Programowanie obiektowe

Polimorfizm, metody wirtualne i klasy abstrakcyjne

Paweł Rogaliński
Instytut Informatyki, Automatyki i Robotyki
Politechniki Wrocławskiej

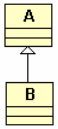
pawel.rogalinski @ pwr.wroc.pl

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**

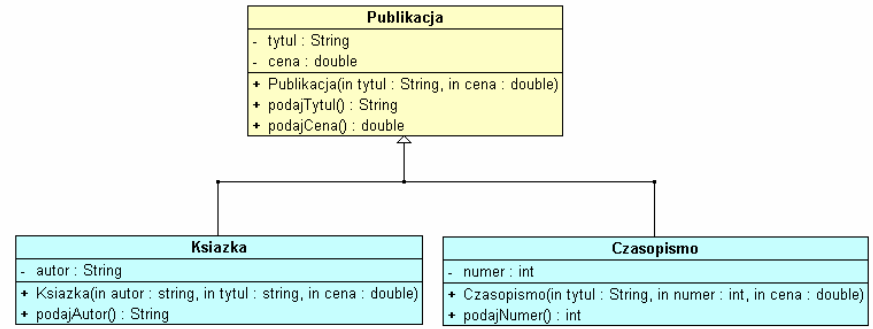
Dziedziczenie cd.

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.

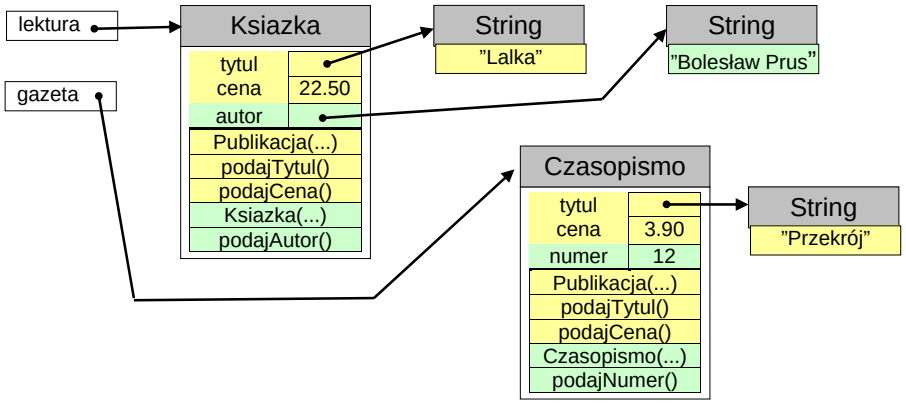
Definicja klas pochodnych **Ksiazka** i **Czasopismo**, które dziedziczą po klasie **Publikacja**

```
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(”Przekrój”, 12, 3.90 );
```



Konwersje referencyjne

Można zauważyć, że obiekt klasy pochodnej posiada wszystkie atrybuty i metody klasy bazowej, a więc „zawiera w sobie” obiekt klasy bazowej (nadklasy). Dlatego odniesienie do takiego obiektu można zapamiętać w zmiennej referencyjnej klasy bazowej.

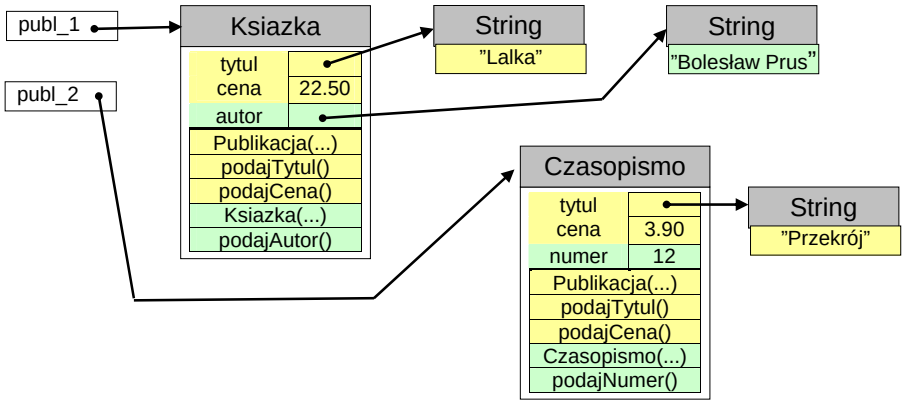
Obiekty klasy **Ksiazka** i klasy **Czasopismo** mają właściwości obiektów klasy **Publikacja** (tzn. posiadają wszystkie atrybuty i metody klasy **Publikacja**).

