ISSN 1899-8534

Kwartalnik Wojewódzkiego Ośrodka Doskonalenia Informatycznego i Politechnicznego

e WODIiP

Edukacja w drodze do społeczeństwa informacyjnego

Nr 2/2008

SPIS TREŚCI

http://kwartalnik.wodip.opole.pl

kwartalnik@wodip.opole.pl

mgr Lesław Tomczak

Wywiad z Profesorem Maciejem M. Sysło

LT: **Jak zmienia się na świecie edukacja „informatyczna" w szkole?**

MMS: Tradycyjnie przyjmuje się, że edukacja informatyczna to działania w szkole, związane z przygotowaniem młodzieży do korzystania z technologii informacyjno-komunikacyjnych (TIK), i również tradycyjnie dzieli się szkolne zajęcia na lekcje z wydzielonych przedmiotów informatycznych i na posługiwanie się tymi technologiami we wszystkich innych przedmiotach. Na świecie to zawsze wyglądało nieco inaczej, gdyż niewiele uwagi przykładano do wydzielonych zajęć. W ostatnich latach, w związku ze słabnącym zainteresowaniem (np. w USA i UK) studiowaniem na kierunkach ścisłych, w tym również na informatyce, bije się na alarm, że szkoły powinny lepiej przygotowywać uczniów do przyszłych wyborów kariery zawodowej. Jak zwykle, prawda jest gdzieś pośrodku. W związku z przesuwaniem się nacisku w kierunku podstaw informatyki dla wszystkich uczących się, pojawiło się pojęcie computational thinking, które oznacza głębsze zrozumienie metod komputerowych w rozwiązywaniu różnych problemów, nie tylko za pomocą komputerów, i umiejętności posługiwania się takimi metodami również z wykorzystaniem komputerów. Stwierdzono już bowiem, że lepsza znajomość podstaw informatyki i wyższe umiejętności posługiwania się TIK mają pozytywny wpływ na osiągnięcia w innych dziedzinach.

LT: **Jaką drogą my powinniśmy w tej edukacji podążać?**

MMS: Uważam, że droga, którą podąża edukacja informatyczna w Polsce jest właściwa. Jest oczywiście w niej miejsce na modyfikacje, które uwzględniają zarówno rozwój technologii, jak i metod nauczania z wykorzystaniem technologii (Cechą działalności intelektualnej jest stała modyfikacja drogi, która się podąża – Seymour Papert). Przygotowywane dokumenty rządowe zawierają już pewne propozycje zmian. W nowej podstawie programowej umacnia się pozycja zarówno wydzielonych zajęć informatycznych, jak i wykorzystania technologii w innych przedmiotach, chociaż w tym drugim obszarze panuje nadal bardzo tradycyjne myślenia. Znaczące zmiany i postęp obiecuję sobie po „Planie działań dotyczących nauczania dzieci i młodzieży w zakresie problematyki funkcjonowania w społeczeństwie informacyjnym. Nowe technologie w edukacji.", który ma być przyjęty przez rząd jako strategia działania w zakresie edukacji informatycznej na lata 2008-2015. Ten Plan to zmiana filozofii myślenia, wystarczy, że powiem iż dwoma najważniejszymi priorytetami strategicznymi w tym Planie są: personalizacja kształcenia i przygotowanie do uczenia się przez całe życia. Nie muszę dodawać, jak ważna jest rola technologii informacyjno-komunikacyjnych w realizacji obu tych priorytetów. Inne priorytety są „bardziej informatyczne".

(Ciąg dalszy na stronie 3)

Wywiad z Profesorem Maciejem M. Sysło

(Ciąg dalszy ze strony 2)

LT: **Czy projekty typu e-szkoła, nie zburzą porządku szkolnego? Są zagrożeniem czy raczej szansą?**

MMS: We wspomnianym Planie, e-szkoła jest określeniem szkoły, która realizuje priorytety strategiczne stosując w tym celu najnowsze technologie. A zatem, takie projekty, jeśli tylko wpisują się w działania Państwa i uwzględniają technologie dla poprawy poziomu kształcenia, rozumianego jako wszechstronny rozwój obywateli w ciągu całego ich życia, są spełnianiem misji Państwa. Szkoła oczywiście wypełnia tylko fragment życia obywateli, ale ma olbrzymią rolę, by przygotować na całe życie. Technologie mogą w tym znakomicie pomóc, istniejące i te których jeszcze nie znamy.

LT: **Jak widzi Pan profesor projekt „Laptop dla ucznia”?**

MMS: Ten projekt widzę jako realizację priorytetów strategicznych nakreślonych we wspomnianym wcześniej Planie. A zatem krótko, szkoła powinna być przygotowana na otrzymanie laptopów, zarówno w sensie technicznym, jak i mając opracowany plan swojego rozwoju z wykorzystaniem nowych rozwiązań; niezbędne są platformy edukacyjne jako miejsca, w których przebiegać będzie kształcenie; zanim uczniowie otrzymają laptopy powinni je dostać nauczyciele, których wcześniej należy przygotować do stosowania nowych technologii („miękkich” i „twardych”); i dopiero wtedy spokojnie można uczniom umożliwić dostęp do laptopów. Widzę w tym projekcie duże szanse na realizację jednego z założeń Planu, że technologia („miękka” – oprogramowanie i zasoby oraz „twarda” – sprzęt) będzie wszędzie tam, gdzie potrzebują jej uczniowie i nauczyciele, w szkole i poza nią.

LT: **Na koniec, czy nowa konferencja w Toruniu to nowe podejście do Konferencji IwSz?**

MMS: Konferencja nie jest całkiem nowa, w 2008 roku odbyła się już piaty raz. Różni się od konferencji „Informatyki w Szkole” tym, że większy nacisk jest kładziony na warsztaty dla nauczycieli, czyli na aktywność uczestników. Nie zapominamy przy tym o dobrych wykładach i prezentacjach nowych technologii. Czas pokaże, na ile ta nowa formuła przyjmie się w skali ogólnopolskiej, bo lokalnie, w województwie kujawsko-pomorskim, jest to bardzo popularna konferencja, umożliwiająca stały kontakt z nauczycielami. Mam też nadzieję, że ta konferencja będzie dobrym forum dla prezentacji zmian i nowych rozwiązań w edukacji informatycznej i zastosowań technologii informacyjno-komunikacyjnych w kształceniu.

Dziękuję za wywiad

Lesław Tomczak

MMS: Pozdrawiam i Życzę sukcesów opolskiej edukacji

mgr Janusz Podolak

Podsumowanie Egzaminu Maturalnego z Informatyki 2008

W tym roku maturzyści mogli zdawać egzamin maturalny z informatyki jedynie jako przedmiot dodatkowy, na poziomie rozszerzonym. W 2009 roku egzamin z informatyki będzie można, po raz pierwszy i chyba ostatni, zdawać jako przedmiot obowiązkowy na poziomie podstawowym lub rozszerzonym (więcej na ten temat w poprzednim wydaniu kwartalnika).

Egzamin składał się z dwóch części: pierwszej trwającej 90 minut i drugiej 150 minut. W obydwóch częściach abiturienci mieli do rozwiązania po 3 zadania. W pierwszej bez komputera a w drugiej przy użyciu z komputera. W sumie można było zdobyć maksymalnie 100 punktów, 40 za pierwszą i 60 za drugą część.

Oprócz arkuszy egzaminacyjnych abiturienci mieli do dyspozycji, w drugiej części egzaminu płyty z danymi oraz zainstalowane oprogramowanie, które wcześniej wybrali z listy CKE.

Środowisko	Język programowania	Program użytkowy*
Windows z systemem plików NTFS	- Turbo Pascal 5.5 lub nowszy - Free Pascal (FPC 2.0) lub nowszy - MS Visual Studio. NET C++ - Borland C++ Builder 6 Personal - Dev C++ 4.9.9.2 lub nowszy - Delphi 7 Personal - MS Visual Studio NET VB	- MS Office 2000 lub nowszy (w tym: Word, Excel, Access, PowerPoint)
Linux z KDE	- FreePascal (FPC 2.0) lub nowszy	- OpenOffice i MySQL 5.0 lub nowszy

**tylko jeden dla wybranego środowiska*

Na stronie Centralnej Komisji Egzaminacyjnej znajdują się do pobrania materiały związane z tym egzaminem:

- Arkusz części pierwszej
- Przykładowe rozwiązania do części pierwszej
- Arkusz części drugiej
- Dane
- Przykładowe rozwiązania do części drugiej
- Wyniki

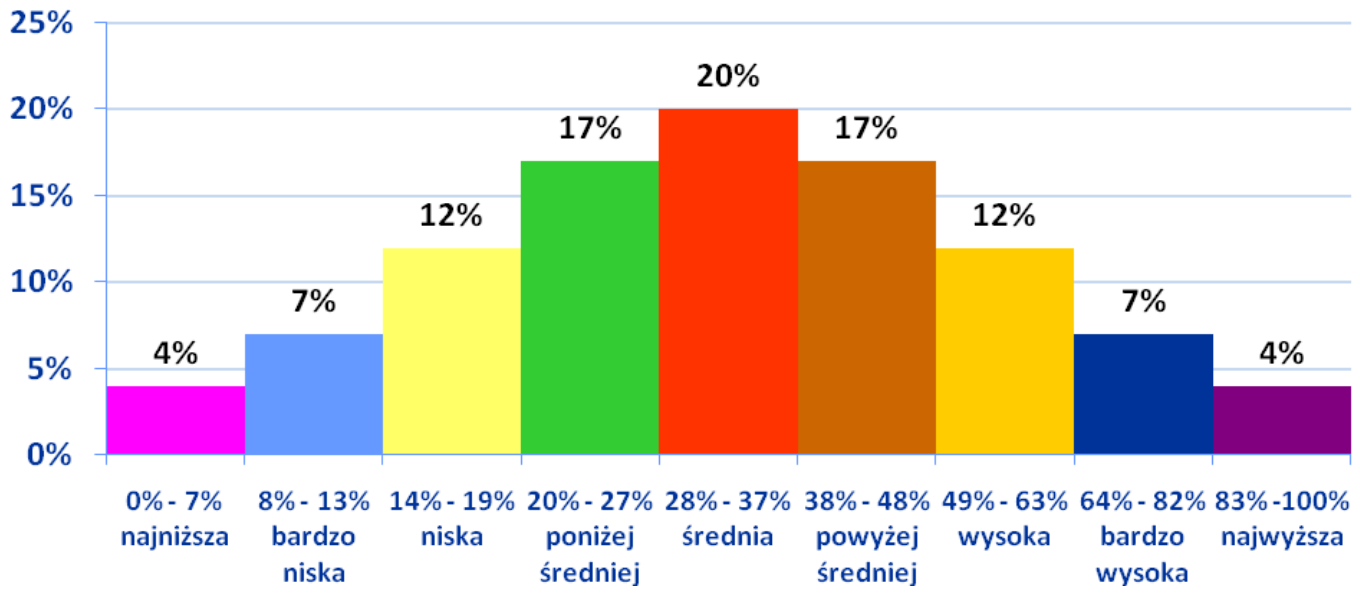
(Ciąg dalszy na stronie 5)

Podsumowanie Egzaminu Maturalnego z Informatyki 2008

(Ciąg dalszy ze strony 4)

Do egzaminu maturalnego z informatyki przystąpiło 1547 osób. Nie było w tym przypadku progu zaliczeniowego. Absolwenci na świadectwie mieli wpisany procent zdobytych punktów. Gdyby jednak przyjąć taki próg na poziomie tradycyjnych 30%, to zdawalność tego egzaminu wyniosłaby 60%. Jeśli porównam średnie zdobytych punktów, z przedmiotów zdawanych na poziomie rozszerzonym, informatykę znajdziemy na samym końcu z wynikiem 36%. Na pierwszym miejscu znajduje się język rosyjski z wynikiem ponad dwa razy lepszym 74%. Średnia ze wszystkich przedmiotów zdawanych na poziomie rozszerzonym wynosi około 58%. Wyniki można różnie interpretować i dochodzić do odmiennych wniosków. Musimy jednak pamiętać, że w tym roku szkolnym część uczniów wybierze informatykę jako przedmiot obowiązkowy. Sądzę, że warto im przedstawić sytuację z ostatniego egzaminu maturalnego, żeby już dziś byli w pełni świadomi czekającej ich pracy w przygotowaniach do tego egzaminu.

Rozkład wyników z informatyki w 2008 roku był dosyć klasyczny. Poniżej wyniki w podziale na dziewięć klas.



Informacje wykorzystane w artykule pochodzą ze strony http://cke.edu.pl/.

mgr Wiesław Bartoszek

Karta motorowerowa — jak zdobyć?

Po co karta motorowerowa?

- Po to, by przesiąść się z roweru na coś szybszego.
- Nawet jeśli nie masz jeszcze 18 lat i prawa jazdy.
- Karta motorowerowa to przepustka do śmigania swoim pierwszym "prawdziwym" pojazdem z silnikiem - skuterem czy motorynką.
- Jest oczywiście niezbędna, by jeździć bezpiecznie i legalnie, unikając nieprzyjemnych skutków w przypadku kontroli drogowej.

Ważna zasada !

- Motorowerzysta, który ukończył 13 lat i nie osiągnął 18 lat, może kierować motorowerem tylko wtedy, jeśli posiada uprawnienia, czyli kartę motorowerową.
- Motorowerem możesz wyjechać na drogę, jeśli masz przy sobie **3 dokumenty**: kartę motorowerową, dowód rejestracyjny pojazdu oraz ważną polisę ubezpieczeniową OC - odpowiedzialności cywilnej lub dowód opłacenia składki.
- Kierowca motoroweru i przewożona przez niego osoba muszą w czasie jazdy mieć na głowach kaski.
- Motorower musi być sprawny technicznie i posiadać czytelne tablice rejestracyjne.

Jak zdobyć kartę motorowerową?

- Kartę motorowerową wydaje dyrektor szkoły.
- Żeby ją uzyskać, musisz mieć ukończone 13 lat i zdany egzamin.
- Egzamin może przeprowadzić uprawniony przez dyrektora szkoły nauczyciel wychowania komunikacyjnego, policjant posiadający specjalistyczne przeszkolenie lub wyznaczony egzaminator.
- Egzamin jest z podobnych zagadnień, jak na kartę rowerową, lecz rozszerzonych o przepisy dla motorowerzystów. Jeśli masz już kartę rowerową, z łatwością uzupełnisz wiedzę i zdasz egzamin na kartę motorowerową.

Zasada ważna dla ciebie.

- Karta motorowerowa to dokument uprawniający do prowadzenia pojazdów z silnikiem do pojemności 50 cm3 - skuterów, motorowerów i motorynek.
- Motorowerzystę obowiązuje na drodze wiele reguł identycznych z tymi, jakie rządzą ruchem samochodowym. Dlatego warto je poznać - będziesz bezpieczniejszy na drodze, a w przyszłości łatwiej zdasz egzamin na prawo jazdy.

