Programowanie obiektowe

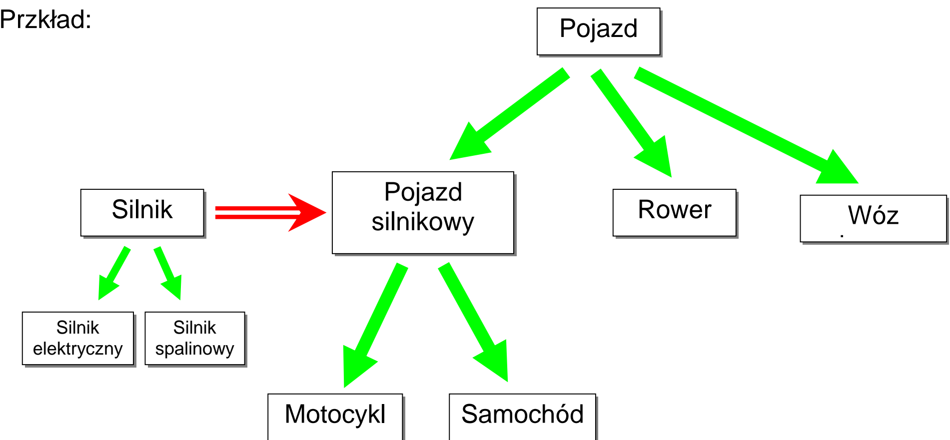
Kompozycja i dziedziczenie klas

Paweł Rogaliński
Instytut Informatyki, Automatyki i Robotyki
Politechniki Wrocławskiej

pawel.rogalinski@pwr.wroc.pl

Związki między klasami: „jest" i „zawiera"

Przkład:



Pojazd silnikowy jest szczególnym rodzajem **Pojazdu**

Motocykl jest szczególnym rodzajem **Pojazdu silnikowego**

Pojazd silnikowy zawiera **Silnik**

Ponowne wykorzystanie klas

Przy tworzeniu nowych klas można wykorzystywać już istniejące klasy za pomocą:

- **kompozycji**,
- **dziedziczenia**.

Kompozycje stosuje się wtedy, gdy między klasami zachodzi relacja typu „**całość ↔ część**" tzn. nowa klasa **zawiera** w sobie istniejącą klasę.

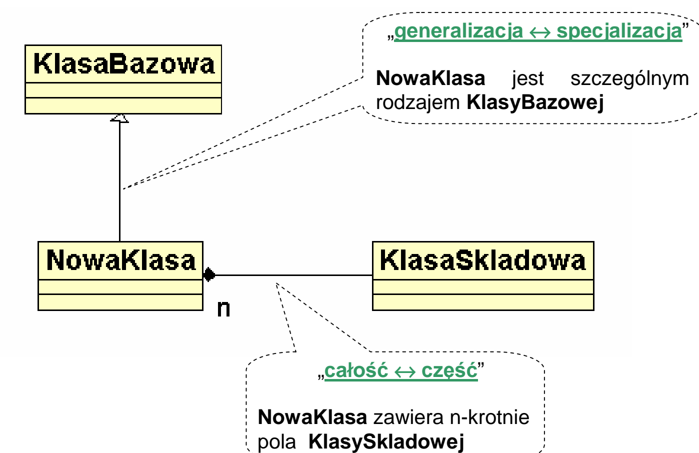
Dziedziczenie stosuje się wtedy, gdy między klasami zachodzi relacja „**generalizacja ↔ specjalizacja**" tzn. nowa klasa **jest szczególnym rodzajem** juz istniejącej klasy.

Uwaga:
Zwykle tworzy się nowe klasy wykorzystując jednocześnie kompozycję i dziedziczenie np.:
klasa **Pojazd silnikowy** jest uszczegółowieniem klasy **Pojazd** oraz zawiera w sobie klasę **Silnik**.

Diagramy klas w języku UML

UML (ang. Unified Modeling Language) – zunifikowany język modelowania do tworzenia systemów obiektowo zorientowanych.

Diagram klas pokazuje klasy i zachodzące między nimi relacje.



Kompozycja

Kompozycję uzyskujemy poprzez definiowanie w nowej klasie pól, które są obiektami istniejących klas.

