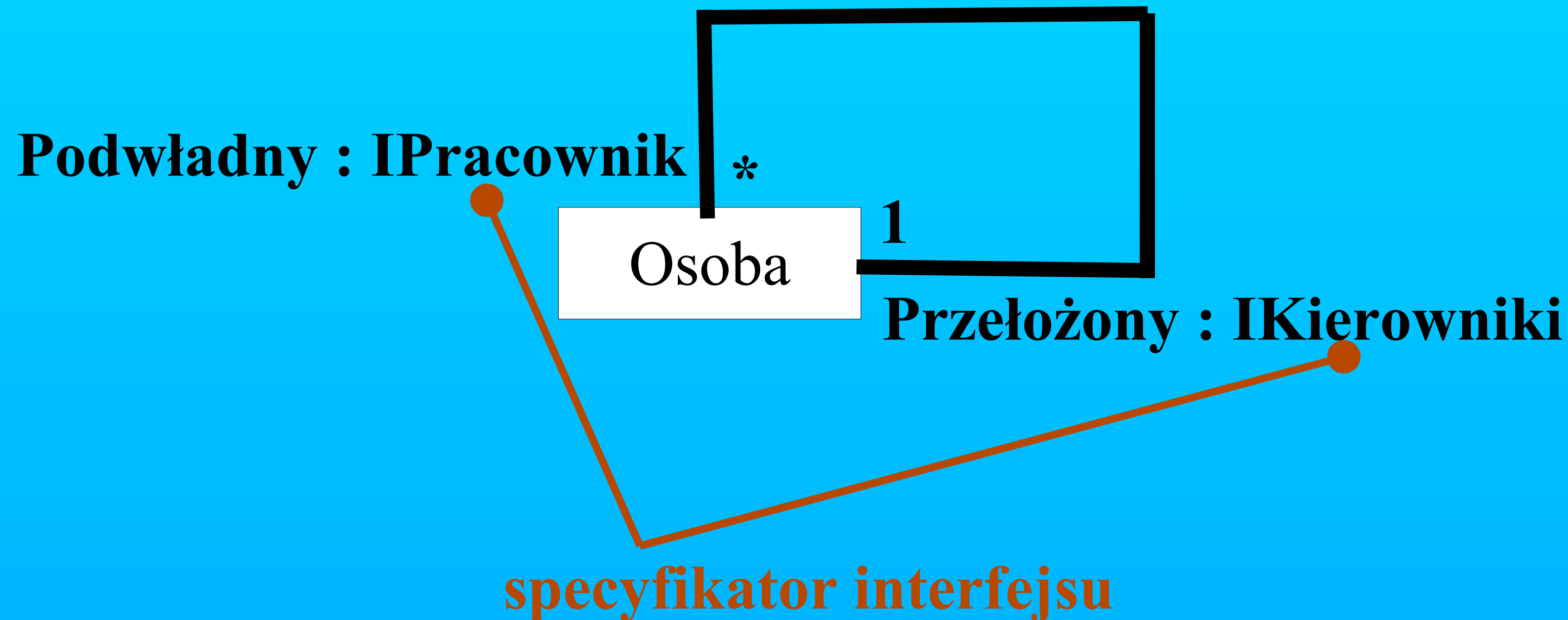
DMCS

Kwalifikacja



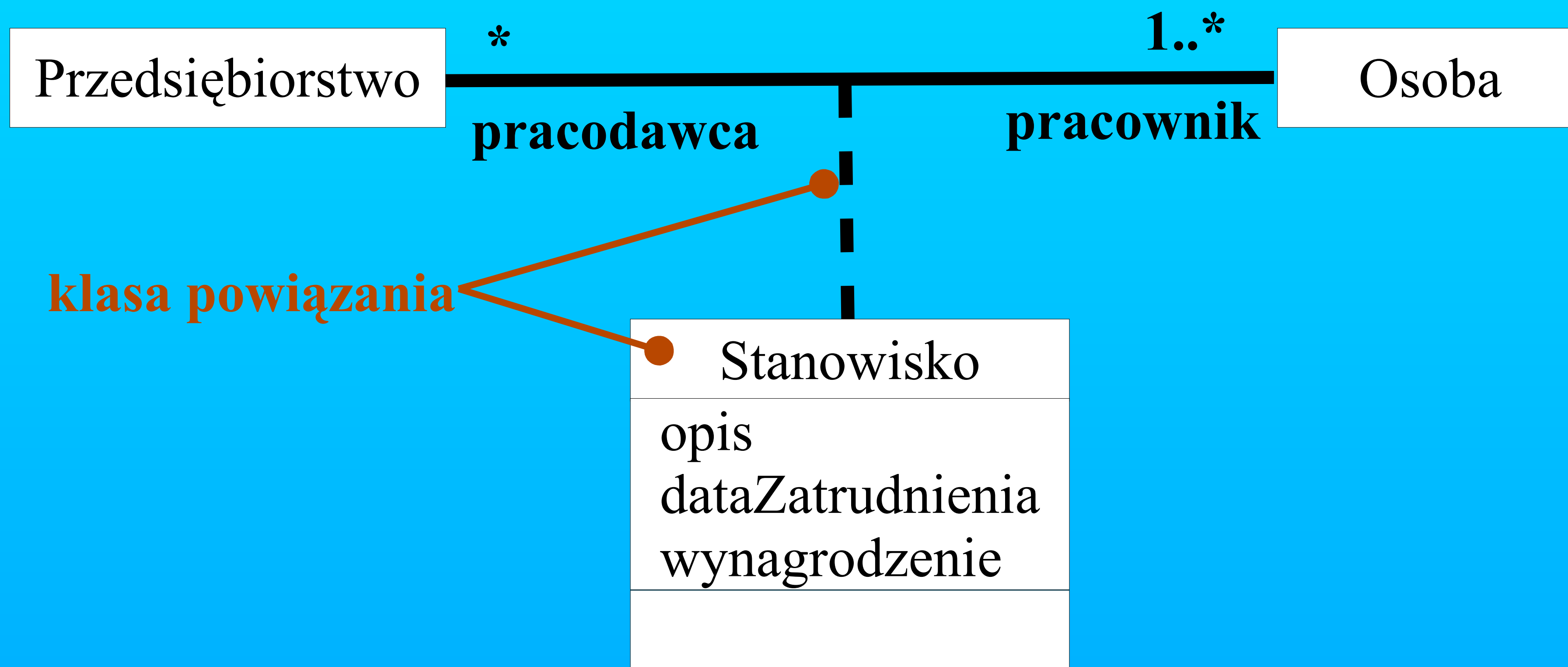
kwalifikacja – ma zastosowanie w przypadku wyszukiwania zbioru obiektów. Obiekt źródłowy (Warsztat) wraz z wartościami atrybutów kwalifikatora wskazuje obiekt docelowy (TowarReklamowany) lub zbiór obiektów. Kryteria wyszukiwania definiowane są jako kwalifikatory.

Specyfikator interfejsu



specyfikator interfejsu – określa jaki interfejs udostępnia klasa w danej roli.

Klasa powiązania



klasa powiązania – byt posiadający cechy klasy i powiązania. Umożliwia zdefiniowanie właściwości powiązania jeśli są potrzebne.

Podstawowe elementy występujące w diagramie obiektów

Na diagramie obiektów przedstawia się egzemplarze elementów z diagramu klas. Obrazuje on zbiór elementów i ich związków w ustalonej chwili. Diagramy obiektów wyobrażają rzut systemu w danej chwili (= statyczna część diagramu interakcji). Uwzględnia się w nich zbiór obiektów, ich stan i związki. Zawierają na ogół obiekty i wiązania, mogą zwierać notatki i ograniczenia.

Na kolejnych slajdach przedstawione zostaną elementy, które mogą wystąpić w diagramach obiektów.

Elementy diagramu obiektów

obiekt

p: Przedsiębiorstwo

wiązanie

Wiązanie omówione zostanie wraz z interakcjami

d1 : Dział
nazwa = “Sprzedaż”

d1 : Dział
nazwa = “Badania i rozwój”

d3 : Dział
nazwa = “Sprzedaż w Polsce”