(Ciąg dalszy na stronie 7)

Karta motorowerowa — jak zdobyć?

(Ciąg dalszy ze strony 6)

REGULAMIN ZDOBYWANIA KARTY MOTOROWEROWEJ

1) Do egzaminu mogą przystąpić wszyscy uczniowie gimnazjum, którym pozwalają na to warunki zdrowotne potwierdzone zaświadczeniem lekarskim.

2) Warunkiem przystąpienia do egzaminu jest uczestnictwo rodziców ucznia w spotkaniu z prowadzącym szkolenie i podpisana osobiście zgoda rodziców ucznia.

3) Karta motorowerowa jest wydawana w szkole BEZPŁATNIE.

4) Każdy uczeń chcący zdawać egzamin zobowiązany jest do:

   a) odbycia szkolenia teoretycznego ze znajomości Prawa o Ruchu Drogowym w ramach przedmiotu Technika,

   b) uczestnictwa w spotkaniach z przedstawicielami Policji z Wydziału Ruchu Drogowego, których tematem jest prawidłowe zachowanie się na: drodze, podczas kontroli drogowej i podczas wypadku na drodze w wymiarze 2 godzin,

   c) uczestnictwa w spotkaniach z przedstawicielem Służby Zdrowia, których tematem jest udzielanie pierwszej pomocy poszkodowanym w wypadkach drogowych w wymiarze 2 godzin,

   d) odbycia jazdy motorowerem u osoby, która posiada uprawnienia i kwalifikacje do nauki jazdy,

   e) wypełnienia podania o wydanie karty motorowerowej i dołączenia dwóch zdjęć,

   f) dostarczenia opinii wychowawcy klasy o uczniu.

5) Uczniowie składają egzamin w formie testu.

6) Test zawiera 25 pytań o następującej tematyce: znaki i sygnały drogowe, manewry na drodze, typowe sytuacje drogowe (skrzyżowania drogi głównej z drogami podporządkowanymi , skrzyżowania dróg równorzędnych i z ruchem okrężnym), pierwsza pomoc ofiarom wypadków.

7) Za pozytywnie zdany egzamin uważa się udzielenie przez ucznia 80% poprawnych odpowiedzi w teście i dostarczenie zaświadczenia potwierdzającego umiejętności jazdy motorowerem.

8) Uczniowi, który nie zdał egzaminu przysługuje prawo do zdawania jednego egzaminu poprawkowego.

9) Jeżeli wynik egzaminu poprawkowego jest negatywny uczniowi przysługuje prawo do składania go dopiero w następnym roku szkolnym.

10) Komisja egzaminacyjna składa się: z Dyrektora Szkoły, przedstawiciela Policji i nauczyciela prowadzącego szkolenie.

dr Beata Bułka

Perspektywy przedmiotu Technika

Streszczenie

Poniżej zostały przedstawione rozważania na temat miejsca wychowania przez pracę w szkole ogólnokształcącej w Polsce. Poddano analizie propozycję nowej podstawy programowej kształcenia ogólnego pod kątem przedmiotu nauczania gospodarstwo domowe.

Wychowanie przez pracę, czyli *zamierzony i celowo zorganizowany rodzaj działalności wychowawczej, którego cechę szczególną stanowi wykorzystywanie pracy w procesach oddziaływania na jednostkę i dokonywania zmian w jej osobowości* (Wiatrowski, 2005, s. 154) realizowane jest w Polsce od dawna, począwszy od „Ustaw" Komisji Edukacji Narodowej. Nie możemy więc tak po prostu, zrezygnować z realizacji powyższych idei, Musimy pamiętać, że to właśnie wychowanie przez pracę jako jedna z wielu dziedzin wychowania jest podstawą wychowania młodego pokolenia w Polsce. Niestety przyglądając się poczynaniom wybitnych polskich naukowców, śmiem w to wątpić.

Analizując poddany pod publiczną dyskusję projekt podstawy programowej kształcenia ogólnego zauważa się brak przedmiotu technika na drugim etapie kształcenia. W przedstawionych założeniach projektu nowej podstawy programowej kształcenia ogólnego czytamy „Tworząc projekt nowej podstawy, nie zamierzamy radykalnie zrywać z dotychczasowymi rozwiązaniami, lecz jedynie poprawić je tam, gdzie to konieczne". Czy zlikwidowanie tak ważnego przedmiotu nie było działaniem zbyt radykalnym? Czy szacowny zespół koordynatorów ds. podstawy programowej nie zrobił tego zbyt pochopnie?

Rozumiem, jak każdy nauczyciel i obywatel, że zmiany są konieczne, wymaga tego od nas rewolucja naukowo-techniczno-informatyczna, ale zmiany na lepsze nie na gorsze!

W obowiązującej podstawie programowej kształcenia ogólnego dla szkół podstawowych i gimnazjów czytamy: „Nadrzędnym celem działań edukacyjnych szkoły jest wszechstronny rozwój ucznia. Edukacja szkolna polega na harmonijnej realizacji przez nauczycieli zadań

(Ciąg dalszy na stronie 9)

Perspektywy przedmiotu Technika

(Ciąg dalszy ze strony 8)

w zakresie nauczania, kształcenia umiejętności i wychowania." Czy szkoła według zaproponowanego projektu będzie mogła zapewnić uczniom wszechstronny rozwój ucznia? Jak będzie wyglądała sprawa równowagi, o ważności, której nie trzeba nikogo przekonywać, przekazywania wiedzy, kształcenia umiejętności i postaw? Jak nauczyć dzieci twórczego i abstrakcyjnego myślenia, bez którego świat w chwili obecnej wyglądałby zupełnie inaczej? Jak przygotować dzieci i młodzież „do osiągnięcia celów życiowych i wartości ważnych dla odnalezienia własnego miejsca w świecie" w przypadku braku zagadnień preorientacji zawodowej? Do zadań szkoły z zakresu kształcenia technicznego należy: doprowadzenie ucznia do poznania i oceniania swoich cech, możliwości i predyspozycji technicznych oraz organizowanie wielostronnej aktywności technicznej ucznia. Jak zrealizować powyższe zadania bez elementów kształcenia politechnicznego, bez wychowania przez pracę?

Nie znajdziemy również w projekcie podstawy programowej przedmiotu technika na trzecim etapie kształcenia. W tym przypadku zaproponowano wprowadzić przedmiot gospodarstwo domowe, lecz niestety nie jest to tylko zmiana nazewnictwa. Przede wszystkim poddano bardzo dużej modyfikacji cele i treści kształcenia, które w obecnym brzmieniu nie są całkowicie spójne z polskimi celami kształcenia technicznego w gimnazjum (głównym celem edukacyjnym do tej pory było: „Przygotowanie do życia w cywilizacji technicznej"). Należy zadać sobie pytanie, czy przedmiot gospodarstwo domowe w zaproponowanej postaci przygotuje młodzież do życia w cywilizacji technicznej? Uważam, że nie spełni tego zadania, z kilku bardzo ważnych powodów:

Brak celów i treści z zakresu wychowania komunikacyjnego. Pomijam istotny fakt, że umiejętności zachowania się na drodze, jako pieszy, pasażer i rowerzysta uczeń powinien zdobyć jeszcze w szkole podstawowej. Nie będę przytaczać danych statystycznych, ale od czasu wprowadzenia do szkół wychowania komunikacyjnego znacznie zmniejszyła się ilość wypadków drogowych z udziałem dzieci zarówno jako pieszych jak i rowerzystów. Śmiem przypomnieć o odrębnych przepisach zawartych w Prawie o

(Ciąg dalszy na stronie 10)

Perspektywy przedmiotu Technika

(Ciąg dalszy ze strony 9)

ruchu drogowym, które to zobowiązują szkoły do realizacji zagadnień związanych z wychowaniem komunikacyjnym.

Brak treści z mechaniki. Jak osiągnąć wymagania określone w p. 9 „Złożyć i wyregulować według instrukcji urządzenie codziennego użytku (np. rower, deskorolka)", bez podstawowych wiadomości z zakresu mechaniki? Z drugiej strony, kto w dzisiejszych czasach sam składa rower?

Brak treści z historii techniki i wynalazczości.

Brak treści z projektowania (z rysunku technicznego). Jak zrealizować treści zawarte w p. 4 Adaptacje i wytwarzanie? Jak wytworzyć karmnik dla ptaków czy ramkę bez znajomości podstaw rysunku technicznego?

Brak treści z materiałoznawstwa. Jak wytworzyć np. budkę lęgową bez znajomości właściwości drewna? Jak przyszyć odpowiednią łatę bez znajomości rodzajów materiałów włókienniczych?

Należy zmodyfikować następujące treści kształcenia

- **Żywienie.** Czy o higienie przygotowywania posiłku, jak również umiejętności nakrywania do stołu nie powinno się uczyć dzieci już w szkole podstawowej? Czy 12 - letnie dzieci nie przygotowywują sobie małych przekąsek?
- **Ubiór.** Czy o higienie ubioru, dostosowywaniu stylu do okoliczności będziemy uczyć dopiero młodzież w gimnazjum? Czy to aby nie za późno? Pewne nawyki dziecko kształtuje w sobie od najmłodszych lat i trudno jest nagle po 13 latach coś zmieniać. Jak nauczyć higieny ubioru, czy też naszywania odpowiednich łat bez znajomości zagadnień z włókiennictwa?
- **Savoir-vivre.** Zarówno zasady witania się, przedstawiania, żegnania, zachowania się przy stole, prowadzenia rozmowy towarzyskiej dzieci powinny poznać w szkole podstawowej a może nawet w przedszkolu. Czas gimnazjum to stanowczo za późno!

(Ciąg dalszy na stronie 11)

Perspektywy przedmiotu Technika

(Ciąg dalszy ze strony 10)

I na zakończenie moich rozważań na temat proponowanej podstawy programowej kształcenia ogólnego zadam jeszcze jedno, bardzo istotne z punktu widzenia najbardziej zainteresowanego, czyli ucznia, pytanie: Jak rozbudzać i rozwijać indywidualne zainteresowania ucznia, skoro on sam nie będzie miał szans ich poznać?

Postulaty:

1. Należy zachować przedmiot technika w szkołach podstawowych. Oczywiście niekoniecznie w tej samej niezmienionej postaci. Modyfikujmy, ale nie likwidujmy!
2. Jeśli w miejsce techniki wprowadzić nowy przedmiot na przykład gospodarstwo domowe to:
   a. Koniecznie należy zachować zagadnienia wychowania komunikacyjnego w gimnazjum, być może w treściach kształcenia przedmiotu gospodarstwo domowe.
   b. Do treści kształcenia przedmiotu gospodarstwo domowe należy wprowadzić następujące zagadnienia:
      - projektowania
      - mechaniki
      - materiałoznawstwa
      - treści z historii techniki i wynalazczości.

Literatura:

Wiatrowski Z.: Podstawy pedagogiki pracy, Wydawnictwo Akademii Bydgoskiej, Bydgoszcz 2005, ISBN 83-7096-524-5

Bartnik E., Konarzewski K., Kowalczykowa A., Marciniak Z., Merta T.: Podstawa programowa kształcenia ogólnego. Projekt. - Warszawa: Instytut Spraw Publicznych, 2005, ISBN 838981-7551

mgr Lesław Tomczak

Nowe oprogramowanie firmy Microsoft do szkolnej pracowni

Czym jest Pakiet Przejścia dla SBS2003?

Pakiet przejścia (ang. Transition Pack) dla systemu MS Windows SBS 2003 R2 Premium Edition umożliwia zamianę zainstalowanego (skonfigurowanego i działającego) serwera SBS w jednej z poniższych wersji:

- MS Windows SBS 2003 Premium
- MS Windows SBS 2003 SP1 Premium
- MS Windows SBS 2003 R2 Premium

na pełną wersję systemu serwerowego **Windows Server 2003 R2 Standard Edition.**

Tym samym **znosi wbudowane do systemu SBS ograniczenia**, czyniąc go pełnoprawnym systemem serwerowym dającym znaczne możliwości skalowalności w zależności od potrzeb.

Największą zaletą Pakietu Przejścia jest możliwość jego instalacji na **DZIAŁAJĄCYM SERWERZE SBS.** W wyniku prawidłowej instalacji zachowane zostają wszystkie ustawienia konfiguracyjne, informacje dotyczące kont użytkowników. Nie tracimy żadnych kreatorów, ustawień ani dokumentów.

W jakich wypadkach warto zastosować Pakiet Przejścia?

W szkolnych warunkach główną zaletą zastosowania Pakietu Przejścia jest zniesienie ograniczenia licencyjnego nie pozwalającego podłączyć do jednego serwera więcej niż 75 stacji roboczych. Ma to szczególne znaczenie dla szkół, w których obecnie znajduje się już ponad 75 komputerów lub w najbliższym czasie jest szansa, że taka liczba zostanie przekroczona.

Mimo iż liczba 75 komputerów dla szkoły może wydawać się dużą, to biorąc pod uwagę, że szkolna pracownia ponadgimnazjalna dostarczana w ramach projektów EFS liczyła 20 komputerów + serwer, a wiele szkół na przestrzeni kilku lat zostało wyposażonych w trzy takie pracownie oraz jedną bądź kilka pracowni dla szkolnego Centrum Informacji Multimedialnej, to liczba komputerów w szkole zbliża się do tej granicy. Jeżeli weźmie się dodatkowo pod uwagę komputery stojące jako pojedyncze stanowiska pracy nauczycieli w ich salach

(Ciąg dalszy na stronie 13)

Nowe oprogramowanie firmy Microsoft do szkolnej pracowni

(Ciąg dalszy ze strony 12)

lekcyjnych, oraz komputery będące na usługach szkolnej administracji, to może się okazać, że w coraz większej ilości szkół maksymalna liczba 75 komputerów już została przekroczona.

Warto też wziąć pod uwagę fakt, iż coraz większa liczba nauczycieli i uczniów posiada różnego rodzaju urządzenia mobilne. Coraz częściej opiekun ma do czynienia w szkole z sytuacją, w której dostaje pytania i prośby o możliwość podłączenia takich urządzeń do szkolnej sieci celem wymiany dokumentów i korzystania z sieci Internet. W takich wypadkach liczba urządzeń współpracujących z serwerem i korzystających z zasobów na nim udostępnionych , takich jak dokumenty, drukarki, poczta, miejsce przeznaczone na prywatne strony WWW, może znacząco wzrosnąć w krótkiej perspektywie czasu.

Zastosowanie Transition Pack pozwala wszystkie te komputery podłączyć do jednego serwera i zarządzać nimi w sposób scentralizowany - korzystny w tym wypadku zarówno dla osoby zarządzającej szkolną siecią, jak również dla użytkowników tej sieci. Do głównych zalet można zaliczyć:

- Prostszą administrację – mając pod kontrolą jeden serwer łatwiej jest nim zarządzać, dbać o jego aktualizację, wdrażać konta uczniowskie i zasady dotyczące działania stanowisk roboczych w naszej sieci,
- Centralne repozytorium dokumentów użytkowników – użytkownicy niezależnie od miejsca logowania (sala informatyczna, sala lekcyjna, szkolna biblioteka czy sekretariat) zawsze mają dostęp do swoich dokumentów oraz skonfigurowanych preferencji systemowych.
- Możliwości wykorzystania komputerów pełniących wcześniej rolę serwerów, jako systemy awaryjne i / lub stacje robocze.
- W razie dalszej, znacznej rozbudowy sieci możliwość rozdzielenia ról serwera na oddzielne maszyny (np. komputer tylko z systemem Exchange, oddzielny z Active Directory itd.)
- Inne.