Referencje do obiektów klas **Ksiazka** i **Czasopismo** można więc przypisywać do zmiennych referencyjnych klasy **Publikacja**.

Konwersje referencyjne cd.

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

```
Publikacja publ_1 = new Ksiazka(”Bolesław Prus”,”Lalka”, 22.50 );

Publikacja publ_2 = new Czasopismo(”Przekrój”, 12, 3.90 );
```



Referencyjna konwersja rozszerzająca

Referencyjna konwersja rozszerzająca to przekształcenie referencji do obiektu klasy pochodnej na referencję do typu wyższego czyli nadklasy (klasy bazowej).

Referencyjna konwersja rozszerzająca jest dokonywana automatycznie przy:

- przypisywaniu zmiennej referencyjnej odniesienia do obiektu klasy pochodnej,
- przekazywaniu argumentów metodzie, gdy parametr metody jest typu „referencja do obiektu nadklasy" przekazywanego argumentu
- zwracaniu wyniku metody, gdy wynik podstawiamy na zmienną będącą „referencją do obiektu nadklasy" zwracanego wyniku.

Referencyjna konwersja rozszerzająca

```
class Wydawnictwo
{
    static Publikacja utworzPublikacje(int rodzaj)
    {
        switch(rodzaj)
        { case 1: // utworz ksiazke
                  return new Ksiazka("Barteczko", "Java", 45.00);
          case 2: // utworz czasopismo
                  return new Czasopismo("PC Format", 11, 9.50);
        }
        return null;
    }

    static double roznicaCeny(Publikacja p1, Publikacja p2 )
    {    return p1.podajCene() - p2.podajCene();
    }

    public static void main(String [] args)
    {
        Publikacja p1 = utworzPublikacje(1);
        Publikacja p2 = utworzPublikacje(2);
        roznicaCeny(p1, p2);

        Ksiazka k = new Ksiazka("Barteczko", "Java", 45.00);
        Czasopismo c = new Czasopismo("PC Format", 11, 9.50);
        roznicaCeny(k, c);

        p1 = k;
        p2 = c;
        roznicaCeny(p1, p2);
    }
}
```

Konwersja referencyjna przy zwracaniu wyniku

Konwersja referencyjna przy przekazywaniu argumentow

Konwersja referencyjna przy przypisywaniu

Referencyjna konwersja zawężająca

Referencyjna konwersja zawężająca to przekształcenie referencji klasy bazowej na referencję do typu niższego czyli podklasy (klasy pochodnej). Taka konwersja jest dozwolona tylko wtedy, gdy referencja klasy bazowej wskazuje na obiekt, który w rzeczywistości należy do klasy pochodnej.

Referencyjna konwersja zawężająca (konwersja „w dół") :

- zawsze wymaga jawnego użycia operatora konwersji,
- jest bezpieczna. Java w trakcie wykonywania programu sprawdza czy obiekt, na który wskazuje referencja, jest faktycznie obiektem należącym do klasy pochodnej. Gdy tak nie jest to zostanie zgłoszony wyjątek **`ClassCastException`**.

Referencyjna konwersja zawężająca

```
class Wydawnictwo
{
    static Publikacja utworzPublikacje(int rodzaj)
    {   switch(rodzaj)
        { case 1: // utworz ksiazke
                  return new Ksiazka("Barteczko", "Java", 45.00);
          case 2: // utworz czasopismo
                  return new Czasopismo("PC Format", 11, 9.50);
        }
        return null;
    }

    public static void main(String [] args)
    {
        Publikacja publ_1, publ_2;
        String autor_1, autor_2;
        Ksiazka ksiaz;

        publ_1 = utworzPublikacje(1); // utworzenie obiektu klasy Ksiazka
        publ_2 = utworzPublikacje(2); // utworzenie obiektu klasy Publikacja

        ksiaz = (Ksiazka)publ_1;
        autor_1 = ksiaz.podajAutor();

        ksiaz = (Ksiazka)publ_2;
        autor_2 = ksiaz.podajAutor();

        autor_1 = ((Ksiazka)publ_1).podajAutor();

        autor_2 = ((Ksiazka)publ_2).podajAutor();
    }
}
```

Zawężająca konwersja referencyjna

Tu zostanie zgłoszony wyjątek ClassCastException

Operator `instanceof`

Operator **`instanceof`** jest wykorzystywany do stwierdzenia , do jakiej klasy należy obiekt. Wyrażenie:

`nazwaZmiennej` **`instanceof`** ***`nazwaKlasy`***

ma wartość **`true`**, jeśli zmienna ***`nazwaZmiennej`*** wskazuje na obiekt należący do klasy ***`nazwaKlasy`***, albo dowolnej jej podklasy.