kierownik

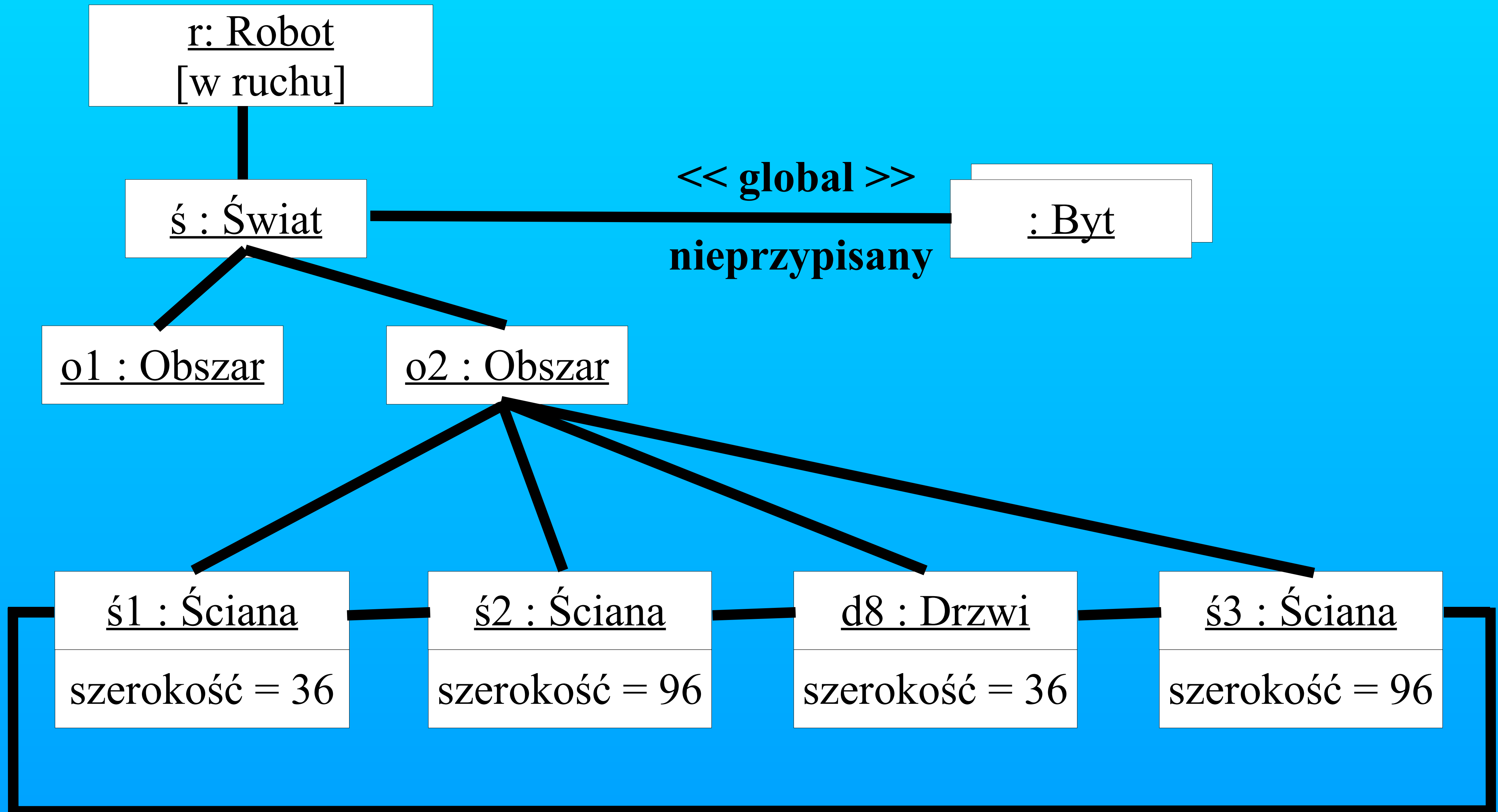
wartość atrybutu

obiekt anonimowy

o : Osoba
Nazwisko = “Erin” IDPracownika = 8765 stanowisko = “Szef sprzedaży”

: AdresyKontaktowe
adres = “kluczowa 10”

Obiekty, wiązania - przykład



Diagramy interakcji. Interakacja.

interakcja to zachowanie polegające na wymianie komunikatów między obiektami w pewnym otoczeniu, w pewnym celu.

Podstawowe elementy występujące w diagramie interakcji

Diagramy interakcji obrazują interkację jako zbiór obiektów i związków między nimi, w tym też komunikaty, jakie obiekty przekazują między sobą. Diagram interakcji jest w istocie rzutem bytów biorących udział w interakcji.
Diagram przebiegu jest diagramem interakcji, na którym uwypukla się kolejność komunikatów w czasie.
Diagram kooperacji jest diagramem interakcji, na którym uwypukla się związki strukturalne między obiektami wysyłającymi i odbierającymi komunikaty.
Diagramy kooperacji i przebiegu są równoważne.
Na kolejnych slajdach przedstawione zostaną elementy, które mogą wystąpić w diagramach interkacji.