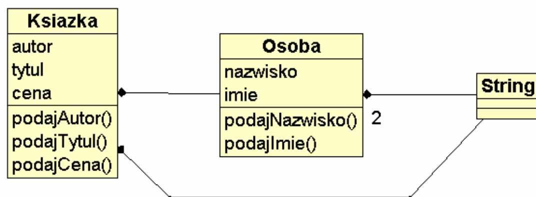
Przykład:

Klasa `Osoba` zawiera:

- pola `nazwisko` i `imie`, które należą do klasy `String`.

Klasa `Książka` zawiera:

- pole `autor` należące do klasy `osoba`,
- pole `tytul` należące do klasy `String`,
- pole `cena` typu `double`.



Kompozycja cd.

Definicja klasy `Osoba`

```
class Osoba
{ private String nazwisko;
  private String imie;

  public Osoba(String nazwisko, String imie)
  { this.nazwisko = nazwisko;
    this.imie = imie;
  }

  public String podajNazwisko()
  { return nazwisko;
  }

  public String podajImie()
  { return imie;
  }
}
```


Kompozycja cd.

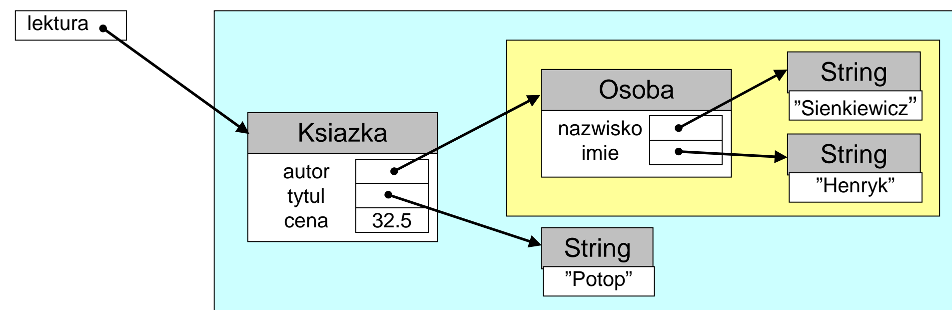
Definicja klasy `Ksiazka`

```
class Ksiazka
{ private Osoba autor;
  private String tytul;
  double cena;

  public Ksiazka(Osoba autor, String tytul, double cena)
  { this.autor = autor;
    this.tytul = tytul;
    this.cena = cena;
  }

  public Osoba podajAutor()
  { return autor;
  }

  public String podajTytul()
  { return tytul;
  }

  public double podajCena()
  { return cena;
  }
}
```


Kompozycja cd.

Przykładowa instrukcja tworząca nowy obiekt klasy `Ksiazka`:

```
Ksiazka lektura = new Ksiazka( new Osoba("Sienkiewicz", "Henryk"),
                               "Potop",
                               32.5 );
```

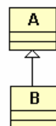


Dziedziczenie

Dziedziczenie polega na przejęciu właściwości i funkcjonalności obiektów innej klasy i ewentualnej modyfikacji tych właściwości i funkcjonalności w taki sposób, by były one bardziej wyspecjalizowane.

Do wyrażania relacji dziedziczenia jednej klasy przez drugą służy słowo kluczowe `extends`

```
class B extends A
{
   ...
}
```



Klasa `B` **dziedziczy** (rozszerza) klasę `A`, tzn.

- klasa `A` jest **klasą bazową**, (superklasą) klasy `B`
- klasa `B` jest **klasą pochodną** klasy `A`

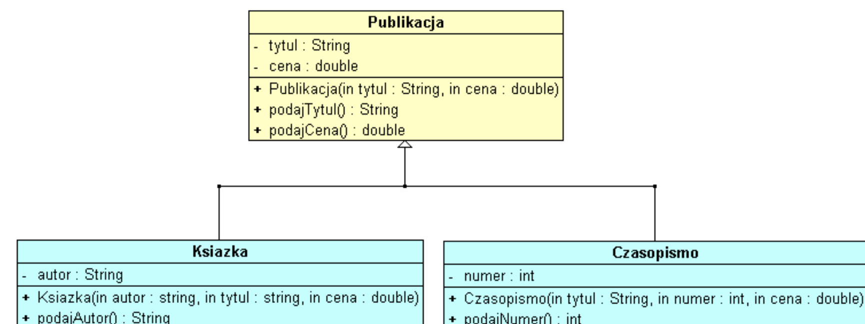
Dziedziczenie cd.

Przykład:

Klasa `Publikacja` zawiera:
- pole `tytul` z klasy `String` i pole `cena` typu `double`.

