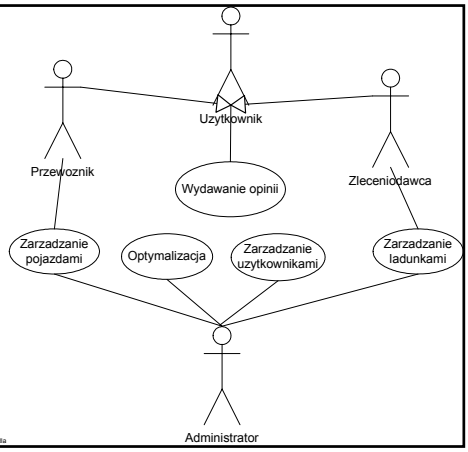
UML a kod w C++ i Javie

- Projektowanie oprogramowania
- Dokumentowanie oprogramowania

Diagramy przypadków użycia



Klasy użytkowników i wykorzystywane funkcje

- Mogą sugerować podział systemu na odrębne aplikacje, np.
  - aplikacja dla użytkowników systemu
  - aplikacja dla administratora
- Elementy interfejsu użytkownika, np.
  - różne tryby pracy, np. różne systemy menu
  - oddzielne fragmenty w strukturze menu
  - blokowanie dostępu do pewnych funkcji

Przypadki użycia

- Funkcje systemu
- Elementy interfejsu użytkownika
  - menu, podmenu, polecenia w menu
  - dialogi

Związki pomiędzy przypadkami użycia

- Struktura menu
- Polecenia dostępne w dialogach, np. wywoływanie innych dialogów
- Kreatory (creators, wizards)

Diagramy klas

- Bezpośrednie przełożenie na kod
- Wiele dodatkowych elementów wykorzystywanych na etapie projektowania

Klasa

Pojazd

```
class Pojazd {
...
}
```

Nazwy - prefixy, suffixy, zamiana spacji i niedozwolonych znaków

Student dzienny

```
class CStudentDzienny
{
...
}
```


Pola C++

Pojazd
Nazwa
Cena kilometra
Cena godziny

```
class CPojazd {
...
  ... Nazwa;
  ... CenaKilometra;
  ... CenaGodziny;
}
```

Na przykład:

```
class CPojazd {
protected:
  char* Nazwa;
  double CenaKilometra;
  double CenaGodziny;
}
```


Pola Java

Pojazd
Nazwa
Cena kilometra
Cena godziny

```
class Pojazd {
  ... nazwa;
  ... cenaKilometra;
  ... cenaGodziny;
}
```

Na przykład:

```
class CPojazd {
  protected String nazwa;
  protected double cenaKilometra;
  protected double cenaGodziny;
}
```


Symbole widoczności pól i operacji

\+ public – publiczne
\# protected – zabezpieczone/chronione
\- private – prywatne

Pojazd
+Nazwa
#Cena kilometra
-Cena godziny

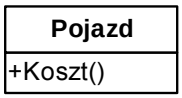
```
class CPojazd {
public:
  ... Nazwa;
protected:
  ... CenaKilometra;
private:
  ... CenaGodziny;
}
```

```
class Pojazd {
  public ... nazwa;
  protected ... cenaKilometra;
  private ... cenaGodziny;
}
```


Typy pól

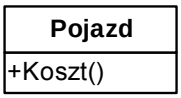
Pojazd
+Nazwa : char*
#Cena kilometra : double
-Cena godziny : double

Operacje i metody C++



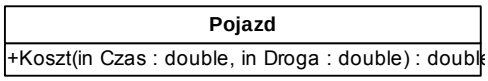

```
class CPojazd {
...
public:
  ... Koszt (...);
};
...

... CPojazd::Koszt (...) {
  ...
}
```


Operacje i metody Java



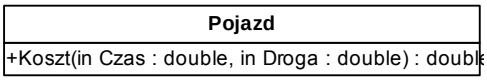

```
class Pojazd {
...
  public ... koszt (...) {
    ...
  }
}
```


