Programowanie obiektowe cz.2

Mechanizm dziedziczenia

- **Grupowanie obiektów w systemach obiektowych**
  - **W językach opisujących dziedzinę problemową przez prototypy (a nie klasy) występuje mechanizm delegacji.**
  - **Dziedziczenie grupuje obiekty zgodnie z ich typami.**
  - **Delegacja nie definiuje kategorii obiektów lecz umożliwia współdzielenie między dowolnymi obiektami.**
  - **System obiektowy oparty na dziedziczeniu oferuje klasy i obiekty.**
  - **System oparty na delegacji bazuje na jednym obiekcie łączącym w sobie cechy klasy i obiektu**

Mechanizm dziedziczenia

- **Delegacja:**
  - **Pozwala na stopniowe definiowanie obiektów w terminach innych obiektów.**
  - **Pozwala na współdzielenie funkcji i zmiennych.**
    - ***Delegacja atrybutów: funkcji lub zmiennych do innego prototypu, wymusza zmianę tych atrybutów, ich wartości (dotyczy zarówno obiektu i prototypu).***
    - ***W ten sposób obiekty w hierarchii delegacji mogą być wzajemnie zależne:***
      - ***ponieważ w systemie opartym na delegacji nie ma grupowania poprzez typy,***
      - ***dwa obiekty o różnych typach, mogą delegować do tego samego prototypu,***
      - ***podobnie, dwa obiekty tego samego typu mogą delegować do różnych prototypów (choć w rzeczywistości mogą dotyczyć całkowicie różnej części hierarchii).***

Mechanizm dziedziczenia

- **Różnica między klasycznym, (opartym na klasach) a prototypowym podejściem dotyczy:**
  - **wielowiekowej, filozoficznej debaty odnoszącej się do statusu i reprezentacji abstrakcji,**
  - **według Platona abstrakcja (ideał) istnieje bardziej rzeczywiście niż przedmiot w rzeczywistym świecie,**
- **Język obiektowy (Smalltalk) nawiązuje do tych idei w sensie:**
  - **bezpośredniego użycia klas do reprezentowania podobieństwa wśród kolekcji obiektów,**
- **Klasyczne i prototypowe podejście języków obiektowych różni się w sposobie reprezentowania i współdzielenia abstrakcji.**
- **Klasyczny język rozróżnia współdzielenie atrybutów klas przez:**
  - **obiekty**
  - **współdzielenie atrybutów nadklas przez klasy.**
  - **języki prototypowe posiadają jeden rodzaj współdzielenia - przez prototypy,**

Mechanizm dziedziczenia

- **Przykłady klas w systemach opartych na delegacji:**
  - **Aktor (ang. *actor).***
  - **Prototypy.**
- **Grupowanie poprzez typy jest użyteczne:**
  - **dla kompilacji,**
  - **indeksowania baz danych.**
- **Mały rozmiar obiektów w modelu delegacji zwiększa szybkość działania systemu.**
- **W systemie dziedziczenia opartym na klasach nowe obiekty tworzone są przez odwołanie cech klasy, do której należą.**
- **W modelu delegacji nowe obiekty - prototypy tworzone są poprzez klonowanie (ang. clonning) czyli kopiowanie.**

Mechanizm dziedziczenia

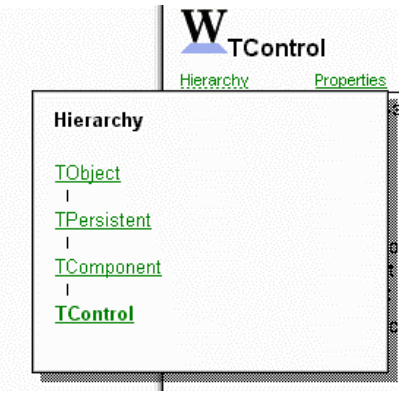
- **W systemie opartym na klasach żaden z obiektów nie jest samowystarczalny.**
  - **potrzebny jest inny obiekt (klasa) aby opisał jego strukturę i zachowanie.**
- **W systemie opartym na prototypach, obiekt (prototyp) zawiera swój własny stan behawioralny w związku z czym nie musi być zależny od innych obiektów.**
- **System prototypowy pozbawiony dziedziczenia prowadzi do problemu:**
  - **Mianowicie każdy obiekt (prototyp) powinien zawierać wszystkie swoje stany.**
- **Rozwiązaniem umożliwiającym współdzielenie stanów behawioralnych dla rodziny obiektów jest wykreowanie obiektu, który zawiera wspólne stany dla kolekcji obiektów, podobnie jak to czyni klasa, z wyjątkiem informacji dotyczących samej struktury obiektu.**
- **Kierunki doskonalenia systemu dotyczą głównie:**
  - **współbieżności,**
  - **przetwarzania rozproszonego.**