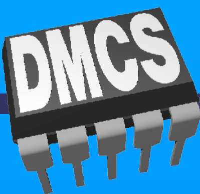
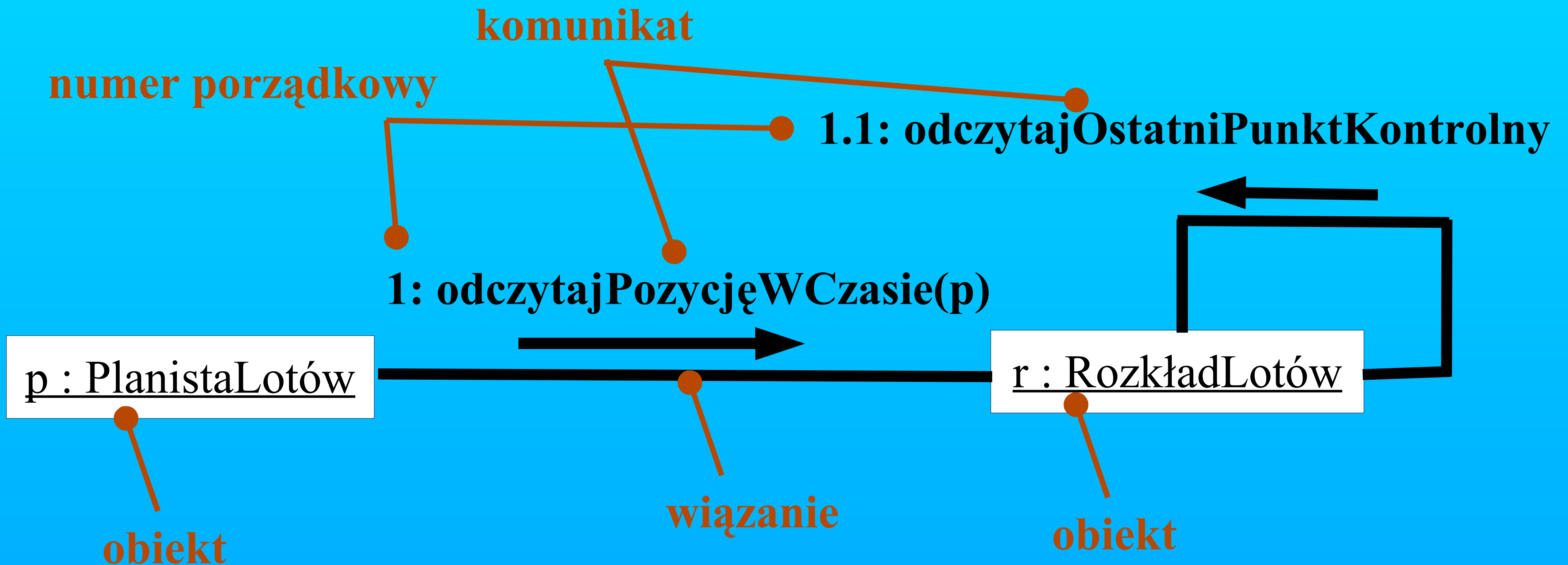


Diagram kooperacji



Na diagramie kooperacji uwypukla się organizację obiektów biorących udział w interakcji.