(Ciąg dalszy na stronie 14)

Nowe oprogramowanie firmy Microsoft do szkolnej pracowni

(Ciąg dalszy ze strony 13)

Powyższą listę kończy punkt inne, gdyż zalet jest znacznie więcej. My staraliśmy się wymienić te najważniejsze z punktu widzenia szkoły i sieci komputerowej w niej implementowanej.

Czy zasadniczo zmienia się sposób zarządzania serwerem po zainstalowaniu Pakietu Przejścia?

Jedną z zalet zastosowania Pakietu Przejścia jest fakt, że Administrator nie traci narzędzi, do których jest przyzwyczajony. Wszystkie występujące w Sewerze SBS kreatory (np. Kreator dostępu do sieci Internet) pozostają w systemie i dalej działają poprawnie. Oprócz tego, użytkownik ma możliwość wyboru narzędzi przeznaczonych do administracji w systemach Windows Server 2003. Więc nie tylko nie tracimy dotychczasowych możliwości administracyjnych, ale zyskujemy również nowe.

Czy, a jeśli tak, to na jakie wsparcie dotyczące instalacji i konfiguracji programu Pakiet Przejścia mogę liczyć?

W ramach współpracy firmy Microsoft oraz Ośrodka Edukacji Informatycznej i Zastosowań Komputerów został przygotowany cały pakiet wsparcia przeznaczonego dla szkolnych opiekunów, którzy zdecydują się na wdrożenie Pakietu Przejścia w swojej placówce. Do elementów pakietu wsparcia, oprócz standardowej pomocy oferowanej przez firmę Microsoft, można zaliczyć:

- Internetowy serwis pod adresem: http://www.partnerstwodlaprzyszlosci.edu.pl/tpack gdzie opublikowane zostały m.in. instrukcje instalacji Pakietu Przejścia dla serwerów instalowanych z poszczególnych wersji tzw. DVD kolekcji
- Oprócz instrukcji instalacji, w serwisie można znaleźć porady dotyczące użytkowania serwera już po zainstalowaniu i skonfigurowaniu Pakietu Przejścia, instrukcje oraz materiały audio/wideo z posługiwania się serwerem
- Na stronach serwisu jest dostępne forum przeznaczone do zadawania pytań związanych z działaniem serwera po zainstalowaniu Pakietu Przejścia, gdzie można otrzymać szczegółowe podpowiedzi rozwiązań dla nietypowych sytuacji awaryjnych.

Instrukcje z opisem instalacji dla serwerów przygotowanych przy użyciu DVD kolekcji w dowolnej wersji, dostępne są w niniejszym serwisie.

mgr Marian Mysior

Nowości Microsoftu—SQL Server 2008 Express

Treści programowe z działu baz danych, w szkolnej informatyce i technologii informacyjnej, najczęściej realizowane są w oparciu o *Microsoft Access*. Niekoniecznie jednak tak musi być. Może wykorzystać tu nowocześniejsze narzędzie o architekturze klient – serwer: *SQL Server 2008*?

Zgodnie z wcześniejszymi zapowiedziami, udostępniono do darmowego pobrania SQL Server 2008 Express w wariantach SQL Server 2008 Express with Tools oraz SQL Server 2008 Express with Advanced Services. Charakteryzują się one większym zestawem funkcjonalności, niż wersja podstawowa. Obydwie edycje zawierają w sobie również najnowszą wersję narzędzia SQL Server Management Studio Basic (narzędzie od zarządzania bazą). Przy okazji warto wspomnieć, że według zapowiedzi twórców nie będzie możliwości pobrania go osobno – osoby, które zainstalowały już podstawową wersję SQL Server 2008 Express, a chcą używać tego wygodnego narzędzia, będą musiały zaktualizować ją przynajmniej do wersji SQL Server 2008 Express with Tools.

Zawartość, dostępnych w różnych wariantach, pakietów *SQL Server 2008 Express*:

SQL Server 2008 Express

- Silnik serwera baz danych – możliwość tworzenia, przechowywania, aktualizowania i pobierania danych

SQL Server 2008 Express with Tools

- Silnik serwera baz danych – możliwość tworzenia, przechowywania, aktualizowania i pobierania danych
- SQL Server Management Studio Basic – narzędzie do wizualnego zarządzania bazami danych

SQL Server 2008 Express with Advanced Services

- Silnik serwera baz danych – możliwość tworzenia, przechowywania, aktualizowania i pobierania danych
- SQL Server Management Studio Basic – narzędzie do wizualnego zarządzania bazami danych
- Wyszukiwanie pełnotekstowe – możliwość wydajnego wyszukiwania danych w tekście
- Usługi raportujące – zintegrowane środowisko do tworzenia i projektowania raportów

Źródła:

- www.codeguru.pl,
- www.microsoft.com/downloads/details.aspx?FamilyId=7522A683-4CB2-454E-B908-E805E9BD4E28&displaylang=en,
- www.microsoft.com/downloads/details.aspx?FamilyId=B5D1B8C3-FDA5-4508-B0D0-1311D670E336&displaylang=en.

mgr Janusz Podolak

Hot Potatoes—Gorące ziemniaki

Hot Potatoes to zestaw programów do tworzenia multimedialnych ćwiczeń i testów. Materiały utworzone za pomocą HP mogą być publikowane w postaci stron internetowych lub w formacie SCORM.

UWAGA!

Hot Potatoes nie jest programem typu freeware. W wersji komercyjne jego koszt to około 90$. HP może być darmowy dla osób, które tworzą przy jego pomocy zasoby do celów edukacyjnych w instytucjach oświatowych. Za utworzone w ten sposób zasoby nie można pobierać opłat a dodatkowo muszą one być osiągalne na ogólnie dostępnej stronie internetowej.

Poniżej krótki opis składowych programu

JQuiz

Moduł ten służy do tworzenia testów/quizów. Mam do wyboru cztery rodzaje pytań.

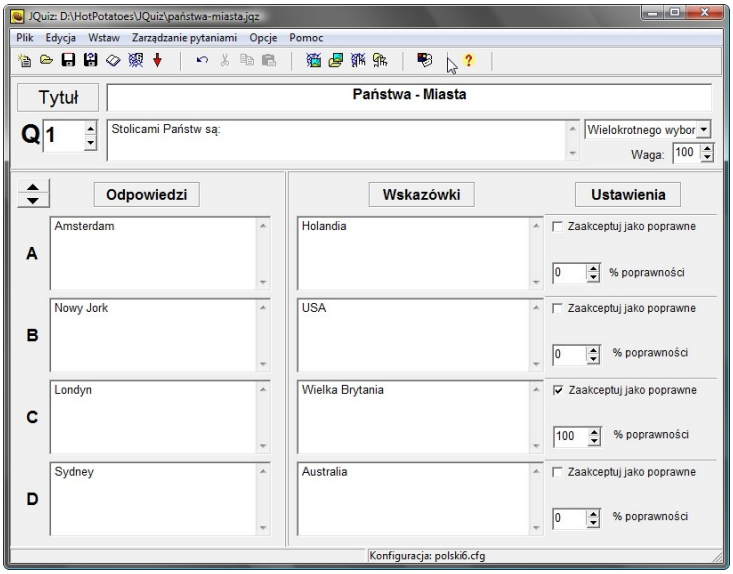
(Ciąg dalszy na stronie 17)

Hot Potatoes—Gorące ziemniaki

(Ciąg dalszy ze strony 16)

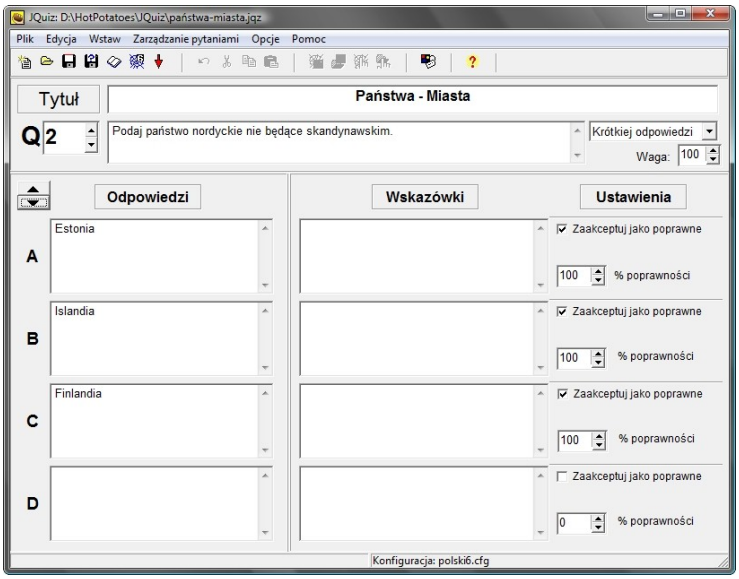
Wielokrotnego wyboru

W tak przygotowanym pytaniu możemy wybierać odpowiedzi, do momentu kiedy nie trafimy na odpowiedź poprawną. Nasz wybór ma wpływ na punktację. Jeżeli trafimy za pierwszym razem w poprawną odpowiedź otrzymamy maksymalną ilość punktów. Wybór poprawnej odpowiedzi na końcu nie daje nam żadnych punktów. Między tymi skrajnymi sytuacjami są też pośrednie.



Krótkie odpowiedzi

W tym przypadku należy wpisać jedną z oprawnych odpowiedzi. Może być jedna z jednej. Kwestia rozróżniania wielkich liter jest do ustawienia w opcjach konfiguruj format wyjściowy (zakładka inne).



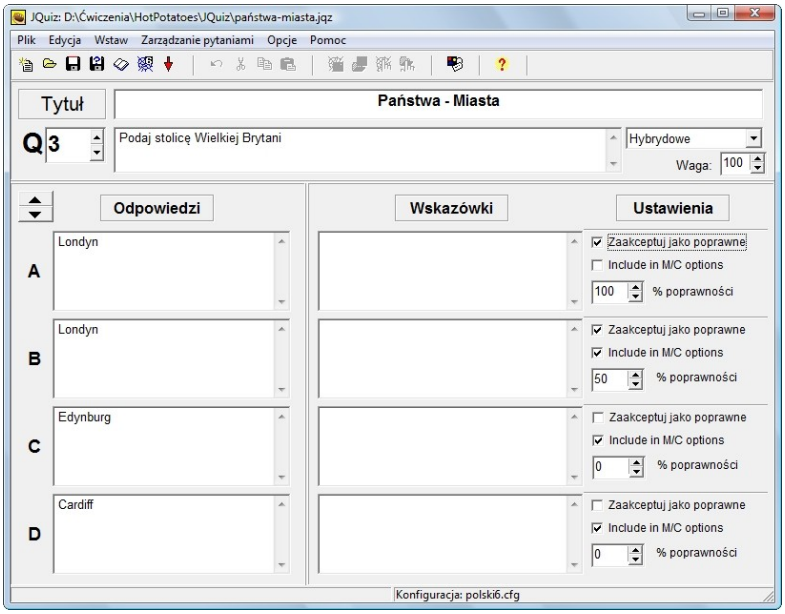
(Ciąg dalszy na stronie 18)

Hot Potatoes—Gorące ziemniaki

(Ciąg dalszy ze strony 17)

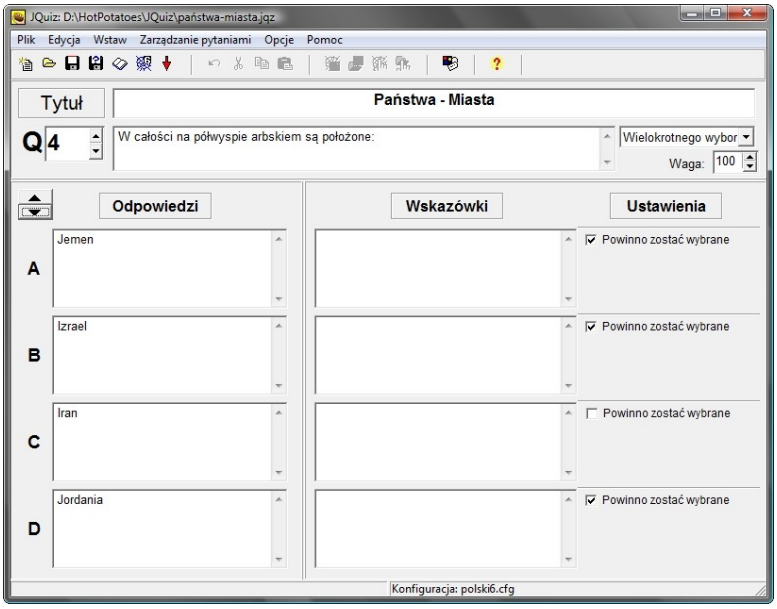
Hybrydowe

To pytanie wymaga wpisania odpowiedzi, podobnie jak w poprzednim. Różnica polega na tym, że jeżeli odpowiedź jest błędna, to wyświetlają się odpowiedzi do wyboru. We wspomnianym wcześniej oknie konfiguruj format wyjściowy, zakładka inne, ustalam ile odpowiedzi należy wpisać aby pojawiły się te do wyboru.



Wielokrotnego wyboru (2)

Przy tym pytaniu (nazwa taka sama jak w pierwszym) wybieramy poprawne odpowiedzi. Program liczy punkty, zarówno za znaczenie poprawnych jak i pozostawienie niezaznaczonych błędnych odpowiedzi.