Mechanizm dziedziczenia

- **Zastosowanie mechanizmu dziedziczenia w przetwarzaniu rozproszonym prowadzi do niespójności:**
  - **pomiędzy rozproszeniem i dziedziczeniem - na skutek różnicy celów modularności i współdzielenia.**
    - **modularność wymaga separacji pomiędzy komponentami systemu,**
    - **współdzielenie wymaga fuzji komponentów.**
  - **dynamiczne współdzielenie wymaga fuzji komponentów podczas wykonywania programu i stoi w sprzeczności z rozproszeniem, które wymaga separacji w trakcie wykonywania zadania**

Mechanizm dziedziczenia

- **Cecha dziedziczenia zawiera dwa aspekty:**
  - **dziedziczenie strukturalne (podklasa danej klasy dziedziczy strukturę nadklasy),**



  - **dziedziczenie funkcjonalne polegające na dziedziczeniu funkcji występujących w nadklasie.**
    - **np. w klasie TControl funkcja UpdateAction wywodzi się z nadklasy TComponent**

Mechanizm dziedziczenia

- **Klasa obiektu opisuje swoją strukturę za pomocą zmiennych.**
  - **Zmienne podklasy muszą zachować ten sam typ informacji co zmienne nadklasy.**
  - **Zmienne w hierarchiach dziedziczenia są wzajemnie niezależne. Jeżeli typ podrzędny dziedziczy cechy typu nadrzędnego, to każdy obiekt typu podrzędnego automatycznie zawiera, jako część swojej reprezentacji, lokalne stany (zmienne) obiektu typu nadrzędnego.**

Mechanizm dziedziczenia

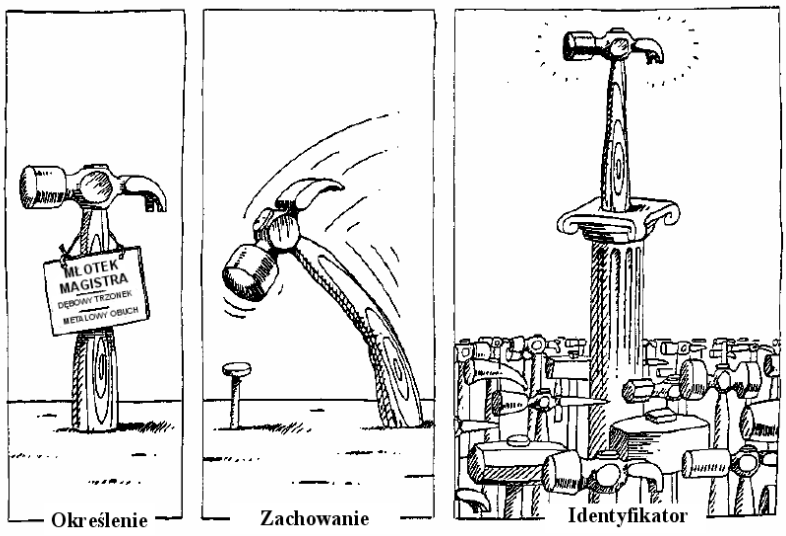
- **Podklasy nie mogą odwoływać się do zmiennych nadklasy bezpośrednio.**
  - **Zmienne nadklasy mogą być zmieniane jedynie za pomocą funkcji zdefiniowanych dla danej klasy.**
- **Bardziej ogólnym podejściem, które łączy efektywność i elastyczność jest wprowadzenie notacji:**
  - **public,**
  - **private,**
  - **visible,**
  - **pozostawienie konkretnej implementacji by zapewnić wymaganą ochronę danych.**