Klasa `Książka` dziedziczy po klasie `Publikacja` i dodatkowo zawiera:
- pole `autor` należące do klasy `String`.

Klasa `Czasopismo` dziedziczy po klasie `Publikacja` i dodatkowo zawiera:
- pole `numer` typu `int`.



Dziedziczenie cd.

Definicja klasy bazowej `Publikacja`

```
class Publikacja
{
    private String tytul;
    private double cena;

    Publikacja(String tytul, double cena)
    { this.tytul = tytul;
      this.cena = cena;
    }


    public String podajTytul()
    { return tytul;
    }

    public double podajCene()
    { return cena;
    }
}
```


Dziedziczenie cd.

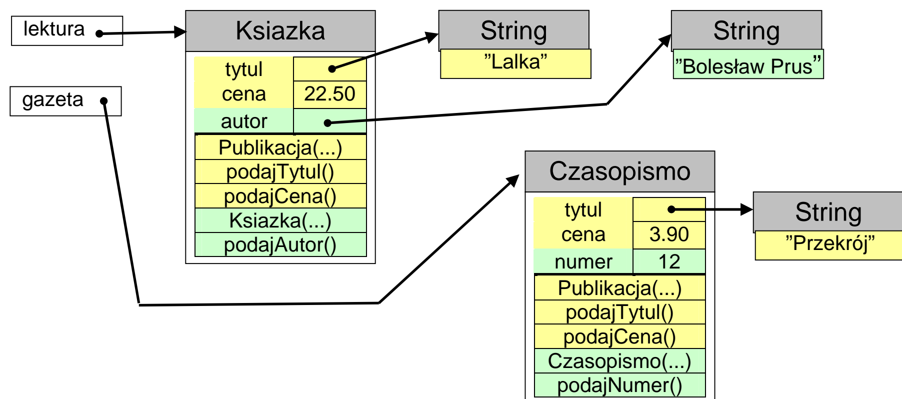
Definicja klas pochodnych `Ksiazka` i `Czasopismo`, które dziedziczą po klasie `Publikacja`

```
class Ksiazka extends Publikacja
{
    private String autor;

    Ksiazka(String autor, String tytul, double cena)
    { super(tytul, cena);    // Wywołanie konstruktora klasy bazowej Publikacja
      this.autor = autor;
    }

    public String podajAutor()
    { return autor;
    }
}

class Czasopismo extends Publikacja
{
    private int numer;

    Czasopismo(String tytul, int numer, double cena)
    { super(tytul, cena);    // Wywołanie konstruktora klasy bazowej Publikacja
      this.numer = numer;
    }

    public int podajNumer()
    { return numer;
    }
}
```

Dziedziczenie cd.

Przykładowe instrukcje tworzące nowe obiekty klas `Ksiazka` i `Czasopismo`:

```
Ksiazka lektura = new Ksiazka("Bolesław Prus", "Lalka", 22.50 );

Czasopismo gazeta = new Czasopismo("Przektój", 12, 3.90 );
```



Inicjowanie obiektów przy dziedziczeniu

Pola klasy bazowej można inicjować za pomocą wywołania z konstruktora klasy pochodnej konstruktora klasy bazowej:

```
super(args);
```

gdzie *args* jest listą argumentów przekazanych do konstruktora klasy bazowej

> Jeśli konstrukcja `super(...)` występuje, **MUSI** być pierwszą instrukcją konstruktora klasy pochodnej.
>
> Gdy nie występuje, przed wykonaniem kodu klasy pochodnej zostanie wywołany konstruktor bezparametrowy klasy bazowej.

Przykład:

```
Czasopismo(String tytul, int numer, double cena)
{ super(tytul, cena);
  this.numer = numer;
}
```

Wywołanie konstruktora klasy bazowej

Inicjowanie obiektów przy dziedziczeniu cd.

Przy tworzeniu obiektów klas pochodnych podstawową regułą jest, że **najpierw muszą być zainicjowane pola odziedziczone z klasy bazowej, potem dodatkowe pola deklarowane w klasie pochodnej**.

Sekwencja inicjowania obiektu klasy pochodnej jest następująca:

1. Wywoływany jest konstruktor klasy pochodnej,
2. Jeśli pierwszą instrukcją jest `super(args)`, wykonywany jest konstruktor klasy bazowej z argumentami *args*.
3. Jeśli nie ma `super(args)`, wykonywany jest konstruktor bezparametrowy klasy bazowej.
4. Wykonywane są instrukcje wywoływanego konstruktora klasy pochodnej.