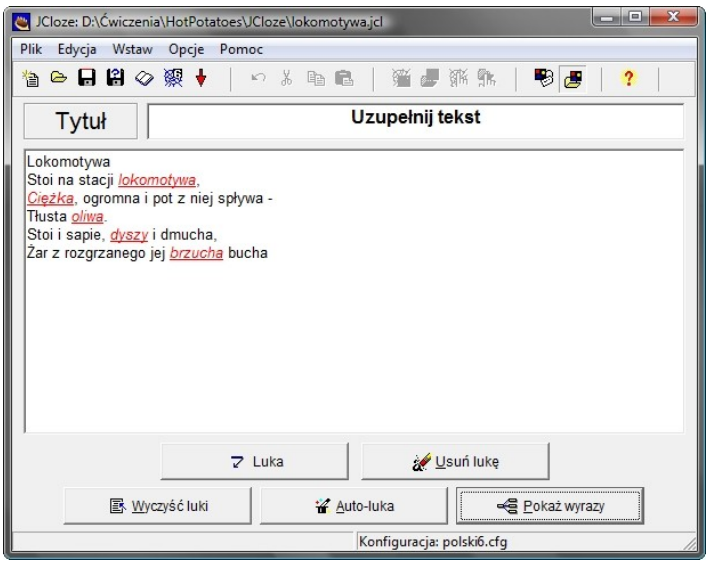
(Ciąg dalszy na stronie 19)

Hot Potatoes—Gorące ziemniaki

(Ciąg dalszy ze strony 18)

JCloze

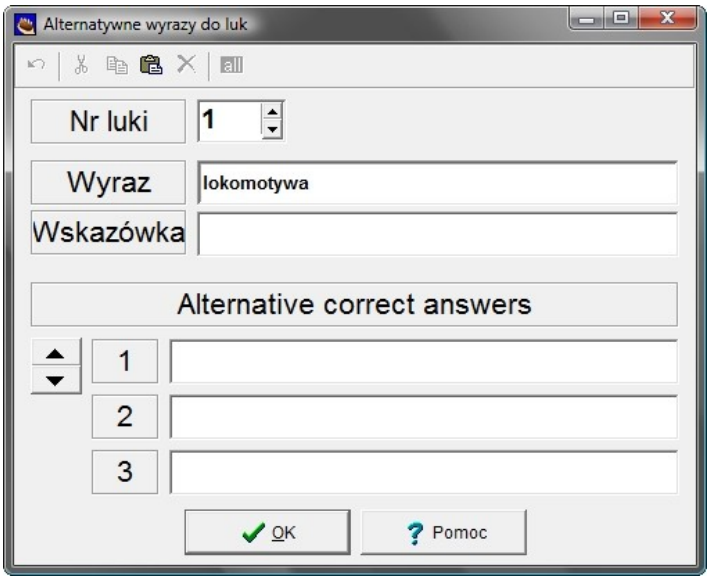
Moduł pozwala na umieszczenie tekstu z lukami do uzupełnienia.



Luki możemy tworzyć zaznaczając fragment tekstu a następnie wybierając przycisk luka. Ewentualnie wybieramy przycisk Auto-luka i podajemy liczbę wskazującą, co który wyraz ma być tworzona luka.

Lukę możemy uzupełnić wskazówką lub też alternatywną odpowiedzią. Jest możliwe bezpośrednio po wybraniu wyrazu lub po naciśnięciu klawisza Pokaż wyrazy.

Po wybraniu polecania Konfiguruj format wyjściowy, z paska narzędzi lub menu opcje, otwiera się okno pozwalające na konfigurację zadania. W zakładce klawisze możemy, między innymi, zdecydować czy będą wyświetlane wprowadzone przez nas wskazówki. Natomiast w zakładce inne zdecydujemy, czy widoczne będą wyrazy do uzupełnienia. Możemy również w tym miejscu spowodować, że wyrazy będą wybierane z listy rozwijalnej.



(Ciąg dalszy na stronie 20)

Hot Potatoes—Gorące ziemniaki

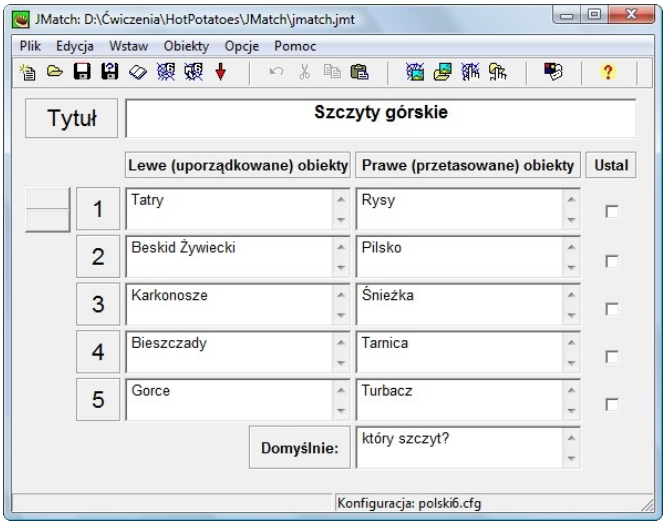
(Ciąg dalszy ze strony 19)

JMatch

Moduł ten pozwala na przygotowanie zestawu ćwiczeń polegających dopasowaniu wyrazów.

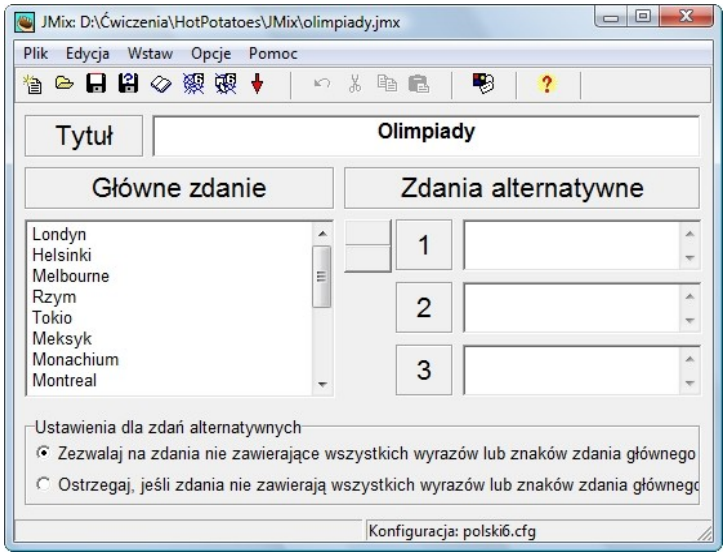
W gotowym ćwiczeniu należy z listy rozwijalnej wybrać odpowiednie wyrazy.

Zaznaczenie opcji ustal powoduj, że w tym przypadku rozwiązanie zostanie wyświetlone.



JMix

Rozsypanka. Podane elementy należy ułożyć w odpowiedniej kolejności.



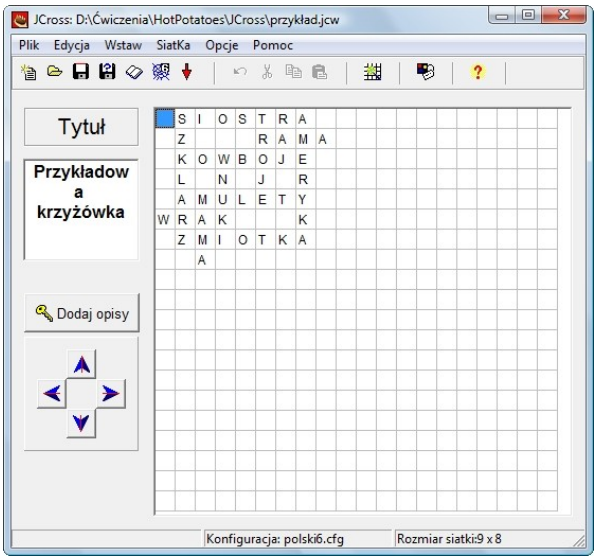
(Ciąg dalszy na stronie 21)

Hot Potatoes—Gorące ziemniaki

(Ciąg dalszy ze strony 20)

JCross

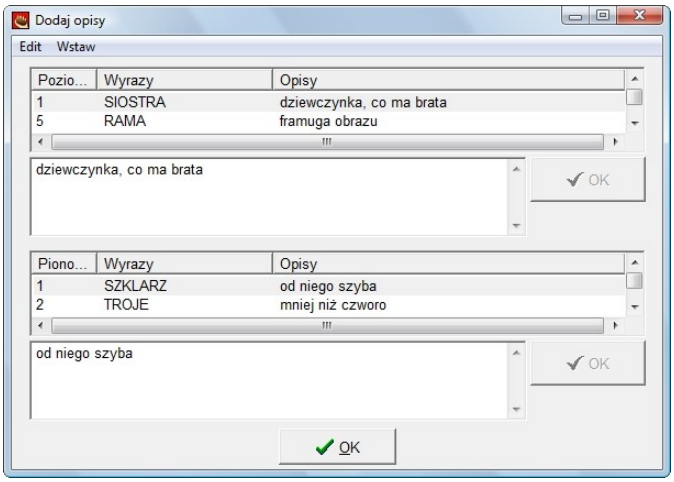
W tym module możemy stworzyć krzyżówkę Rozpoczynamy od wpisania haseł do krzyżówki. Możemy również skorzystać z narzędzia Automatyczne tworzenie siatki, w menu siatka. Należy wówczas podać hasła, które mają się znaleźć w krzyżówce a program sam utworzy krzyżówkę.



Żeby wprowadzić opisy haseł wybieramy przycisk Dodaj opisy.

Tu zostaje nam zaznaczenie hasła i wprowadzenie opisu.

W znanym z wcześniejszych modułów oknie Konfiguruj format wyjściowy możemy dokonać modyfikacji ustawień. Na przykład spowodować, że wszystkie opisy haseł, od początku będą widoczne.



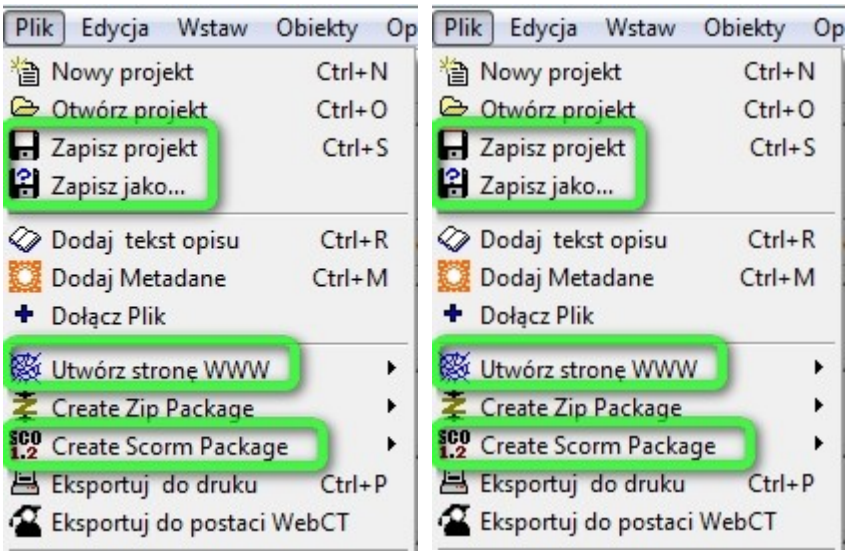
Masher

Jest to narzędzie do kompilowania ćwiczeń utworzonych w poprzednich modułach w jeden materiał. Korzystanie z tego modułu wymaga posiadania komercyjnej licencji.

Zapisywanie modułów

(Ciąg dalszy na stronie 22)

Hot Potatoes—Gorące ziemniaki



Każdy moduł można zapisać w postaci projektu. Wówczas możemy modyfikować utworzony zasób. Istniej też możliwość publikacji w formacie HTML lub SCORM. Przykłady omawiane w artykule można pobrać tu. Są one zapisane zarówno w formie projektu (to znaczy można je modyfikować) jak i stron htm.

Odnośniki

web.uvic.ca/hrd/hotpot/ – stron główna Hot Potatoes (stąd można pobrać program)

www.hotpot.hojnacki.net/ – polski serwis HP autorstwa Lechosława Hojnackiego, znajdziemy tu między innymi pliki umożliwiające spolszczenie programu oraz przydatne, w tworzeniu materiałów informacje a także gotowe ćwiczenia.

Quizy i testy przygotowane w HP odnajdziemy na wielu stronach WWW. Poniżej parę odnośników do takich stron:

www.matematyka.hojnacka.net/2008/07/testy-dla-uczniw-elementy-logiki-w.html

mi.kn.bielsko.pl/~mi00iga/hotpot/

www.sp81.edu.lodz.pl/quiz/quiz_index.html

mi.kn.bielsko.pl/~mi00akus/testy/

mi.kn.bielsko.pl/~mi01kcz/procenty/zabawy/

http://www.oeiizk.edu.pl/matma/kwasnik/start.html

mgr Roland Zimek

Opera i jej funkcjonalność

Kto pamięta **Netscape Navigatora**, przemianowanego później na **Netscape Communicator**?. Program ten , mający w ówczesnych czasach niepodzielne panowanie na większości komputerów został wyparty przez znanego nam **Microsoft Internet Explorera** i praktycznie zapomniany. Znany wszystkim **IE** jest dołączany „bezpłatnie” do każdej wersji **Microsoft Windows**. Dlatego też to on właśnie jest dziś królem przeglądarek. Jeżeli jednak nie będzie on tak dynamicznie rozwijany jak przeglądarki konkurencji, to podzieli losy opisanego wcześniej **Netscape-a**. Już obecnie w Polsce **Firefox 3** jest częściej używaną przeglądarką wśród informatyków niż **IE**. Jednak gdzie dwóch się bije tam trzeci korzysta. I właśnie tą trzecią przeglądarką jest **Opera 9.6**.

Opera 9.6 jest przeglądarką dostępną na rynku od połowy lat 90-tych ubiegłego wieku. Początkowo była ona wydawana w wersji shareware, a później adware, jednak już od roku 2005 jest ona w pełni darmowa, a także polskojęzyczna. Już w 2001 roku użytkownicy Opery mogli cieszyć się z kart i pola wyszukiwarki, które to dodatki pojawiły się w innych wersjach przeglądarek znacznie później. A i dziś **Opera** dystansuje rywali umożliwiając głosową obsługę najważniejszych poleceń.

Przeglądarka ta jest szczególnie interesująca i godna polecenia z kilku niezmiernie istotnych punktów. Razem z **Firefoksem 3** jest uważana za najszybszą i najbezpieczniejszą przeglądarką, znacznie pod tym względem wyprzedzając **IE**. Jednak niezaprzeczalną zaletą **Opery** jest znacznie mniejsza ilość błędów znaleziona w ciągu kilku ostatnich lat, niż u konkurencji. A i same błędy nie były krytyczne. Jako jedyna przeglądarka, **Opera** niemal natychmiast łata wszystkie „dziury”. U konkurencji nie jest już tak różowo, gdyż zawsze pozostaje przynajm-

(Ciąg dalszy na stronie 24)

Opera i jej funkcjonalność

(Ciąg dalszy ze strony 23)

niej kilka otwartych luk. Dzięki temu użytkownicy **Opery** mogą zawsze czuć się bezpieczni. A i sami hakerzy znacznie częściej koncentrują się na **IE** oraz **Firefoksie**, ze względu na ich większą popularność.

Użytkownicy **Firefoksa** zachwalają możliwość doinstalowania wielu dodatków zwiększających możliwości tej przeglądarki i pozwalających dostosowaniu do własnych potrzeb. A co muszą zrobić użytkownicy **Opery**, aby mieć funkcjonalność taką samą jak **Firefox** z doinstalowanymi dodatkami? Muszą tylko zainstalować **Operę**, gdyż domyślnie posiada ona już odpowiednią funkcjonalność. Jeżeli jednak jesteśmy zwolennikami „wodotrysków”, to pełno ich znajdziemy wybierając z górnego menu *Widżety/Dodaj Widżety*.