Nagłówki operacji C++



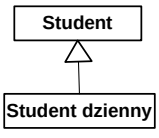

```
class CPojazd {
...
public:
  double Koszt (double Czas, double Droga);
};
...
double CPojazd::Koszt (double Czas, double
Droga) {
  ...
}
```


Nagłówki operacji Java



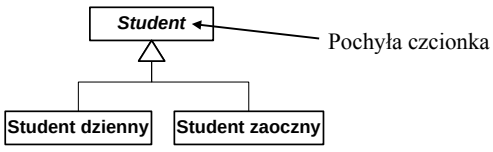

```
class Pojazd {
...
  public double koszt (double czas, double droga) {
    ...
  }
}
```


Generalizacja-specjalizacja - dziedziczenie




```
class CStudentDzienny :
  public CStudent {
  ...
}

class StudentDzienny
  extends Student {
  ...
}
```


Klasy abstrakcyjne



Klasy nie posiadające obiektów (bezpośrednio tej klasy)

Klasy abstrakcyjne C++

- Brak tworzenia obiektów tej klasy w kodzie
- Operacje abstrakcyjne – muszą być zdefiniowane w każdej ze specjalizacji, której obiekty będą tworzone

```
class CStudent {
  ...
  virtual CGrupa* PodajGrupe () = 0;
}
```


Klasy abstrakcyjne Java

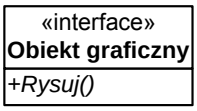
```
abstract class Student {
  ...
}
```

albo

```
abstract class Student {
  ...
  abstract CGrupa podajGrupe ();
}
```


Interfejsy (interfaces)

- Zbiór operacji (deklaracji metod)
  - *Przypomina klasę zawierającą wyłącznie operacje abstrakcyjne*
  - *Może zawierać stałe*



Interfejsy w C++

- Nie wspierane?

```
class CObiektGraficzny {
public:
  virtual void Rysuj () = 0;
}
```

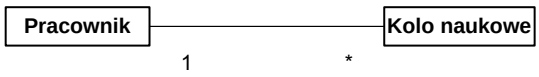

Interfejsy w Javie

```
interface IObiektGraficzny {
  void rysuj ();
}
```

Implementacja interfejsu

```
class Rysunek implements IObiektGraficzny
{
...
  public void rysuj ();
}
```

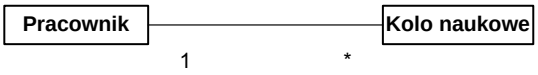

Związki klas



- Ogólnie dowolny sposób pozwalający na przechowanie informacji o powiązanych obiektach
  - Np. tablica zawierająca pary powiązanych obiektów

Kolo naukowe	Pracownik
K. Informatyki	Nowak
K. Fizyki	Kamiński
K. Chemii	Zieliński

Związki klas

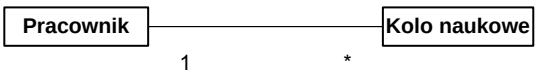


- Najczęściej dodatkowe pola przechowujące informacje o powiązanych obiektach
  - Każdy obiekt klasy Pracownik będzie przechowywał informacje o powiązanym obiekcie (dokładnie jednym jednym) klasy Kolo naukowe
  - Każdy obiekt klasy Kolo naukowe będzie przechowywał informacje o powiązanych obiektach (dowolnej liczbie) klasy Pracownik

Sposób przechowywania informacji o powiązanych obiektach

- Identyfikatory (np. nazwy)
- Wskaźniki/referencje

Związki w C++



- Najczęściej wskaźniki

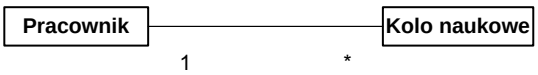
```
class CPracownik {
...
protected:
 vector <CKoloNaukowe*> rKoloNaukowe;
}
```

```
class CKoloNaukowe {
...
protected:
 CPracownik* rPracownik;
}
```


Związki w C++

- Krotność 1
  - Wskaźnik, który musi wskazywać na powiązany obiekt
- Krotność 0..1
  - Wskaźnik, który może mieć wartość NULL
- Krotność *, 1..*
  - Klasa vector (biblioteka STL) – dla 1..* nie może być pusty
  - Tablica wskaźników
  - Inna struktura danych