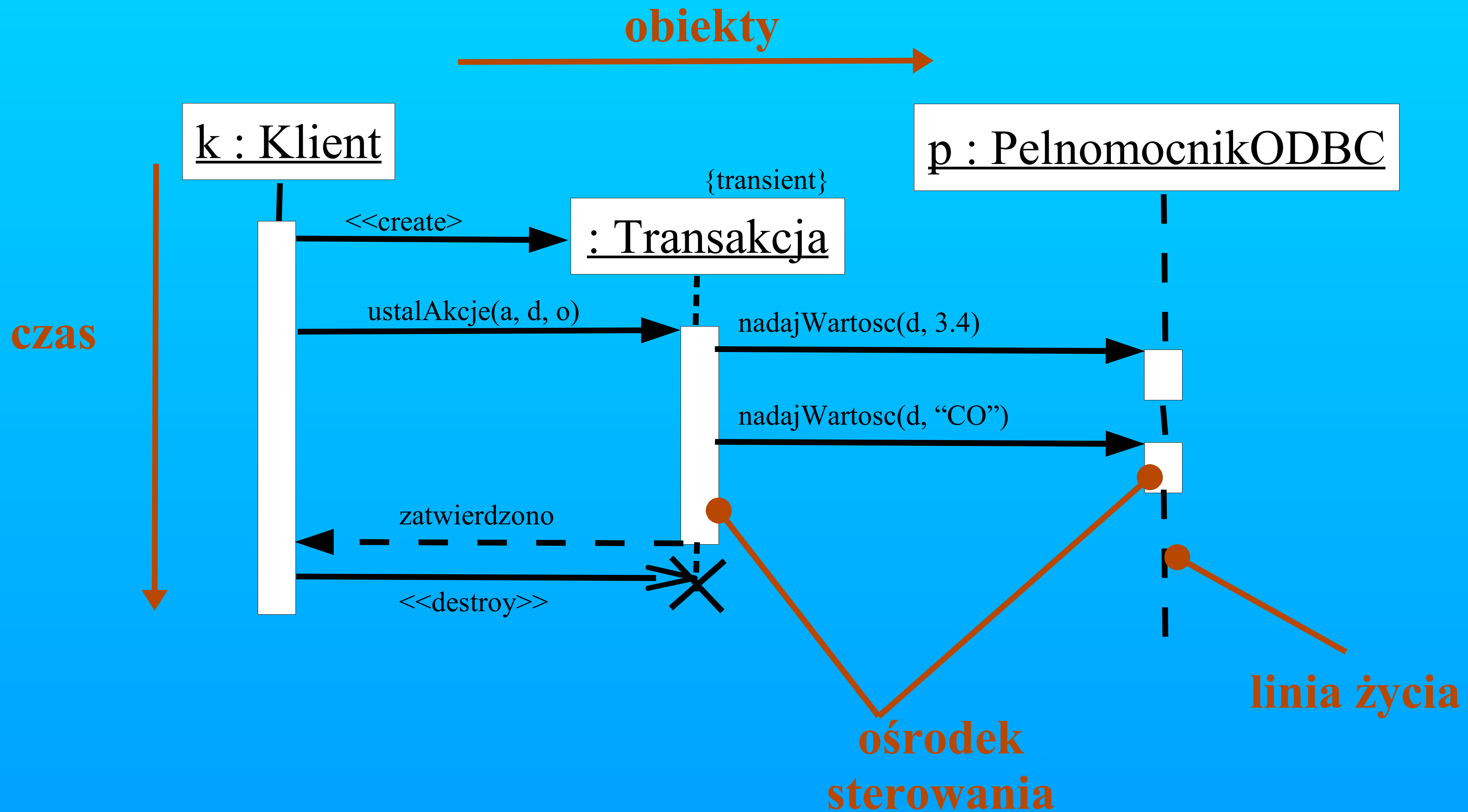
DMCS

Wiązania, komunikaty

wiązanie jest zazwyczaj egzemplarzem powiązania. Jeśli dana klasa ma powiązanie z inną klasą to między egzemplarzami klas może wystąpić wiązanie.

komunikat to specyfikacja łączności między obiektami, uwzględniająca zlecenia wykonania określonych czynności. Oznaczenia komunikatów poprzedzone są numerami oznaczającymi kolejność komunikatu w czasie. Komunikaty przekazywane są wzdłuż wiązania.

Diagram przebiegu.



DMCS

Równoważność d. kooperacji i diagramu przebiegu

k : Klient

{transient}

: Transakcja

p : PelnomocnikODBC

<<create>

ustalAkcje(a, d, o)

nadajWartosc(d, 3.4)

nadajWartosc(d, “CO”)

zatwierdzono

<<destroy>>

k : Klient

1: <<create>>
2: ustaAkcje(a,d,o)
3:<<destroy>>

<<local>>

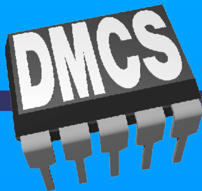
: Transakcja

{transient}

<<global>>

p : PelnomocnikODBC

2.1: nadajWartosc(d, 3.4)
2.2: nadajWartosc(a, “CO”)



Diagramy komponentów.

Diagramy komponentów przedstawia fizyczne aspekty systemów obiektowych. Obrazuje uporządkowanie komponentów i zależności między nimi.
Używany do modelowania statycznych aspektów perspektywy implementacyjnej systemu. Diagramy komponentów są w istocie diagramami klas, na których kładzie się nacisk na komponenty systemu.