Operator `instanceof`

```
public static void main(String [] args)
    {
        Publikacja publ_1, publ_2;
        String autor_1, autor_2;
        Ksiazka ksiaz;

        publ_1 = utworzPublikacje(1); // utworzenie obiektu klasy Ksiazka
        publ_2 = utworzPublikacje(2); // utworzenie obiektu klasy Publikacja

        if (publ_1 instanceof Ksiazka)
           {  ksiaz = (Ksiazka)publ_1;
              autor_1 = ksiaz.podajAutor();
           }

        if (publ_2 instanceof Ksiazka)
           {  ksiaz = (Ksiazka)publ_2;
              autor_2 = ksiaz.podajAutor();
           }

        if (publ_1 instanceof Ksiazka)
           {  autor_1 = ((Ksiazka)publ_1).podajAutor();
           }

        if (publ_2 instanceof Ksiazka)
           {  autor_2 = ((Ksiazka)publ_2).podajAutor();
           }
    }
```

Użycie operatora **`instanceof`** zabezpiecza przed próbą wykonania niedozwolonej konwersji zawężającej

Metody wirtualne

Jeśli w podklasie (klasie pochodnej) zostanie przedefiniowana jakaś metoda, zdefiniowana pierwotnie w nadklasie (klasie bazowej) to przy wywołaniu tej metody zostanie uruchomiona metoda tej klasy, do której faktycznie należy obiekt, a nie tej klasy która jest typem zmiennej referencyjnej zawierającej odniesienie do obiektu.

Oznacza to, że wiązanie odwołań do metod z kodem programu następuje nie w czasie kompilacji programu, lecz fazie wykonania programu tuż przed każdorazowym wykonaniem instrukcji wywołującej przedefiniowaną metodę.

Metody wirtualne to takie metody, dla których wiązanie odwołań z kodem programu następuje w fazie wykonania programu

Metody wirtualne cd.

Metody wirtualne to takie metody, dla których wiązanie odwołań z kodem programu następuje w fazie wykonania programu

W Javie wszystkie metody są wirtualne za wyjątkiem:

- **metod statycznych** (bo nie dotyczą obiektów, a klasy)
- **metod deklarowanych ze specyfikatorem `final`**, który oznacza, że metoda jest ostateczne i nie może być przedefiniowana,
- **metod prywatnych** (bo metody prywatne nie mogą zostać przedefiniowane).

Odwołania do metod wirtualnych są **polimorficzne**, gdyż efekt każdorazowego odwołania może przybierać różne kształty, w zależności od tego jaki jest faktyczny typ obiektu, na rzecz którego wywołano metodę wirtualną.

Metody wirtualne - przykład.

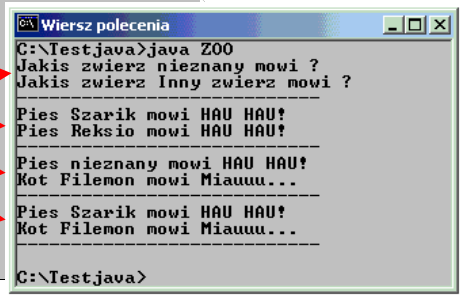
```
class Zwierz
{   String nazwa = "nieznany";

    Zwierz(){ }
    Zwierz(String n){ nazwa = n; }

    String podajGatunek() { return "Jakis zwierz"; }
    String podajNazwe()   { return nazwa; }
    String podajGlos()    { return "?"; }

    void mowa()
    {   System.out.println(podajGatunek() + " " +
                           podajNazwe() + " mowi " +
                           podajGlos() );
    }
}

class Pies extends Zwierz
{
    Pies(){ }
    Pies(String n){ super(n); }
    String podajGatunek() { return "Pies"; }
    String podajGlos()    { return "HAU HAU!"; }
}

class Kot extends Zwierz
{
    Kot() { }
    Kot(String n){ super(n); }
    String podajGatunek() { return "Kot"; }
    String podajGlos()    { return "Miauuu..."; }
}
```

Metody **podajGatunek(), podajNazwe(), podajGlos()** są wirtualne.

Działanie metod wirtualnych wywołanych w metodzie **mowa()** będzie zależeć od klasy obiektu, na rzecz którego zostanie wywołana metoda **mowa()**.