Inicjowanie obiektów przy dziedziczeniu cd.

```
class A
{
    A()
    { System.out.println("  Konstruktor bezparametrowy klasy A");
    }

    A(String t)
    { System.out.println("  Konstruktor klasy A z parametrem String " + t);
    }
}

class B extends A
{
    B()
    { System.out.println("  Konstruktor bezparametrowy klasy B");
    }

    B(int i)
    { System.out.println("  Konstruktor klasy B z parametrem int " + i);
    }

    B(String t)
    { super(t);
      System.out.println("  Konstruktor klasy B z parametrem String " + t);
    }
}

class C extends B
{
}
```

Inicjowanie obiektów przy dziedziczeniu cd.

```
class KlasyABC
{
    public static void main(String [] args)
    {
        System.out.println("Tworzenie obiektu B -> new B();");
        new B();

        System.out.println("\nTworzenie obiektu B -> new B(int);");
        new B(1);

        System.out.println("\nTworzenie obiektu B -> new B(String);");
        new B("Ala");

        System.out.println("\nTworzenie obiektu C -> new C();");
        new C();
    }
}
```

Wiersz polecenia

```
C:\Testjava>java KlasyABC
Tworzenie obiektu B -> new B();
  Konstruktor bezparametrowy klasy A
  Konstruktor bezparametrowy klasy B

Tworzenie obiektu B -> new B(int);
  Konstruktor bezparametrowy klasy A
  Konstruktor klasy B z parametrem int 1

Tworzenie obiektu B -> new B(String);
  Konstruktor klasy A z parametrem Strin
  Konstruktor klasy B z parametrem Strin

Tworzenie obiektu C -> new C();
  Konstruktor bezparametrowy klasy A
  Konstruktor bezparametrowy klasy B

C:\Testjava>_
```


Przeciążanie metod

Przeciążanie metod w klasie polega na definiowaniu wielu metod o tej samej nazwie ale odmiennej liście parametrów. Przeciążane mogą być zarówno konstruktory jak i metody statyczne i niestatyczne.

Przykład:

```
class Para
{ int a, b;

   Para()
   { System.out.println(" Konstruktor bezparametrowy");
     a = b = 0;
   }

   Para(int x, int y)
   { System.out.println(" Konstruktor z parametrami");
     a = x;  b = y;
   }

   void pokaz()
   { System.out.println("Metoda bez parametru");
     System.out.println("Para("+ a +", "+ b +")" );
   }

   void pokaz(String tekst)
   { System.out.println("Metoda z parametrem");
     System.out.print(tekst);
     pokaz();
   }
```

Przeciążone konstruktory

Przeciążone metody

Wywołanie metody przeciążonej

Przeciążanie metod cd.

```
   public static void main(String[] args)
   {
    System.out.println("<=== Tworzenie obiektow ===>");
    Para p1 = new Para();
    Para p2 = new Para(3,7);


    System.out.println("\n<=== Metoda pokaz bez parametru ===>");
    p2.pokaz();

    System.out.println("\n<=== Metoda pokaz z parametrem ===>");
    p2.pokaz("Para p2: ");

   }
}
```

Wiersz polecenia

```
C:\Testjava>java Para
<=== Tworzenie obiektow ===>
 Konstruktor bezparametrowy
 Konstruktor z parametrami

<=== Metoda pokaz bez parametrow ===>
Metoda bez parametrow
Para(3, 7)

<=== Metoda pokaz z parametrem ===>
Metoda z parametrem
Para p2: Metoda bez parametrow
Para(3, 7)

C:\Testjava>
```


Przedefiniowanie metody

Przedefiniowanie metody klasy bazowej w klasie pochodnej następuje wtedy, gdy w klasie pochodnej zdefiniujemy metodę z taką samą sygnaturą (nazwa i lista parametrów) i typem wyniku jak sygnatura i typ wyniku nieprywatnej i niestatycznej metody klasy bazowej.

Wówczas metoda klasy bazowej zostaje ukryta.