Z innych funkcjonalności Opery wystarczy wymienić obsługę gestów myszki, blokowanie wyskakujących okienek, wbudowany menedżer pobierania plików, blokowanie niebezpiecznych stron WWW, pełną obsługę sesji z możliwością zapisania stanu na dysku, klienta pocztowego.

Czy **Opera** posiada jakieś braki? Tak jak w każdym programie zawsze jest coś czego brakuje. Jak dla mnie istotnym brakiem w **Operze** jest brak zainstalowanego słownika ortograficznego, sprawdzającego poprawność pisowni. Na szczęście można go sobie doinstalować.

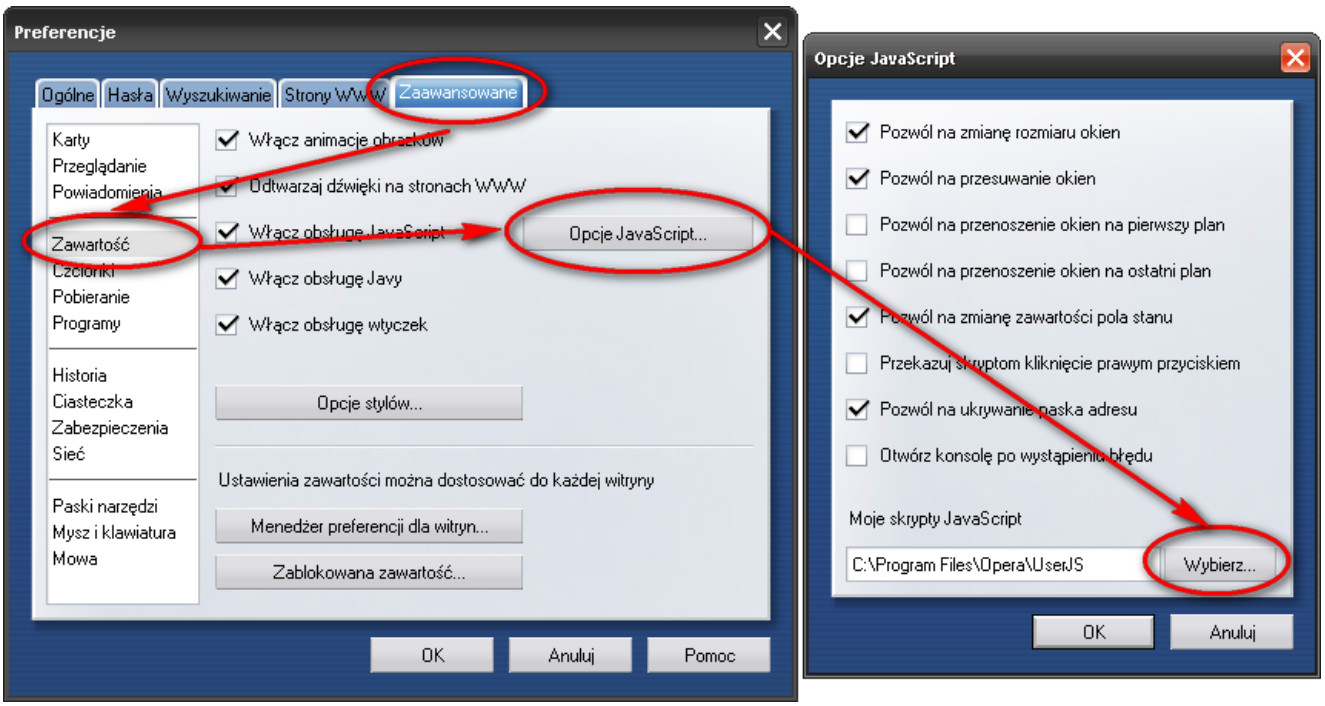
Oto szybki przepis jak to zrobić. Po pierwsze przejdź do folderu, w którym masz zainstalowaną **Operę**. W tym celu wybierz przycisk *Start*, a następnie *Mój komputer* (w systemie *Vista* – *Komputer*). Przejdź na *dysk lokalny C:*, następnie *Program Files* i wybierz folder *Opera*. Utwórz własny folder. Następnie odszukaj w sieci i pobierz pliki *ospell.js* oraz *aospell_prefs.js* i zapisz je w utworzonym wcześniej folderze. Teraz wystarczy tylko poinfor-

(Ciąg dalszy na stronie 25)

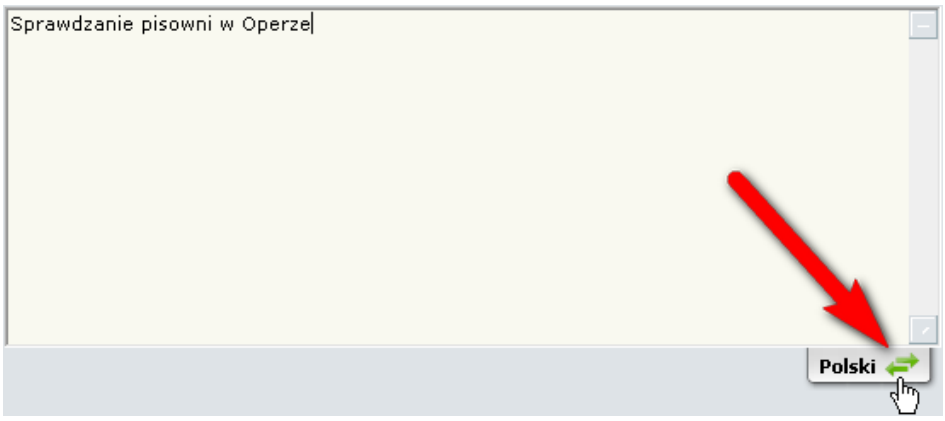
Opera i jej funkcjonalność

(Ciąg dalszy ze strony 24)

mować **Operę** gdzie ma szukać zapisanych plików. W tym celu wybierz z górnego menu *Narzędzia/Preferencje*. Wybierz zakładkę *Zaawansowane*, a następnie *Zawartość* i kliknij na przycisku *Opcje JavaScript*. W nowym okienku kliknij na przycisku *Wybierz* i wskaż folder w którym zapisałeś opisane wcześniej pliki.



Gotowe, od tego momentu **Opera** umożliwia sprawdzanie pisowni w polach formularzy na stronach WWW. Wystarczy w takim polu wpisać dowolny tekst i kliknąć na zielonych strzałkach w prawym dolnym rogu tegoż pola.



mgr Marian Mysior

Python w Studiu

1. Wprowadzenie

1.1. Język Python

Python jest obiektowym językiem programowania używanym na różnych platformach (Win32, Unix, MacOS, Solaris, PalmOS i innych). Jest językiem interpretowanym, a więc łatwo może być przenoszony między różnymi systemami. Jest darmowy i powszechnie dostępny (przez Internet). Z jego pomocą można pisać zarówno niewielkie skrypty jak i duże aplikacje. Można łączyć go z innymi językami (bibliotekami napisanymi w innych językach). Kod *Pythona* można zagnieżdżać w HTML i XML.

Python powstał w 1989 r. Jego twórcą jest holender Guido van Rossum. Nazwę języka nie należy kojarzyć z gatunkiem węża, lecz z serialem telewizyjnym „Latający cyrk Monty Pythona", którego sympatykiem jest jego autor. Obecnie kod źródłowy Pythona jest rozwijany, modyfikowany, poprawiany i testowany przez wiele osób. W 1998 roku powstało *Python Consortium*. Jego celem jest wprowadzanie poprawek, modyfikacji, normalizacja i publikowanie kolejnych wersji języka.

Główną zaletą *Pythona* jest jego wieloplatformowość. Skrypt napisany pod kontrolą systemu *MS Windows* będzie działał zarówno pod *Linuxem,* pod *MacOS*, jak i pod wieloma innymi systemami operacyjnymi. Liczba dystrybucji interpretatora języka dla różnych systemów ciągle zwiększa się. *Python* jest dołączany do różnych wersji *Linuksa* i *Solarisa*. W *MS Windows* (oprócz standardowego uruchamiania) może być również używany przez systemowy interpretator *Windows Script Host*, obok *JavaScript* i *Visual Basic Script*.

Dużym atutem języka *Python* jest jego prosta składnia i łatwość stosowania obiektowego stylu programowania.

1.2. Wersje

Oficjalna strona internetowa języka *Python*, www.python.org jest miejscem, skąd można pobrać interpreter języka (dla różnych platform) wraz dokumentacją, bibliotekami, przykładami i różnymi rozszerzeniami. W chwili przygotowywania ćwiczeń dostępna była wersja 2.5.2 i wersja testowa 3.0a5.

(Ciąg dalszy na stronie 27)

Python w Studiu

(Ciąg dalszy ze strony 26)

Najprostszym sposobem uruchamiania poleceń języka *Python* jest praca w trybie interakcyjnym. Wpisujemy wtedy, w wierszu poleceń instrukcje, które są natychmiast interpretowane. Ewentualne wyniki również wyświetlane są w tym trybie.

Alternatywą dla trybu natychmiastowego jest uruchamianie interpretacji wcześniej napisanych i zapisanych na dysku skryptów.

Pisanie interpretowanych skryptów jest wygodną formą programowania, ale nie zawsze wystarczającą. Wychodząc naprzeciw potrzebom unifikacji bibliotek programistycznych i graficznego wspomagania programowania powstała wersja *IronPython*, wykorzystująca platformę Microsoft .NET Framework. Dalszą ewolucją *IronPythona* jest wykorzystanie dobrodziejstw graficznego wspomagania projektowania, jaką zapewnia Microsoft Visual Studio 2008. Tak powstało *IronPython Studio*. Obecnie dostępna jest jego wersja 1.0.

1.3. Visual Studio

Visual Studio .NET integruje wiele środowisk programistycznych w postaci jednolitego interfejsu (*IDE*). Udostępnia środowisko, z którego korzystać mogą różne języki programowania. Microsoft dostosował do platformy *.NET*, od lat znany i popularny, język *Visual Basic*, zmieniając go generalnie. *VB .NET* jest w pełni obiektowym językiem programowania. Dla użytkowników języków rodziny *C* stworzono na platformie *.NET* nowy język *C#* („*C sharp*”). Jest to nowoczesny, obiektowy język programowania, używający wielu konstrukcji języka *C++*, ale bez zbędnych „naleciałości historycznych”. Język *Visual C++* jest również obecny na tej platformie. Teraz dołączył do nich język *Python*.

Wybór języka jest rzeczą drugorzędną - to jedna z zalet środowiska .NET. Integracja wielu języków w jednym środowisku powoduje, że nie musimy tracić czasu na uczenie się od początku nowych języków, aby tworzyć, uruchamiać i rozwijać aplikacje. Visual C# i Visual Basic na platformie .NET dają nam podobne możliwości. Użycie zmiennych, klas, ich właściwości i metod jest prawie identyczne w obydwu językach. Środowisko .NET (tzw. Framework) oferuje wspólne typy danych (Common Type System) i biblioteki klas (.NET Class Framework).

Potrzeby obecnych użytkowników oprogramowania wymagają, aby aplikacje dysponowały graficznymi interfejsami (były „okienkowe”). To z kolei rodzi wymagania względem środowisk programistycznych, które muszą być efektywne. Archaicznym podejściem do

(Ciąg dalszy na stronie 28)

Python w Studiu

(Ciąg dalszy ze strony 27)

nauki programowania jest wykorzystywanie narzędzi, które nie zapewniają wspomagania graficznego w tworzeniu interfejsów, nie preferują programowania obsługującego zdarzenia i programowania obiektowego. Te pojęcia nie muszą oznaczać „wyższego wtajemniczenia" w sztukę programowania. Nie należy klasyfikować ich, jako zbyt zaawansowane. Od nich musimy zaczynać nie tylko programowanie, ale już analizę problemu i projektowanie rozwiązania. Języki platformy .NET umożliwiają jak nigdy dotąd tworzenie programów obiektowych, sterowanych zdarzeniami i silnie wspomaganych graficznie. Od samego początku wykorzystujemy tutaj (przynajmniej pasywnie) właściwości i metody obiektów, nawet nie wspominając o nich. W sytuacjach, gdy interfejs graficzny jest zbędny lub przeszkadza w „zgłębieniu istoty problemu" można zbudować aplikację konsolową.

Platforma .NET to nie tylko języki programowania. W wielu sytuacjach języki pełnią rolę pomocniczą, służąc do zapisywania poleceń. W Visual Studio .NET znajdziemy Server Explorer, za pomocą którego możemy tworzyć tabele baz danych i łączyć je relacjami. Zbiór klas ADO .NET umożliwia z kolei zarządzanie bazami w tradycyjnym środowisku połączonym jak i w bardziej efektywnym i nowoczesnym środowisku rozłączonym. Do wykonania szeregu zadań, jak zapytania i modyfikacje relacyjnych baz danych wykorzystywać możemy popularny język SQL. Dane zapisywać możemy w coraz bardziej popularnym, niezależnym od platformy formacie XML.

Platforma .NET oferuje również (może przede wszystkim) narzędzia do budowania aplikacji internetowych i zarządzania zasobami rozproszonymi. Biblioteka *ASP .NET* umożliwia tworzenie formularzy WWW. Usługi *Web Services* umożliwiają tworzenie nowoczesnych aplikacji rozproszonych.

Platforma .NET stanowi środowisko uruchomieniowe, bibliotekę klas, kompilatory i narzędzia pomocnicze. Programy napisane w środowisku *.NET Framework* nie są kompilowane od razu do kodu maszynowego danego procesora, lecz do kodu pośredniego *MSIL* (*Microsoft Intermediate Language*). Uruchomienie programu wymaga środowiska *CLR* (*Common Language Runtime*), które jest częścią *.NET Framework*. Podczas pierwszego uruchomienia programu na maszynie docelowej kompilator *Just In Time* tworzy kod maszynowy specyficzny dla danego procesora.

W systemach operacyjnych Windows niezbędne jest zainstalowanie pakietu *.NET Framework*, dostępnego bezpłatnie na stronach internetowych Microsoftu. Środowisko to

(Ciąg dalszy na stronie 29)

Python w Studiu

(Ciąg dalszy ze strony 28)

wystarcza jako minimum, do pisania, kompilowania i uruchomiania programów w języku *IronPython*. Aby jednak efektywnie tworzyć kod źródłowy (korzystać z graficznego wspomagania i wielu innych narzędzi wspierających), kompilować go i wykrywać błędy, warto zaopatrzyć się w środowisko zintegrowane *Visual Studio 2008 Shell Runtime* i *IronPython Studio*.

2. Środowisko programowania

2.1. IronPython Studio – instalacja

Środowisko zintegrowane IDE (Integrated Development Environment) *IronPythonStudio* można pobrać bezpłatnie ze strony: www.codeplex.com/IronPythonStudio. Plik instalacyjny znajdziemy pod nazwą IronPythonStudioIsolatedSetup lub IronPythonStudioIntegratedSetup. Przed instalacją tego środowiska należy jeszcze pobrać (również bezpłatnie) i zainstalować tzw. powłokę *Visual Studio 2008 Shell Runtime*, opartą na *Visual Studio 2008* (również dwa alternatywne pliki instalacyjne: Visual Studio 2008 Shell Isolated Mode Redistributable package i Visual Studio 2008 Shell Integrated Mode Redistributable package).