Mechanizm dziedziczenia

- **Klasa definiuje:**
  - **strukturę,**
  - **zachowanie obiektu,**
- **Zachowanie obiektu:**
  - **zachowanie obiektu jest określone za pomocą funkcji stowarzyszonych ze zmiennymi danej klasy,**
  - **funkcje są operacjami, za pomocą których dane mogą być pobierane i uaktualniane.**

Obiekt = Określenie + ID + Zachowanie

Przykładem obiektu może być tak zwany „Młotek Magistra”



Obiekt posiada zespół cech jednoznacznie go określających

Przykład obiektu i jego właściwości

- **Obiekt posiada trzy właściwości:**
  - **określenie - zestaw cech charakterystycznych**
  - **zachowanie - zestaw metod**
  - **identyfikator**

Przykładem obiektu może być plik dyskowy.

Przykłady „nie" obiektów

- **Atrybuty:**
  - **czas, kolor, uroda**
- **Emocje:**
  - **miłość, złość, nienawiść**
- **Byty będące normalnie obiektami, lecz w określonych przypadkach stają się atrybutami**

```
DzienTygodnia=Class
        Enum (Poniedziałek, Wtorek,
        Środa, Czwartek,
        Piątek,Sobota, Niedziela)
End;
```

```
Szkolenie=Class
        DzienTygodnia
End;
```

W definicji klasy bazowej DzienTygodnia dni są traktowane jako obiekty natomiast w klasie Szkolenie nazwa dnia tygodnia jest atrybutem klasy.

Przykłady dziedziczenia w klasach

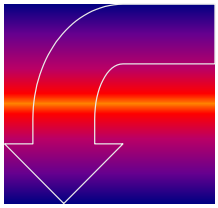
- **Podstawowa definicja typu GLOW**

```
type
  TGlow = class(TForm)
    ToolBar1: TToolBar;
    Panel1: TPanel;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    Image1: TImage;
    procedure MouseDown(Sender:TObject;Button:
    TMouseButton;Shift:TShiftState;X,Y:Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

Przykłady dziedziczenia w klasach

Instancja klasy TGlow w postaci aplikacji i jej elementów składowych.

Przykłady dziedziczenia w klasach



```
TGlow = class(TForm)
    ToolBar1: TToolBar;
    Panel1: TPanel;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    Image1: TImage;
    Button6: TButton;
    Button7: TButton;
    procedure Button5Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Powieksz(Sender: TObject);
    procedure Pomniejsz(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure Button6Click(Sender: TObject);
  private
    { Private declarations }
  public
    i:Integer;
    { Public declarations }
  end;
```

```
 T2Glow=Class(TGlow)
    procedure Button6Click(Sender:
TObject);
    procedure Button7Click(Sender:
TObject);
    procedure FormPaint(Sender: TObject);
  private
  End;
```

Przykłady dziedziczenia w klasach

```
procedure TGlow.Button4Click(Sender:
TObject);
begin
  Image1.Visible:=True;
  Button3.Enabled:=True;
  Button4.Enabled:=False;
end;

procedure TGlow.Button3Click(Sender:
TObject);
begin
  Image1.Visible:=False;
  Button4.Enabled:=True;
  Button3.Enabled:=False;
end;

procedure TGlow.FormCreate(Sender:
TObject);
begin
  Button4.Enabled:=False;
end;
```

```
Type
 Obrazek=Class(TImage)
    procedure Image1MouseDown(Sender:
TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    End;

var
  Glow: TGlow;
  Panel:TPanel;
  Glow2:T2Glow;
  Image:Obrazek;
```

Przykłady dziedziczenia w klasach

```
procedure TGlow.Powieksz(Sender: TObject);
begin
  If (Trunc(1.1*Image1.Width)<Panel1.Width) AND
 (Trunc(1.1*Image1.Height)<Panel1.Height) Then
    Begin
      Image1.Width:=Trunc(1.1*Image1.Width);
      Image1.Height:=Trunc(1.1*Image1.Height);
    End;
  FormPaint(Glow)
end;

procedure TGlow.Pomniejsz(Sender: TObject);
begin
  If (Trunc(Image1.Width/1.1)>30) AND (Trunc(Image1.Height/1.1)>30)
Then
    Begin
      Image1.Width:=Trunc(Image1.Width/1.1);
      Image1.Height:=Trunc(Image1.Height/1.1);
    End;
  FormPaint(Glow)
end;
```