Diagram komponentów

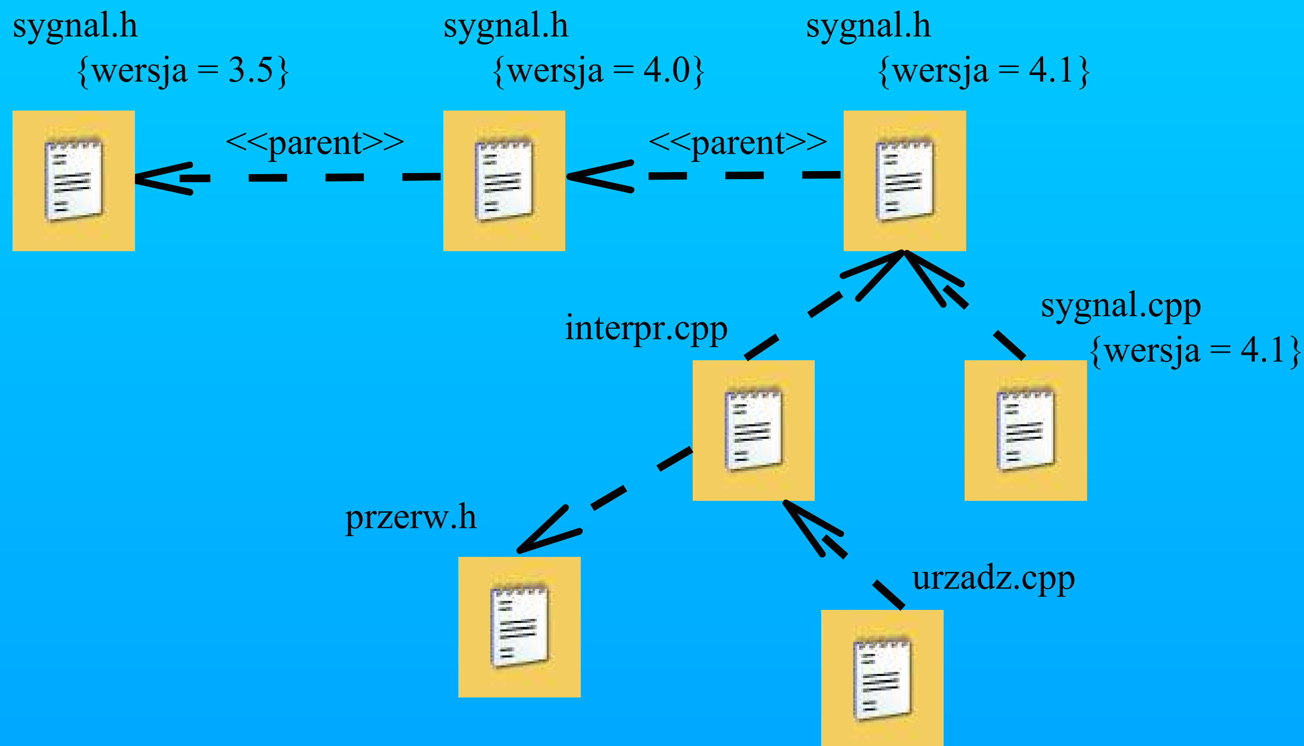
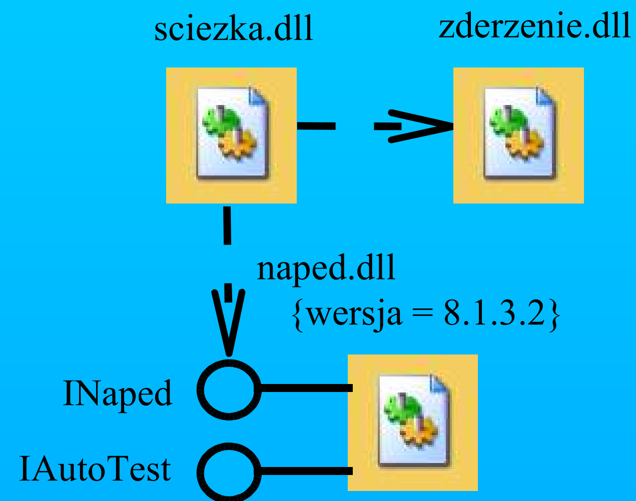


Diagram komponentów



DMCS