Instalacja powłoki może być trochę myląca. Po uruchomieniu jej z pliku instalacyjnego, odpowiednia zawartość zostanie rozpakowana do folderu *C:\VS 2008 Shell Redist\Isolated Mode*. Nie jest to jeszcze koniec instalacji. Dalej należy otworzyć ten folder i uruchomić właściwą instalację z pliku *vsshellisolated_enu.exe*.

Dysponując płatną wersją *Visual Studio 2008* wystarczy zainstalować tylko plik IronPythonStudioIntegratedSetup.

2.2. Tryby pracy

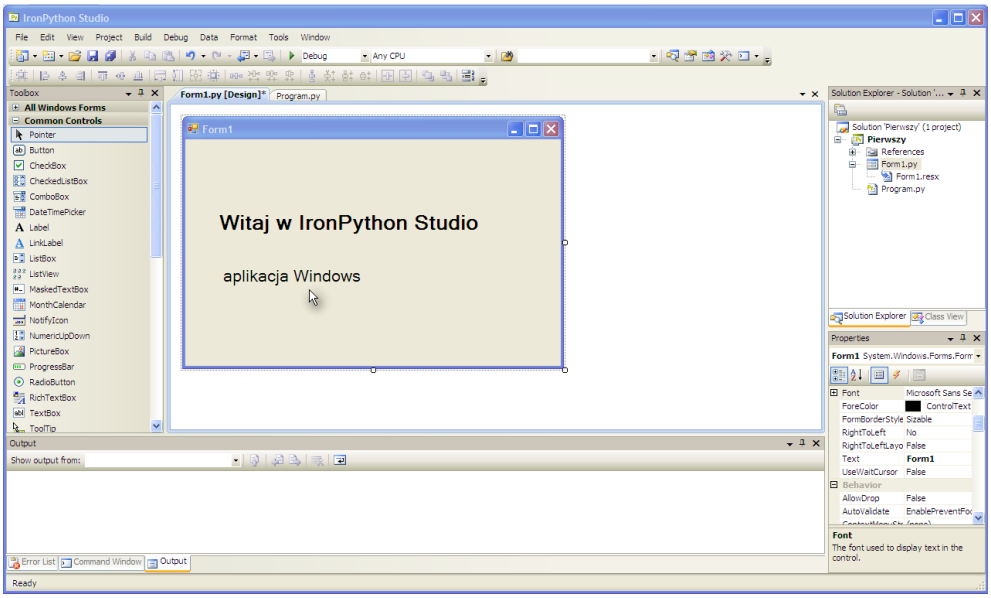
W środowisku zintegrowanym *IronPython* Studio możemy używać kilku trybów pracy lub inaczej wykorzystywać różne szablony.

Podstawowym szablonem jest *Windows Application*, służący to tworzenia programów okienkowych. W tym trybie otrzymujemy silne wsparcie w postaci automatycznego utworzenia kodu wstępnego, graficznie narzędzia w postaci kontrolek z przybornika *Toolbox* i podpowiedzi podczas pisania kodu (*Intellisense*).

(Ciąg dalszy na stronie 30)

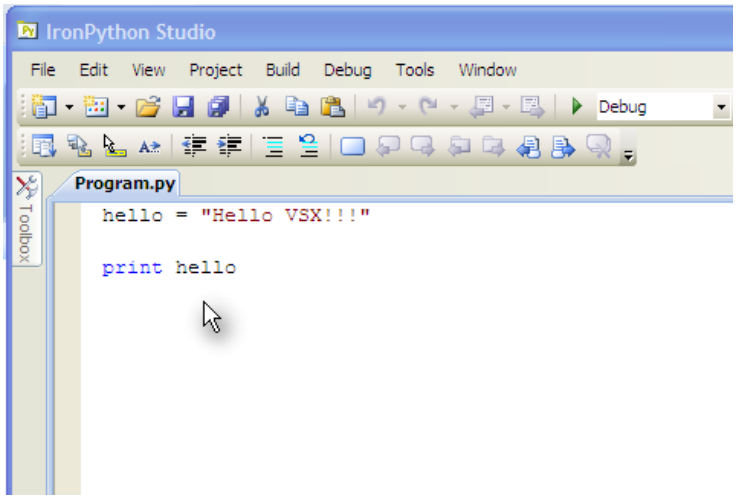
Python w Studiu

(Ciąg dalszy ze strony 29)



Rysunek 2.1 Windows Application

Gdy zbędny jest graficzny interfejs użytkownika, aplikacje możemy tworzyć korzystając z szablonu konsolowego: *Console Application*.



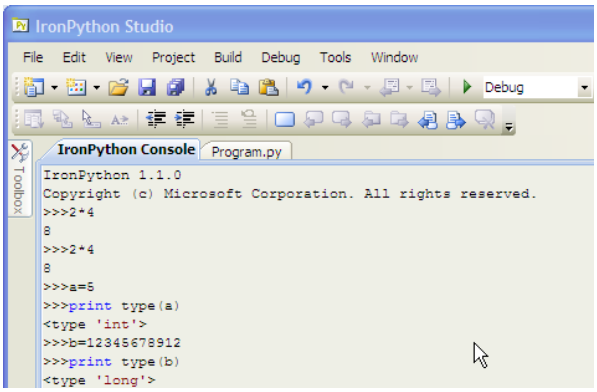
Rysunek 2.2 Console Application

Python znany jest z możliwości pracy w trybie natychmiastowym. W środowisku *IronPython Studio* odnajdziemy to okno pod nazwą *IronPython Console*.

(Ciąg dalszy na stronie 31)

Python w Studiu

(Ciąg dalszy ze strony 30)



Rysunek 2.3 Tryb natychmiastowy

3. Pierwsze próby

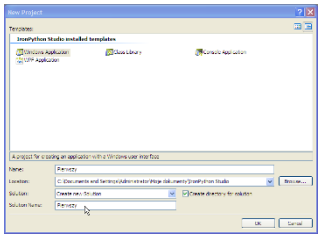
Środowiska zintegrowane zwalniają programistę z „ręcznego” pisania całego kodu, oferując graficzne wspomaganie podczas wybierania typu aplikacji i wstawiania poszczególnych komponentów. Znacząco przyspiesza to cały proces tworzenia programu i zwalnia programistę pamięciowego opanowywania wielu szczegółów.

Ćwiczenie 1. Zaczynamy

Uruchomimy *IronPython Studio* i klikniemy w pierwszy przycisk na pasku narzędzi, czyli w *New Project*.

W oknie dialogowym *New Project* wybierzemy typ projektu (*Templates*) – *Windows Application*, określimy własną nazwę projektu (zostanie automatycznie utworzony katalog o podanej nazwie) i wybierzemy lokalizację na dysku.

Po otwarciu projektu zobaczymy projekt formularza i kilka innych okien (*Rysunek 2.1*). Projekt formularza będzie oknem naszej aplikacji.



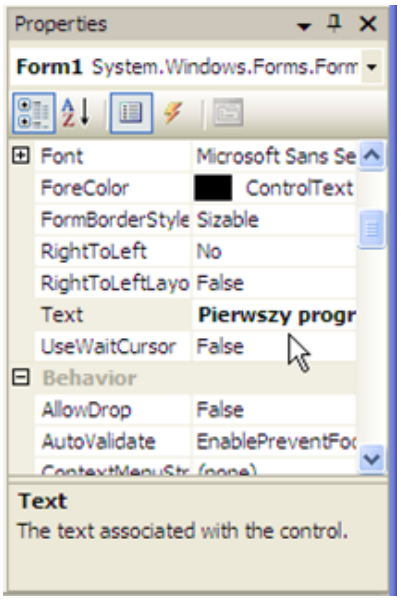
Rysunek 3.1 New Project

(Ciąg dalszy na stronie 32)

Python w Studiu

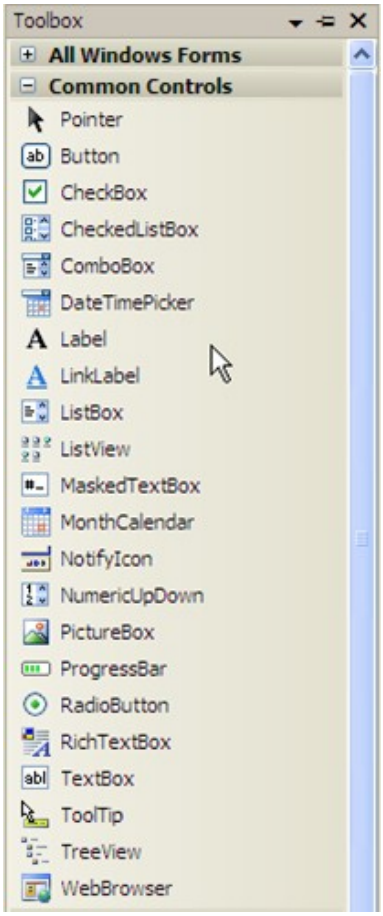
(Ciąg dalszy ze strony 31)

W prawym dolnym narożniku ekranu znajdziemy okno właściwości (*Properties*), w którym wybierzemy właściwość formularza: *Text*. Wpiszemy tam „Pierwszy program”. Będzie to tytuł naszego formularza.



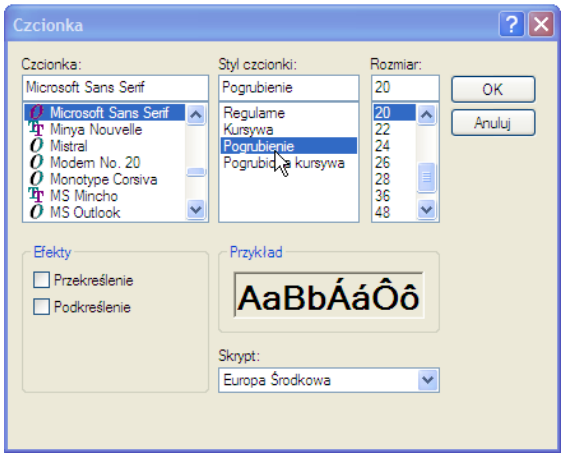
Rysunek 3.2 Okno Properties

Na formularzu umieścimy napis (etykietę): „Witaj w IronPython Studio”. Zrobimy to wykorzystując przybornik (*Toolbox*), zawierający różne kontrolki (inaczej formanty). Po rozwinięciu przybornika wybierzemy kontrolkę *Label* i naniesiemy ją na formularz (przeciągając myszką). W oknie *Properties*, które automatycznie wyświetla właściwości wybranego formantu. Odszukamy pozycję *Text* i wpiszemy swój tekst.



Rysunek 3.3 Fragment przybornika *Toolbox*

Możemy jeszcze zadbać o jego wygląd. Właściwość *Font* w oknie *Properties* pozwala nam wybrać atrybuty czcionki (rodzaj, wielkość, kolor itp.). Po kliknięciu na tej właściwości najlepiej wybrać przycisk […]. Otwiera on okno dialogowe Czcionka (podobne jak w edytorach tekstu).



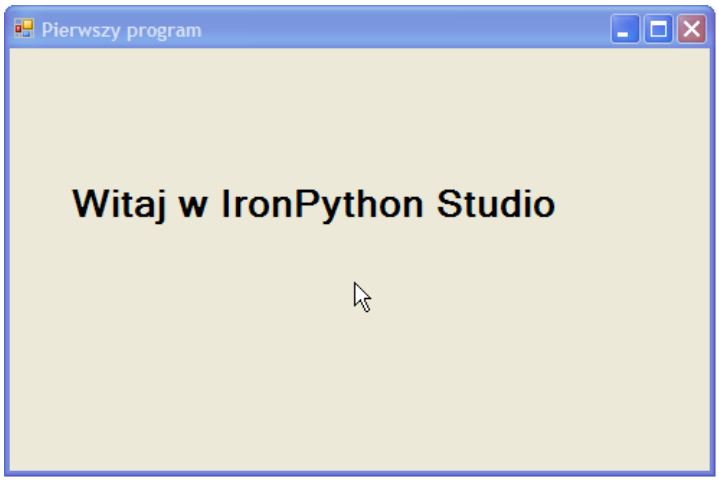
Rysunek 3.4 Ustawianie właściwości czcionki

(Ciąg dalszy na stronie 33)

Python w Studiu

(Ciąg dalszy ze strony 32)

Nasz pierwszy program jest gotowy. Utworzyliśmy go korzystając jedynie z narzędzi graficznych, nie pisząc kodu „ręcznie”. Możemy teraz wypróbować swój produkt, uruchamiając go ze środowiska zintegrowanego. W tym celu wybieramy przycisk *Start Debugging* (4) z paska narzędzi. Alternatywne sposoby to *Debug/Start Debugging*, lub klawisz F5, *Debug/ Start Without Debugging* lub *Ctrl +F5*. Warto polecić ostatni sposób pomijający śledzenie programu (spowalniające kompilację i uruchomienie). Program można oczywiście uruchamiać spoza środowiska zintegrowanego. Odnajdziemy go w swoim projekcie w odpowiednim podkatalogu.



Rysunek 3.5 Pierwszy program po uruchomieniu

Ważną funkcję w środowisku *IronPython Studio* spełnia okno *Solution Explorer* (*Explorator Rozwiązań*). Znajdziemy w nim pliki naszego projektu. Podwójne kliknięcie myszką we wskazany plik, otwiera go do edycji.

Ćwiczenie 2. Podstawowe kontrolki

Utworzymy nowy projekt I nazwiemy go „Kontrolki”.

Projekt formularza *Form1* nieco rozciągniemy, dostosowując go do pożądanej wielkości. Jego właściwości *Text* (w oknie *Properties*) nadamy wartość: „Podstawowe kontrolki”.

Z przybornika *Toolbox* wstawimy na formularz etykietę *Label*. Zmienimy jej właściwość *Text* na napis: „Podaj swoje imie:”. Niestety literę „e” w ostatnim słowie musimy poprawić na „ę” kodzie źródłowym. Zrobimy to później. Korzystając z

(Ciąg dalszy na stronie 34)

Python w Studiu

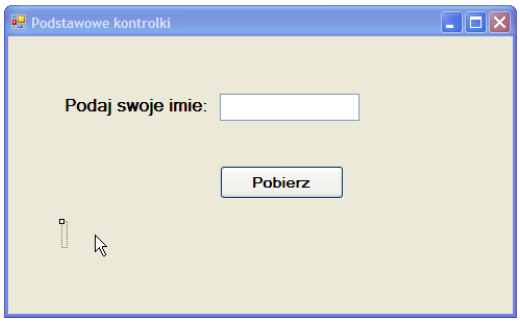
(Ciąg dalszy ze strony 33)

właściwości *Font* zmienimy rozmiar czcionki na 12 pkt. Możemy ją dodatkowo pogrubić.