```
procedure TGlow.FormPaint(Sender: TObject);
begin
    Image1.Left:=(Panel1.Width-Image1.Width) div 2;
    Image1.Top:=(Panel1.Height-Image1.Height) div 2;
end;
procedure T2Glow.FormPaint(Sender: TObject);
begin
  If Image<>Nil Then
    Begin
      Image.Left:=(Panel1.Width-Image1.Width) div 2;
      Image.Top:=(Panel1.Height-Image1.Height) div 2;
    End;
  Image1.Left:=(Panel1.Width-Image1.Width) div 2;
  Image1.Top:=(Panel1.Height-Image1.Height) div 2;
end;
```

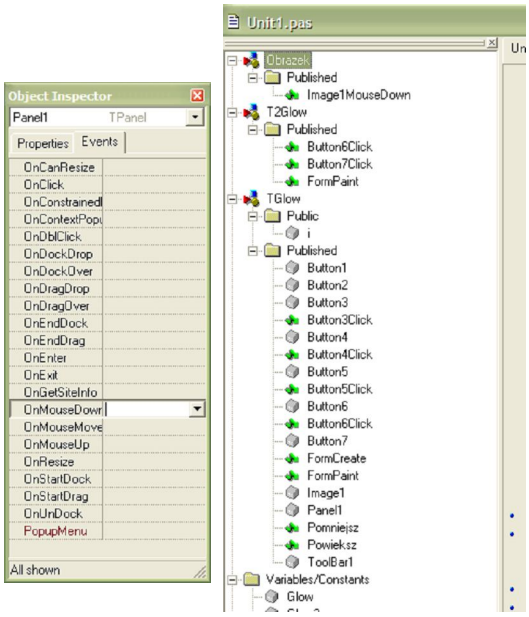
```
procedure T2Glow.Button6Click(Sender: TObject);
begin
  If Panel1.Visible Then
    Begin
      Panel1.Visible:=False;
      Button6.Caption:='Obiekty stat';
      Panel:=TPanel.Create(Nil);
      InsertControl(Panel);
      Panel.Align:=AlClient;
      Panel.Color:=ClMaroon;
      Button7.OnClick:=Button7Click;
    End
      Else
        Begin
          Panel1.Visible:=True;
          Button6.Caption:='Dziedziczenie';
          RemoveControl(Panel);
          Panel.Free
        End;
end;
```

```
procedure TGlow.Button6Click(Sender: TObject);
begin
  Glow2:=T2Glow.Create(Nil);
  Glow2.ShowModal;
  Glow2.i:=0;
end;
```

```
procedure T2Glow.Button7Click(Sender: TObject);
begin
  i:=i+1;
  If i<8 Then
    Begin
      Image:=Obrazek.Create(Nil);
      Image.Picture.LoadFromFile(IntToStr(i)+'.jpg');
      Image.AutoSize:=True;
      Image.OnMouseDown:=Image.Image1MouseDown;
      Panel.InsertControl(Image);
      FormPaint(Glow2)
    End;
end;
```

Przykłady dziedziczenia w klasach