Związki w Javie



- Najczęściej referencje i ich kolekcje

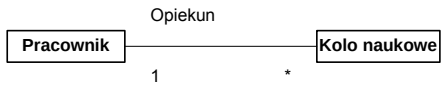
```
class Pracownik {
 ...
 protected Vector rKoloNaukowe;
 // lub
 protected KoloNaukowe[] rKoloNaukowe;
}
```

```
class KoloNaukowe {
 ...
 protected Pracownik
   pracownik;
}
```


Związki w Javie

- Krotność 1
  - Referencja, która musi wskazywać na powiązany obiekt
- Krotność 0..1
  - Referencja, która może mieć wartość NULL
- Krotność *, 1..*
  - Obiekt klasy z biblioteki standardowych struktur danych Javy – dla 1..* nie moze byc pusty
  - Tablica referencji
  - Inna struktura danych

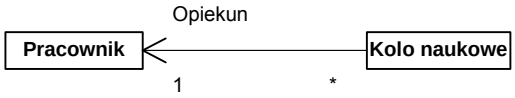
Wykorzystanie nazw ról w związkach




```
class CKoloNaukowe {
...
protected:
  CPracownik* rOpiekun;
}
```

```
class KoloNaukowe {
...
protected Pracownik opiekun;
}
```


Związki skierowane

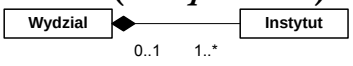



```
class CPracownik {
...
}
```

```
class CKoloNaukowe {
...
protected:
 CPracownik* rPracownik;
}
```

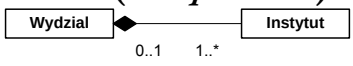
Brak informacji o powiązanych obiektach klasy Kolo naukowe

Związek kompozycji (*composition*)



- W zasadzie na poziomie implementacji nierozróżnialne od związków zwykłych
- Często obiekt będący całością jest odpowiedzialny za przechowywanie swoich składowych (dodawanie, usuwanie)

Związek kompozycji (*composition*)



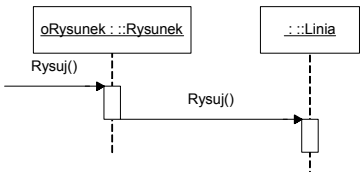
- W C++ czasami wykorzystanie obiektów zamiast wskaźników

```
class CWydzial {
...
protected:
  vector <CInstytut> rInstytut;
}
```


Diagramy sekwencji

- Wywoływanie metod w programie
- Podstawa implementacji metod

Wywołanie metody (*call*)

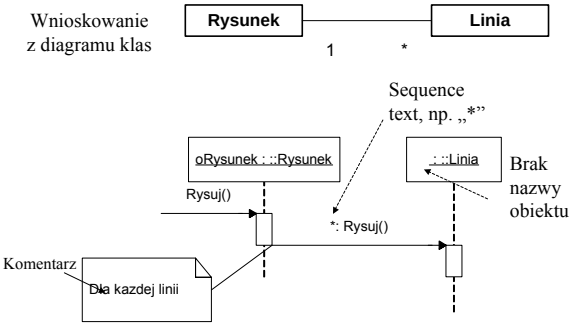



```
void CRysunek::Rysuj () {
  ...
  oLinia.Rysuj ();
  ...
}
```

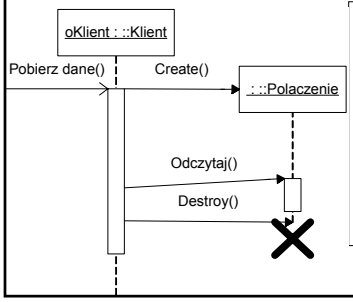
Dla powiązanych obiektów w C++


```
void CRysunek::Rysuj () {
  ...
  rLinia->Rysuj ();
  ...
}
```


Czy wywołanie w pętli?