Do wprowadzania danych w czasie działania programu służy kontrolka *TextBox*. Pobierzemy ją z przybornika *Toolbox* i wstawimy na formularz. Zwróćmy uwagę na automatyczne wyrównywanie (w pionie i w poziomie) kontrolek na formularzu. Kontrolkę umieścimy z prawej strony etykiety. Dostosujemy do potrzeb jej rozmiary i atrybuty czcionki (okno *Properties*).

Kolejna kontrolka służyć będzie do pobrania wprowadzonej wcześniej wartości z okna *TextBox* i wyprowadzenia napisu powitalnego. Jest to *Button*, czyli przycisk. Umieścimy go poniżej poprzednich kontrolek i nadamy mu napis (właściwość *Text*): „Pobierz”. Możemy również sformatować czcionkę przycisku według własnego uznania.

Do wyprowadzenia napisu powitalnego użyjemy drugiej kontrolki *Label*. Zwróćmy uwagę na automatyczne nadawanie nazw umieszczanym na formularzu kontrolkom. W naszym przypadku będzie to: *label2*. Dobrą praktyką jest nadawanie poszczególnym kontrolkom własnych nazw, odzwierciedlających ich przeznaczenie. Szczególnie ważne jest to, gdy zachodzi obawa pomyłek w odwołaniach do kontrolek z poziomu kodu. Domyślną wartość *label2* właściwości *Text* usuniemy, pozostawiając pusty łańcuch. Uwaga: nie można mylić właściwości Text z właściwością *Name*. Obie mają identyczne wartości domyślne. W naszym przypadku jest to *label2*.



Rysunek 3.6 Projekt formularza

(Ciąg dalszy na stronie 35)

Python w Studiu

(Ciąg dalszy ze strony 34)

Czas na pisanie kodu! Graficzne wspomaganie, jakiego doświadczamy nie wyręczy nas w całości od „ręcznego” tworzenia kodu. Przyciski *Button* często wykorzystuje się do kojarzenia z nimi obsługi zdarzeń w systemie operacyjnym. Zainicjujemy „edycję metody obsługi zdarzenia – przy kliknięciu”. Zrobimy to podwójnym kliknięciem w przycisk na projekcie formularza. Otworzy się okno *Form1.py* z kodem utworzonym do tego momentu w pełni automatycznie.

Szkielet aplikacji, tworzony automatycznie, może być częściowo (lub w całości) niezrozumiały dla początkującego programisty, jednakże szczegółowa jego znajomość nie jest konieczna na tym etapie nauki programowania.

Skoncentrujemy się na napisaniu kodu obsługi zdarzenia „przy kliknięciu na przycisku Button1”. Jego treść jest teraz następująca:

```
def _button1_Click(self, sender, e):
    pass
```

Słowo kluczowe def rozpoczyna definicję metody (funkcji).

`_button1_Click(self, sender, e)` to nazwa i parametry tej funkcji.

Uwaga: definiując metody obsługi zdarzeń nie możemy poprawiać ich nazw i parametrów utworzonych automatycznie.

Dwukropek (:) kończy nagłówek funkcji

Słowo kluczowe pass „oznacza nie rób nic”. Jest tu wypełniaczem, ponieważ definicja funkcji musi mieć treść (ciało), nie może kończyć się dwukropkiem. W innych językach programowania mamy tutaj np. { }, czy Begin End.

Uzupełnimy treść metody. Docelowo powinna być następująca:

```
def _button1_Click(self, sender, e):
    self._label2.Text="Witaj "+self._textBox1.Text+"!"
```

„Wcięcia”, inaczej przesunięcie kolejnego wiersza względem poprzedniego w języku Python pełnią rolę „zagnieżdżenia”. Są więc konieczne. Pełnią podob-

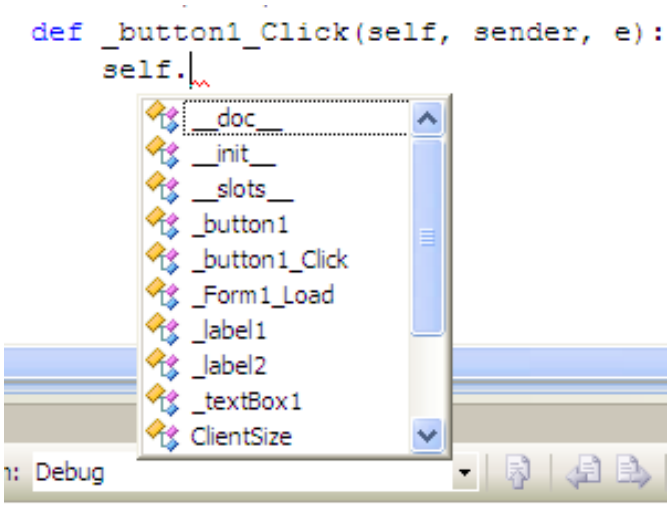
(Ciąg dalszy na stronie 36)

Python w Studiu

(Ciąg dalszy ze strony 35)

ną rolę jak np. nawiasy {} w językach rodziny C. Definicja metody kończy się w miejscu, gdzie kolejny wiersz jest bez przesunięcia względem słowa *def* rozpoczynającego definicję.

Słowo self jest nazwą kwalifikowaną, oznaczającą odniesienie się do atrybutów własnego (tego) obiektu. Etykieta jest tutaj atrybutem formularza, czyli obiektu *Form1*. Pisząc kropkę po tym słowie obserwujmy zadziałanie mechanizmu podpowiedzi nazwanego *Intellisense*. Warto z niego korzystać, skracając sobie pisanie kodu i unikając błędów literowych. Pomoc ta zwalnia nas również od pamięciowego opanowywania szeregu atrybutów.



Rysunek 3.7 Mechanizm automatycznego uzupełniania *Intellisense*

Nazwy kontrolek (obiektów) w *IronPythonie* zaczynają się od znaku podkreślenia „_”.

Łańcuchy znaków (napisy) umieszczamy w cudzysłowu lub pomiędzy apostrofami (parami takie same).Łączymy je natomiast znakami (operatorem) „+”.

Pozostało nam skorygować polską pisownię w zwrocie „Podaj swoje imie”. W automatycznie utworzonym kodzie poprawimy wiersz:

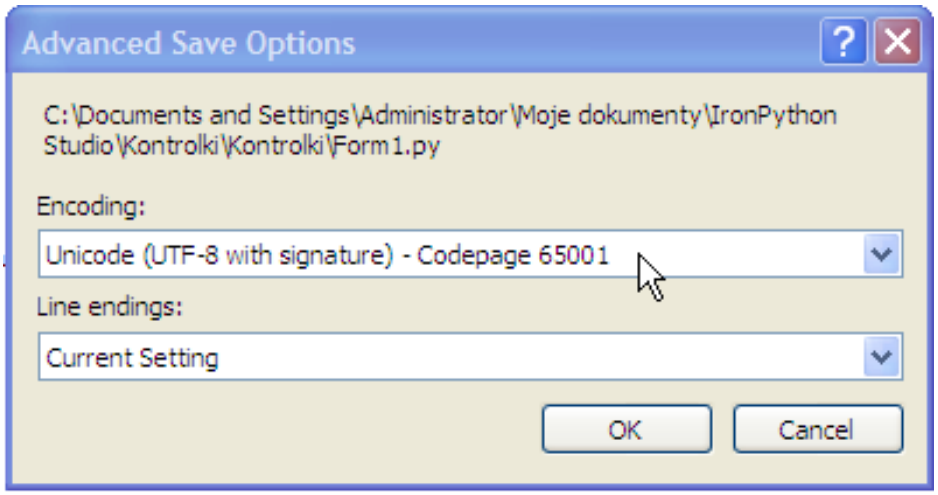
```
self._label1.Text = 'Podaj swoje imię:'
```

(Ciąg dalszy na stronie 37)

Python w Studiu

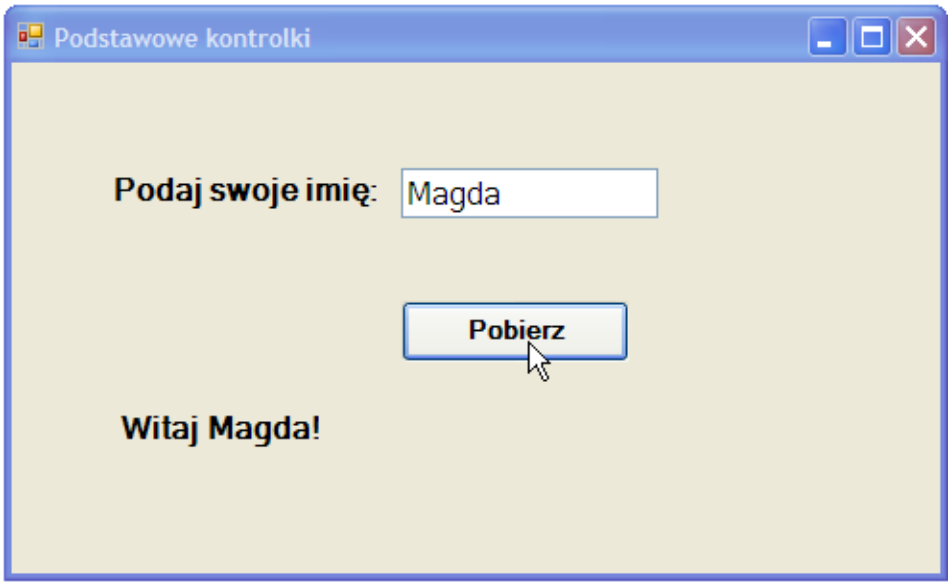
(Ciąg dalszy ze strony 36)

Aby zmiana była trwała należy jeszcze wybrać lokalizację: *File/Advanced Save Options…* i w otwartym oknie dialogowym ustawić opcję *Encoding*: *Unicode (UTF8 with signature) – Codepage 65001*.



Rysunek 3.8 Ustawienie sposobu kodowania polskich znaków

Możemy wypróbować działanie programu, uruchamiając go (Ctrl-F5).



Rysunek 3.9 Program „Podstawowe kontrolki”

mgr Roland Zimek

Pełne wersje programów

Wakacyjne wydania popularnych czasopism komputerowych nie oszczędzają na ciekawych propozycjach darmowych wersji programów. Niewątpliwie więcej jest „lżejszych" gatunkowo programów, ale pośród nich jest kilka wartych polecenia.

CHIP 7/2008 oferuje nie tylko ciekawy program umożliwiający tworzenie stron WWW – **WebSite X5 Smart Editio** – ale także bogaty zestaw dodatków i szablonów, wzbogacających wygląd tworzonych stron – **WebSite X5 Templates**.

PC Format 8/2008 oferuje program **BB FlashBack Express 2.0**. Umożliwia on zgrywanie czynności wykonywanych na ekranie, a następnie opatrzenie takiego materiału własnym komentarzem lub ilustracjami. Tak przygotowany materiał można później zapisać jako film w postaci AVI lub w postaci prezentacji multimedialnej Flash. Doskonała propozycja dla osób przygotowujących filmy instruktażowe.

Bardziej zaawansowanym użytkownikom do gustu przypadnie *PC Word Komputer 8/2008*. Na dołączonej płycie DVD otrzymamy **CodeGear Delhi for PHP 2.0**, który niewątpliwie powinien zainteresować twórców stron WWW, chcących je wzbogacić o aplikacje napisane w języku PHP.

Razem *PC Word Komputer 9/2008* otrzymamy interesujący program **endorphin 2.7 LE**, przy pomocy którego można tworzyć i animować realistycznie wyglądające postaci. Tak wykorzystane animacje można później wykorzystać w filmach lub grach. Osobą chcącym zwiększyć możliwości swojego umysłu, można polecić program **Mistrz Pamięci**. Przy jego pomocy poznamy metody szybkiego uczenia się i zapamiętywania.

Jeżeli posiadasz skaner do którego nie dołączono oprogramowania OCR, kup miesięcznik *NEXT 9/2008*. Na dołączonej płycie DVD znajdziesz **ABBYY Fine Reader 6.0 Sprint**. Jest to bardzo dobry program umożliwiający rozpoznanie zeskanowanego tekstu i przekształcenie go do postaci którą można następnie edytować np. w programie Microsoft Word.

W numerze NEXT 8/2008, zainteresuje tych, którzy zajmują się obróbką materiałów wideo. Znajdziemy tam bowiem program **Plato DVD Creator 3.79** umożliwiający pracę na plikach wideo zapisanych w jednym z najpopularniejszych formatów AVI i zapisanie ich w formacie wideo DVD. Numer *PC Format 9/2008* zawiera program **Anyplace Control 4.5 Basic**, umożliwiający zdalną kontrolę nad komputerami.

Osoby przygotowujące się do zdawania egzaminu na prawo jazdy, powinny zainteresować się programem **Symulator Prawa Jazdy 3D**, wydanym w *PC Word Komputer 10/2008*. Daje on możliwość sprawdzenia naszych umiejętności potrzebnych w trakcie prowadzenia pojazdu. Innym ciekawym programem dostępnym w tym samym wydaniu, jest **Aimer DVD Studio Pack**. Umożliwia on zgrywanie materiału wideo z płyt DVD i konwersję do innego formatu. Przydatny także okaże się niewątpliwie opcja umożliwiająca samodzielne przygotowanie krążka z filmami nagranymi na DVD.

mgr Witold Rudolf

Wprowadzenie do programowania obiektowego na przykładzie języka Turbo Pascal

Dlaczego programowanie obiektowe?

Programowanie obiektowe zadomowiło się na stałe w technice programistycznej. Pozwala na wydajniejsze (czytaj: szybsze) przygotowanie aplikacji, korzystanie z gotowych modułów. Technologia .NET pozwoliła na wykorzystywanie modułów obiektowych przygotowanych w dowolnym języku programowania. Jest zatem naturalne, że na lekcjach informatyki mówiąc o programowaniu powinniśmy zająć się programowaniem obiektowym.

W praktyce szkolnej bardzo często spotykamy się jeszcze z nauczaniem informatyki z wykorzystaniem języka Turbo Pascal. Często nauczyciele *pascalowcy* są piętnowani jako „zabytki". Jak dla każdego języka programowania, tak i dla Pascala można znaleźć argumenty zarówno „za", jak i „przeciw" wykorzystaniu go w nauczaniu jako *języka pierwszego kontaktu*. Faktem jest, że programy nauczania bazują na programowaniu strukturalnym, nie obiektowym. Dyskusja „za czy przeciw" nie jest tematem tego artykułu – ma on pokazać, że wykorzystując Turbo Pascal można wprowadzić uczniów w świat programowania obiektowego. A czy jego opanowanie jest trudniejsze od strukturalnego? Spróbujemy się na tych stronach wykazać, że niekoniecznie.

Pojęcie i przykłady obiektów

Treści, które spróbujemy tu przekazać, nie będą kursem programowania, a już na pewno nie programowania obiektowego. Mają na celu przedstawienie, a w zasadzie uzmysłowienie Ci cech Twojego sposobu postrzegania świata, który stał się przyczyną powstania tzw. *programowania obiektowego*. Do opisu przedstawianych tu pojęć będziemy używać pewnego języka symbolicznego podobnego do Object Pascala.