Przykład:

```
class Para
{ int a, b;

   Para(int x, int y)
   { System.out.println(" Konstruktor z parametrami");
     a = x;  b = y;
   }

   void pokaz()
   { System.out.println("Metoda bez parametru z klasy Para");
     System.out.println("Para("+ a +", "+ b +")" );
   }

}
```

Przedefiniowanie metody cd.

```
class Trojka extends Para
{ int c;

  Trojka(int x, int y, int z)
  { super(x, y);
    c = z;
  }

  void pokaz()
  { System.out.println("Metoda bez parametru z klasy Trojka");
    System.out.println("Trojka("+ a +", "+ b +", "+ c +")" );
  }

  void pokaz(String tekst)
  { System.out.println("Metoda z parametrem z klasy Trojka");
    System.out.println(tekst);
    pokaz();
    super.pokaz();
  }

  public static void main(String[] args)
  {
    System.out.println("<=== Tworzenie obiektow ===>");
    Trojka t1 = new Trojka(1, 2, 3);

    System.out.println("\n<=== Metoda pokaz bez parametru ===>");
    t1.pokaz();

    System.out.println("\n<=== Metoda pokaz z parametrem ===>");
    t1.pokaz("Trojka t1: ");
   }
}
```

Wywołanie konstruktora klasy bazowej

Przedefiniowanie metody z klasy bazowej

Przeciążenie metody

Wywołanie metody przedefiniowanej

Wywołanie metody z klasy bazowej

Przedefiniowanie metody cd.

Wiersz polecenia

```
C:\Testjava>java Trojka
<=== Tworzenie obiektow ===>
 Konstruktor z parametrami

<=== Metoda pokaz bez parametru ===>
Metoda bez parametru z klasy Trojka
Trojka(1, 2, 3)

<=== Metoda pokaz z parametrem ===>
Metoda z parametrem z klasy Trojka
Trojka t1:
Metoda bez parametru z klasy Trojka
Trojka(1, 2, 3)
Metoda bez parametru z klasy Para
Para(1, 2)

C:\Testjava>
```


Pokrycie metody statycznej

Pokrycie metody statycznej klasy bazowej w klasie pochodnej następuje wtedy, gdy w klasie pochodnej zdefiniujemy statyczną metodę z taką samą sygnaturą (nazwa i lista parametrów) i typem wyniku jak sygnatura i typ wyniku nieprywatnej i statycznej metody klasy bazowej.

Wówczas metoda klasy bazowej zostaje ukryta. Można się do niej odwołać z innej metody poprzedzając wywołanie metody nazwą klasy bazowej i operatorem „**.**" .

Uwaga:

W podobny sposób można pokrywać pola klasy bazowej. Należy w klasie pochodnej zdefiniować pole o tej samej nazwie (może być innego typu).

Pokrycie metody statycznej cd.

```
class KlasaBazowa
{
  static void drukuj()
  { System.out.println("Metoda statyczna klasy bazowej");
  }
}

class KlasaPochodna extends KlasaBazowa
{
  static void drukuj()
  { System.out.println("Metoda statyczna klasy pochodnej");
  }

  public static void main(String[] args)
  {
    drukuj();

    KlasaBazowa.drukuj();
  }
}
```

Pokrycie metody statycznej z klasy bazowej

Wywołanie metody statycznej z klasy pochodnej

Wywołanie metody statycznej z klasy bazowej

Przedefiniowanie i pokrycie - podsumowanie

- Metody prywatne nie mogą być przedefiniowane ani pokryte, gdyż nie są dostępne w klasie pochodnej.
- Metoda nieprywatna i niestatyczna jest przedefiniowana w klasie pochodnej, jeśli ma taką samą sygnaturę jak metoda klasy bazowej.
- Typ wyniku metody przedefiniowującej w klasie pochodnej musi być taki sam jak typ wyniku metody przedefiniowanej z klasy bazowej.
- Przy przedefiniowaniu można za pomocą specyfikatorów dostępu rozszerzać dostęp, ale nie można go zawężać.
- Metody statyczne nie mogą być przedefiniowane ale mogą być pokryte.
- Przy przedefiniowaniu i pokryciu metod trzeba zachować zgodność typów wyjątków zgłaszanych przez metodę i deklarowanych w klauzuli **`throws`**.

Hierarchia dziedziczenia w Javie

Jeśli przy definicji klasy nie używamy słowa **`extends`** (tzn. nie żądamy jawnie dziedziczenia) to nasza klasa domyślnie dziedziczy klasę **`Object`**.

W Javie każda klasa może bezpośrednio odziedziczyć tylko jedną klasę. Ale pośrednio może mieć dowolnie wiele nadklas.

W Javie wszystkie klasy pochodzą pośrednio od klasy `Object`

```
class A
{
}

class B extends A
{
}

class C extends A
{
}

class D extends B
{
}

class E extends E
{
}
```