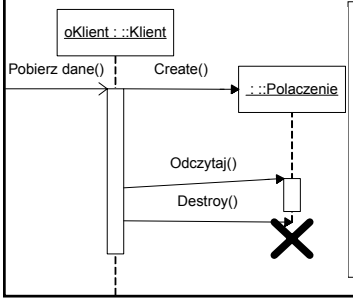


Tworzenie i usuwanie obiektów w C++




```
void CKlient::PobierzDane ()
{
 ...
 oPolaczenie = new
   CPolaczenie ();
 ...
 oPolaczenie->Odczytaj (...);
 ...
 delete oPolaczenie;
 ...
}
```

Tworzenie i usuwanie obiektów w Javie



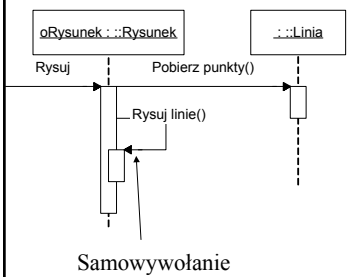

```
public class Klient {
...
 void pobierzDane () {
  ...
  Polaczenie polaczenie =
    new Polaczenie ();
  ...
  polaczenie.odczytaj (...);
  ...
 }
...
}
```

Dostęp do pól

- Operacje:
  - Pobierz dane / Get data
  - Ustaw dane / Set data
  - Pobierz pole / Get field
  - Ustaw pole / Set field

Mogą być implementowane jako odczyt/zapis pól

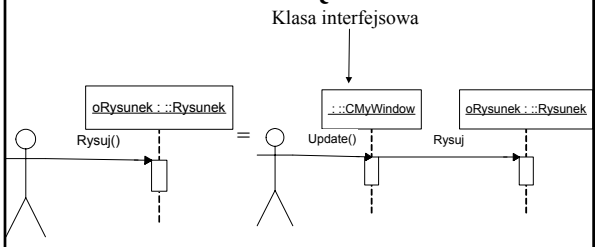
Dostęp do pól i samowywołanie



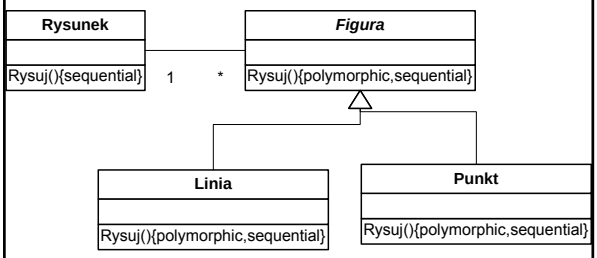

```
void CRysunek::Rysuj ()
{
  ...
  RysujLinie
   (rLinia->Punkty);
  ...
}
```

Pobranie danych

Wywołania pochodzące z zewnątrz



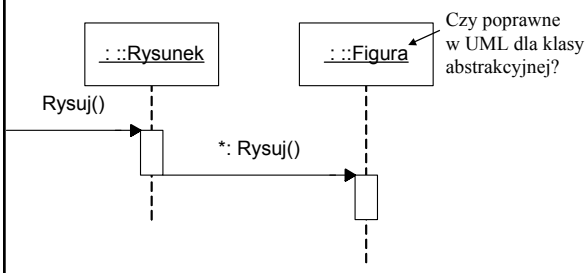
Operacje wirtualne (polimorficzne)