```
procedure
Obrazek.Image1MouseDown(Sender:
TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y:
Integer);
begin
  If Button=MbLeft Then
    Begin
      Free;
    End;
end;
```



Wykorzystanie Inspektora Obiektów oraz Exploratora kodu do analizy struktury klas w omawianym przykładzie.

Polimorfizm

- **Z greckiego, polimorfizm - oznacza "wiele form" (postaci) jednego bytu.**
  - **Słowo "polimorfizm" też jest polimorficzne.**
- **Polimorfizm metod - operacja wywoływana przez komunikat może być różnie wykonana, w zależności od rodzaju obiektu, do którego ten komunikat został wysłany.**
- **Polimorfizm typów - polimorfizm w tzw. polimorficznych językach programowania - oznacza istnienie procedur lub funkcji, które mogą zarówno przyjmować wartości wielu typów jako swoje argumenty, jak też i zwracać wartości wielu typów.**
  - **Przykładowo, funkcja pobierz_pierwszy(lista) zwraca pierwszy element dowolnej listy, niezależnie od tego, czy jest to lista liczb całkowitych, czy lista liczb rzeczywistych, czy lista rekordów, czy też inna. Polimorfizm typów jest uważany za podstawę programowania ogólnego (generic).**

Polimorfizm

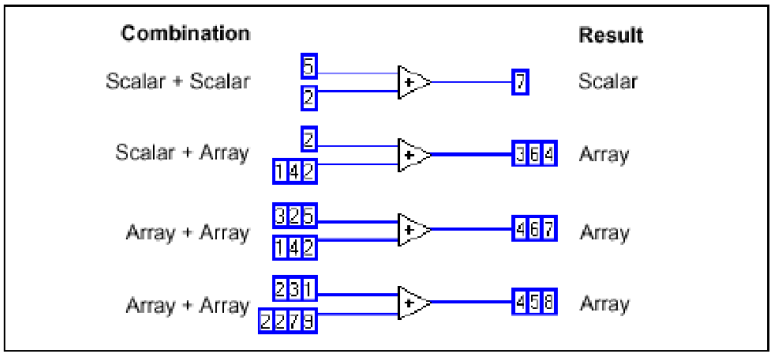
- **Temat pozostaje jednak w strefie rozważań akademickich, gdyż języki z polimorfizmem typów uważane są za zbyt wyrafinowane dla przeciętnego programisty.**
- **Powszechne mylone jest pojęcie polimorfizmu metod z polimorfizmem typów.**
- **Argumentacja:**
  - **polimorfizm metod jest szczególnym przypadkiem polimorfizmu typów (obiekt, do którego jest wysłany komunikat jest dodatkowym parametrem metody) nie jest zbyt przekonywująca.**

Polimorfizm

- **Polimorfizm parametryczny. Rodzaj polimorfizmu typów, który oznacza, że typ bytu programistycznego może być parametryzowany innym typem, np. klasa:**
  - **WEKTOR (int)**
  - **WEKTOR (char).**
- **Polimorficzne języki programowania (ML, Quest, Napier88, ...) nie muszą być obiektowe, ale mogą być obiektowe (Fibonacci).**

LabView a polimorfizm

- **W środowisku LabVIEW polimorfizm występuje w:**
  - **operacjach na tablicach,**
  - **operacjach porównań,**
  - **operacjach na łańcuchach,**
  - **funkcjach typu numeric.**
- **Typ wyjścia funkcji arytmetycznych jest taki sam jak typ wejścia. Dla funkcji z jednym wejściem, funkcje operują na każdym elemencie struktury.**
- **Dla funkcji z dwoma wejściami, należy użyć następujących wejściowych kombinacji:**
  - **wejścia podobne - obydwa wejścia mają takie same struktury, to wyjście też ma taką samą strukturę jak wejścia,**
  - **jedna wielkość skalarna - jedno wejście jest wielkością skalarną, a drugie jest tablicą lub grupą, to wyjście jest tablicą lub grupą,**
  - **tablica - jedno wejście jest tablicą typu numeric, drugie jest samo jest typu numeric, to wyjście jest tablicą.**



Podsumowanie

- **Obiekt - struktura danych, występująca łącznie z operacjami dozwolonymi do wykonywania na niej, odpowiadająca bytowi wyróżnialnemu w analizowanej rzeczywistości.**
- **Tożsamość obiektu - wewnętrzny identyfikator, który pozwala na odróżnienie go od innych obiektów.**
- **Hermetyzacja - rozróżnienie pomiędzy interfejsem do obiektu opisującym co obiekt robi, a implementacją definiującą, jak jest zbudowany i jak robi, to co ma zrobić.**
- **Dziedziczenie - Wielokrotne użycie tego, co wcześniej zostało zrobione: definiowanie klas, które mają wszystkie cechy zdefiniowane wcześniej (z nadklasy) plus cechy nowe.**
- **Polimorfizm - Wybór nazwy dla operacji jest określony wyłącznie semantyką operacji. Decyzja o tym, która metoda implementująca daną operację, zależy od przynależności obiektu do odpowiedniej klasy.**