Metody wirtualne – przykład cd.

```
class ZOO
{
    static void dialogZwierzat(Zwierz z1, Zwierz z2)
    { z1.mowa();
      z2.mowa();
      System.out.println("---------------------------");
    }

    public static void main(String []args)
    {
          Zwierz z1 = new Zwierz(),
                 z2 = new Zwierz("Inny zwierz");
          Pies pies = new Pies(),
             szarik = new Pies("Szarik"),
             reksio = new Pies("Reksio");
        Kot filemon = new Kot("Filemon");

        dialogZwierzat(z1,     z2);

        dialogZwierzat(szarik, reksio);

        dialogZwierzat(pies,   filemon);

        dialogZwierzat(szarik, filemon);
    }

}
```



Metody i klasy abstrakcyjne

Metoda abstrakcyjna to metoda, która nie ma implementacji (ciała) i jest zadeklarowana ze specyfikatorem **abstract**. Taka metoda może być deklarowana tylko w klasie abstrakcyjnej !

```
abstract int obliczCos();
```

nie ma ciała – tylko średnik

Klasą abstrakcyjną to klasa, opatrzona specyfikatorem **abstract**. Taka klasa może (ale nie musi) zawierać metody abstrakcyjne.

```
abstract class JakasKlasa
{  abstract int obliczCos();
   void wypiszCos(){ System.out.printn("cos"); }
}
```

metoda abstrakcyjna

Nie można tworzyć obiektów klasy abstrakcyjnej !!!

Metody i klasy abstrakcyjne cd.

Klasa abstrakcyjna może być dziedziczona przez nowe klasy. Klasa pochodna **MUSI** przedefiniować (a właściwie zdefiniować) wszystkie metody abstrakcyjne, które odziedziczyła z abstrakcyjnej klasy bazowej. W przeciwnym wypadku klasa pochodna nadal pozostanie klasą abstrakcyjną i nie będzie można tworzyć jej obiektów.

Metody i klasy abstrakcyjne - przykład.

```
abstract class Zwierz
{   String nazwa = "nieznany";

    Zwierz(){ }
    Zwierz(String n){ nazwa = n; }

    String podajNazwe()   { return nazwa; }

    abstract String podajGatunek();
    abstract String podajGlos();

    void mowa()
    {   System.out.println(podajGatunek() + " " +
                           podajNazwe() + " mowi " +
                           podajGlos() );
    }
}

class Pies extends Zwierz
{
    Pies(){ }
    Pies(String n){ super(n); }
    String podajGatunek() { return "Pies"; }
    String podajGlos()    { return "HAU HAU!"; }
}

class Kot extends Zwierz
{
    Kot() { }
    Kot(String n){ super(n); }
    String podajGatunek() { return "Kot"; }
    String podajGlos()    { return "Miauuu..."; }
}
```

Metody **podajGatunek(), podajGlos()** są abstrakcyjne.

W metodzie **mowa()** są wywoływane metody abstrakcyjne, które nie zostały jeszcze zdefiniowane.

Przedefiniowanie (Konkretyzacja) metod abstrakcyjnych odziedziczonych z abstrakcyjnej klasy bazowej

Metody i klasy abstrakcyjne – przykład cd.

```
class ZOO
{
    static void dialogZwierzat(Zwierz z1, Zwierz z2)
    { z1.mowa();
      z2.mowa();
      System.out.println("----------------------------");
    }

    public static void main(String []args)
    {
       // Zwierz z1 = new Zwierz(),
       //        z2 = new Zwierz("Inny zwierz");

       Zwierz z1 = new Pies(),
              z2 = new Kot("Bonifacy");

         Pies pies = new Pies(),
            szarik = new Pies("Szarik"),
            reksio = new Pies("Reksio");
        Kot filemon = new Kot("Filemon");

        dialogZwierzat(z1,     z2);

        dialogZwierzat(szarik, reksio);

        dialogZwierzat(pies,   filemon);

        dialogZwierzat(szarik, filemon);
    }

}
```

Nie wolno tworzyć obiektów klasy abstrakcyjnej

Tu następują referencyjne konwersje rozszerzające.

```
Wiersz polecenia
C:\Testjava>java ZOO
Pies nieznany mowi HAU HAU!
Kot Bonifacy mowi Miauuu...
----------------------------
Pies Szarik mowi HAU HAU!
Pies Reksio mowi HAU HAU!
----------------------------
Pies nieznany mowi HAU HAU!
Kot Filemon mowi Miauuu...
----------------------------
Pies Szarik mowi HAU HAU!
Kot Filemon mowi Miauuu...
----------------------------

C:\Testjava>_
```