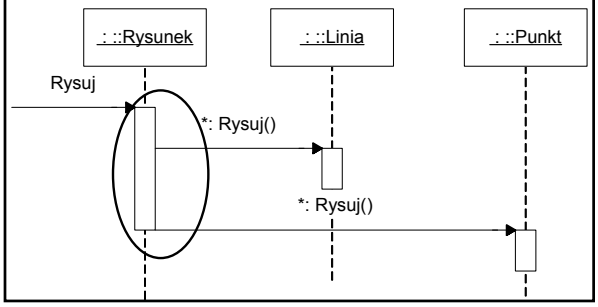
Operacje wirtualne

- W Javie domyślne
- W C++
  - `virtual void Rysuj ();`

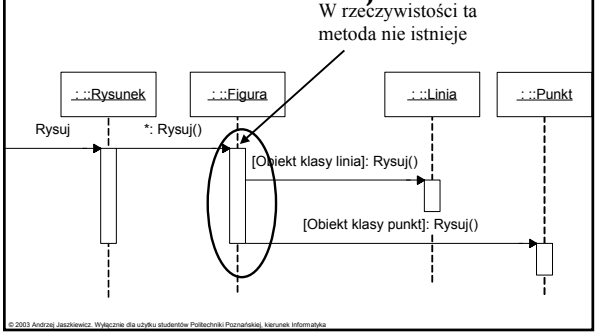
Punkt widzenia klasy Rysunek



Rzeczywiste wywołania metod dla obiektów

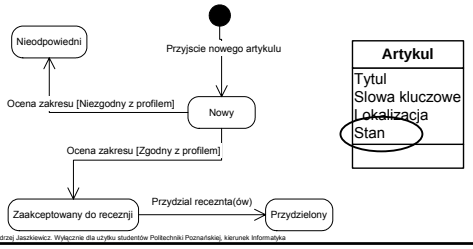


Ilustracja efektu operacji wirtualnej



Diagramy stanów

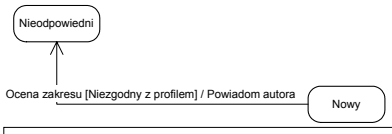
- Realizacja klasyczna – pole przechowujące stan i odpowiednia reakcja metod



Zmiany stanów w metodach

```
void CArtykul::OcenaZakresu (TZakres
Zakres) {
  if (Stan == _NOWY) {
    if (Zakres == _ZGODNY)
      Stan = _ZAAKCEPTOWANY_DO_RECENZJI;
    else
      Stan = _NIEODPOWIEDNI;
  }
  else
    // Niepoprawne
    ...
}
```


Akcje/operacje -> metody (fragmenty metod)



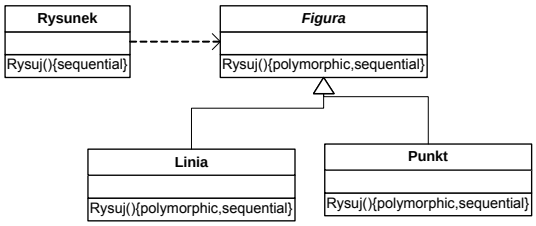

```
...
if (Zakres == _ZGODNY)
    Stan = _ZAAKCEPTOWANY_DO_RECENZJI;
  else {
    Stan = _NIEODPOWIEDNI;
    PowiadomAutora ();
  }
}
```


Akcje/operacje -> metody (fragmenty metod)


```
void CArtykul::Monitoruj () {
  if (Stan == _U_RECENZENTA) {
    ...
  }
}
```

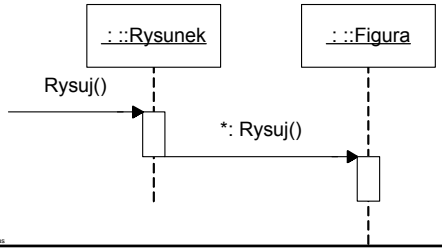

Związek zależności (*dependency*)

- Zmiana jednego elementu może powodować zmianę drugiego

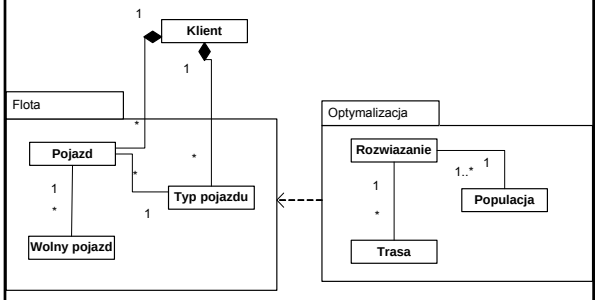


Związek zależności (*dependency*)

- Często oznacza uogólnienie (być może wielu) wywołań metod i dostępów do pól



Pakiety



Diagramy wdrożenia – deployment diagrams