(Ciąg dalszy na stronie 40)

Wprowadzenie do programowania obiektowego...

(Ciąg dalszy ze strony 39)

W codziennym postrzeganiu świata nieświadomie dokonujemy uogólnień i klasyfikowania postrzeganych obiektów. Jeżeli zastanowimy się przez chwilę, każdy z pewnością przyzna, że:

1) dokonujemy różnicowania postrzegania na obiekty i ich atrybuty - bez problemu odróżniamy drzewo od lodówki i potrafimy określić cechy charakterystyczne dla drzew i lodówek
2) rozróżniamy obiekty i ich części składowe - czyż drzewo nie składa się z pnia i gałęzi?
3) tworzymy klasy obiektów i bez trudu je rozróżniamy - drzewa (klasa drzew), ryby (klasa ryb).

Co istotne, w języku potocznym nie rozróżniamy klasy od obiektu, który należy do klasy. Typowy błąd wytykany na lekcjach matematyki: polecenie *narysuj kwadrat* jest niemożliwe do wykonania, gdyż kwadrat jako figura geometryczna jest pojęciem abstrakcyjnym (platońską ideą?). Narysowany kwadrat jest konkretnym przykładem kwadratu. Jest realnym obiektem. Kwadrat jako figura nie ma np. długości boku - narysowany tak.

Wracamy zatem do platońskiej koncepcji idei? A może po prostu tak postrzegamy świat? Spójrz na krzesło: bez trudu odróżnisz je od innych mebli. Zatem: co to jest krzesło? Spróbujmy zbudować abstrakcyjne pojęcie KRZESŁA - nazwijmy je **klasą KRZESŁA**. Jakie cechy wspólne mają wszystkie krzesła? Jakie są ich atrybuty?

Atrybuty:

1) przedmiot martwy
2) ma co najmniej jedną nogę (odrzucamy pufy itp)
3) ma oparcie (bez oparcia - to np. taboret)
4) przeznaczone dla jednej osoby (tak wykluczamy np. kanapę)
5) twarde lub lekko miękkie (wykluczamy fotele)
6) do siedzenia

(Ciąg dalszy na stronie 41)

Wprowadzenie do programowania obiektowego...

(Ciąg dalszy ze strony 40)

W tak zdefiniowanej klasie KRZESŁA mieszczą się wszelkie: drewniane, metalowe, składane, obrotowe itd. Uwaga - właśnie zbudowaliśmy klasy *potomne* od klasy KRZESŁA. Dziedziczą one wszystkie atrybuty tej klasy, ale np. klasa KRZESŁA DREWNIANE dodaje do nich swój własny atrybut: drewniane. Oczywiście możnaby dalej tworzyć kolejne klasy dodając szczególne atrybuty: ilość nóg, z oparciem na ręce lub bez, wyścielane lub nie. Mebel, na którym siedzisz jest natomiast konkretnym obiektem z którejś zdefiniowanej podklasy, ale oczywiście także z klasy KRZESŁA.

Gdy przyjrzysz się otaczającym nas obiektom i chwilę się zastanowisz, zdasz sobie sprawę z tego, że ciągle budujesz klasy: DRZEWA i KRZEWY, KWIATY i KAMIENIE - każdy bez trudu je rozróżni. Spróbuj więc określić atrybuty charakterystyczne dla każdej z tych klas. Pamiętaj, że klasa ma zawierać *wszystko i nic więcej* - czyli określone atrybuty musi posiadać każdy obiekt należący do danej klasy, natomiast obiekt spoza klasy nie posiada co najmniej jednego z atrybutów.

Rozważmy przykład bardziej sterylny, geometryczny. Rozważmy klasę CZWOROKĄTY. Atrybuty tej klasy to:

1) figura geometryczna
2) ma cztery kąty
3) jest figurą zamkniętą

Klasą *potomną* do CZWOROKĄTY jest klasa PROSTOKĄTY. Do atrybutów swego „ojca” dodaje ona:

1) ma wszystkie kąty równe
2) ma boki parami równe

Domyślasz się dalszego ciągu? KWADRAT do potomek PROSTOKĄTA, który dodatkowo ma:

1) wszystkie boki równe

(Ciąg dalszy na stronie 42)

Wprowadzenie do programowania obiektowego...

(Ciąg dalszy ze strony 41)

Zauważ, że klasa KWADRAT ma także wszystkie atrybuty odziedziczone po KWADRACIE, a więc i po CZWOROKĄCIE. Zapiszmy to tak:

CZWOROKĄT --> PROSTOKĄT --> KWADRAT

Jeśli chwilę się zastanowisz, to samodzielnie wyprowadzisz klasy np.

SAMOCHÓD --> SAMOCHÓD OSOBOWY --> POLONEZ

Uwaga: klasa POLONEZ definiuje wszystkie samochody tego typu - pojazd stojący przy krawężniku jest natomiast konkretnym obiektem tej klasy.

Przy okazji zauważmy, że w świecie rzeczywistym nie występuje jedynie proste zjawisko dziedziczenia atrybutów klas. Mamy także do czynienia z *dziedziczeniem krzyżowym*, które pominiemy w dalszych rozważaniach. Oczywistym jest wszakże, że klasa AMFIBIA jest potomkiem rozłącznych klas SAMOCHÓD oraz ŁÓDŹ.

Możemy już chyba stwierdzić, że *klasa* jest pojęciem abstrakcyjnym, ideą, opisem, natomiast *obiekt* jest skonkretyzowanym elementem danej klasy. Na przykład narysowany na kartce papieru zielonym flamastrem kwadrat 5x5 cm jest obiektem klasy KWADRAT, rosnący za oknem kasztanowiec z pękającymi kulami kasztanów obiektem klasy DRZEWO (czyż każde drzewo nie jest niepowtarzalnym obiektem? Czyż każdy człowiek nie jest indywidualnością?...).

Większość z tworzonych przez nas klas posiada jeszcze jeden dodatkowy element - **metodę** korzystania z obiektów tej klasy lub sposób ich zachowania. Konieczne będzie przy tym zdecydowanie się, które funkcje są istotne. Czy jest dla nas ważna, że DRZEWA rosną, czy że gubią liście jesienią? A może oba te zachowania? Czy ważne jest wykorzystywanie krzesła do siedzenia, czy możliwość jego przestawiania? A może obie metody wykorzystania? Wybór *metod* charakterystycznych dla danej klasy będzie rzutował na obiekty reprezentujące daną klasę. Jeżeli ustalimy, że jedyną metodą korzystania z obiektów klasy KSIĄŻKA jest tylko oglą-

(Ciąg dalszy na stronie 43)

Wprowadzenie do programowania obiektowego...

(Ciąg dalszy ze strony 42)

danie obrazków, to na trzymaną w ręce książkę (realny obiekt) będziemy patrzyli jak trzyletnie dziecko - tekst dla nas nie istnieje. Korzystać z danego obiektu możemy wyłącznie poprzez jego metody. Trzeba zatem uważać, żeby nie zaistniała sytuacja, w której nie będziemy mogli przeczytać książki (por. poprzedni przykład). W opisie (definicji) klasy dobór metod jest zatem bardzo istotny.

Definiowanie klasy obiektów

Spróbujemy zapisać bardziej formalnie nasze dotychczasowe przemyślenia. Jako przykład klasy przyjmiemy klasę C_Figures, z której wyprowadzimy klasy potomne opisujące kolejne figury geometryczne. Docelowo spróbujemy otrzymać pełny działający program komputerowy kreślący różne figury i pozwalający poruszać nimi po ekranie komputera. Będzie to dodatkowa atrakcja w naszej pracy, będzie także rzutowało na nasze spojrzenie na definiowane figury. Jako języka opisu użyjemy Turbo Pascala, który w wersji 5.5 jest już bezpłatnie rozpowszechniany prze Borlanda. Równie dobrze mógłby to być inny język obiektowy: C++, Java - idea zawsze pozostanie ta sama, różnice będą dotyczyć szczegółów zapisu.

Zastanówmy się, jakie atrybuty ma **każda** figura geometryczna. Zakładamy oczywiście działania w geometrii euklidesowej w układzie kartezjańskim. Zgodzimy się chyba, że każda figura z klasy C_Figures:

1) zawiera co najmniej jeden punkt
2) punkt ten ma określone współrzędne w układzie współrzędnych kartezjańskich.

Opis ten obejmuje zarówno *punkt*, jak i dowolny *wielobok*. Metody działania na obiektach klasy C_Figury powinny być następujące:

1) narysowanie obiektu (*Show*)

(Ciąg dalszy na stronie 44)

Wprowadzenie do programowania obiektowego...

(Ciąg dalszy ze strony 43)

2) zmazanie obiektu (*Hide*)

3) zmiana położenia obiektu (*Move*)

Dla potrzeb przyszłego programu musimy także założyć, że każda klasa potrafi utworzyć lub zniszczyć obiekt będący realnym przedstawicielem danej klasy. Potrzebna nam będzie dodatkowa pomocnicza struktura opisująca punkt w układzie współrzędnych. Ponieważ ograniczymy się do płaszczyzny, struktura ta zawierać powinna dwie liczby całkowite: X i Y określające odpowiednie współrzędne punktu. Formalny opis tej struktury (nazwijmy ją **PointType**) wyglądać będzie następująco:

```
PointType = record
            X, Y : Integer
            end;
```

Pomińmy na razie znaczenie poszczególnych symboli. Przyjmijmy, że taki zapis dokładnie definiuje opisaną strukturę. Słowo *Integer* oznacza, że X i Y mają być liczbami całkowitymi.

Jak zatem wyglądać będzie formalny opis klasy C_Figures?

```
C_Figures = object  { jest to klasa obiektów   }
               p0 : PointType;     { każda figura musi zawierać }
                        { co najmniej jeden punkt  }
               procedure Show;     { metoda narysowania       }
               procedure Hide;     { metoda zmazania          }
               procedure Move;     { metoda poruszania        }

               constructor Init;   { metoda tworząca obiekt   }
               destructor  Done;   { metoda niszcząca obiekt  }
            end;          { koniec definicji klasy obiektów }
```

Łatwo się domyślić, że teksty w nawiasach są komentarzami objaśniającymi poszczególne deklaracje. Dodamy do tego jeszcze kilka słów wyjaśnienia. Słowo **object** określa, że dana nazwa (w naszym przypadku C_Figures) definiuje abstrakcyjną klasę obiektów o pewnych

(Ciąg dalszy na stronie 45)

Wprowadzenie do programowania obiektowego...

(Ciąg dalszy ze strony 44)

opisanych dalej atrybutach i metodach operowania obiektami. Atrybutem wspólnym dla wszystkich figur jest *punkt* - położenie każdej figury jest definiowane przez co najmniej jeden punkt (np. dla trójkąta będą to trzy punkty). Stąd w definicji klasy umieściliśmy zmienną **p0**, która będzie opisywać położenie jednego punktu, oczywiście po nadaniu pewnych wartości liczbowych współrzędnym X i Y zmiennej p0. Współrzędne te będziemy oznaczać p0.X oraz p0.Y - jest to czytelne, prawda?

Dalej w definicji klasy występują metody operowania obiektami tej klasy. Są to *procedury*, czyli zestawy wykonywanych czynności, nazwane Show, Hide i Move. Wiemy już, *co* mają robić te procedury. Pozostaje kwestia *jak?* Pojawienie się tych metod w definicji abstrakcyjnej klasy C_Figures sygnalizuje, że co najmniej takie metody będą występowały dla każdej z podklas – figur rysowanych na ekranie. Są to z natury rzeczy metody ogólne, bo jak opisać metodę zmazania dowolnej figury? Spróbujemy jednak dojść do pewnych uogólnień.

Przyjrzyjmy się metodom Show i Hide. Jak „normalnie” rysujemy dowolną figurę? Bierzemy ołówek (czarny) i *rysujemy* - cokolwiek to znaczy. A jak zmazać figurę? Bierzemy gumkę (lub ołówek w kolorze tła, efekt będzie ten sam) i znowu *rysujemy* zmazując narysowaną uprzednio linię. Chyba zgodzisz się, że gdy przyjmiemy, że ekran monitora komputera ma czarne tło, to można zapisać tak:

```
Draw(White);          rysowanie figury - Show
Draw(Black);          zmazanie figury  - Hide
```

Szczegółowy opis, jak *rysować* (Draw) będzie inny dla każdej figury, ale sposób rysowania (Show) oraz mazania (Hide) pozostanie niezmieniony! Sprawdź, że jeśli będziesz wiedział, jak robić Draw dla dowolnej figury, to zawsze metoda Show ją narysuje a Hide zmaże.

Płynie stąd wniosek, że opis metod klasy C_Figures musimy wzbogacić o procedurę Draw, przy czym o ile Show i Hide będą identycznie wykonywane dla dowolnej figury, o tyle każda

(Ciąg dalszy na stronie 46)

Wprowadzenie do programowania obiektowego...

(Ciąg dalszy ze strony 45)

z figur należących siłą rzeczy do klas potomnych C_Figures będzie musiała precyzyjnie określać swoją własną, prywatną metodę Draw. Zatem schemat działania przy rysowanie (Show) będzie następujący:

1) wykonaj metodę Show odziedziczoną po C_Figures
2) metoda Show wykorzysta w tym celu prywatną metodę Draw zdefiniowaną w danej podklasie

Jak to osiągnąć? Jak zagwarantować, że Show skorzysta z metody Draw zdefiniowanej w podklasie, a nie w klasie C_Figures? W języku Turbo Pascal należy takiej metodzie nadać atrybut *virtual*. Zatem ostatecznie wypracowany opis klasy C_Figures wyglądać będzie tak:

```
C_Figures = object       { jest to klasa obiektów   }
        p0 : PointType;        { każda figura musi zawierać }
                                    { co najmniej jeden punkt }
        procedure Draw(color: Integer);   virtual;
                                    { kreślenie figury }
        procedure Show;     { metoda narysowania }
        procedure Hide;     { metoda zmazania      }
        procedure Move;     { metoda poruszania }

        constructor Init;     { metoda tworząca obiekt }
        destructor  Done;     { metoda niszcząca obiekt }
      end;        { koniec definicji klasy obiektów }
```

Procedura Draw wymagać będzie (jak w przykładzie) podawania numeru (nazwy) koloru. Operować będziemy na razie kolorami Black i White. Atrybut *virtual* zagwarantuje, że zawsze będzie wykorzystywana metoda Draw z klasy, do której należeć będzie tworzony obiekt. Metodę taką nazywać będziemy *wirtualną*. Brak słowa *virtual* spowodowałby wykorzystywanie przez metody Show i Hide metody Draw z tej klasy, do której należą metody Show i Hide. Musielibyśmy zatem w każdej podklasie definiować identyczne metody Show i Hide

(Ciąg dalszy na stronie 47)

Wprowadzenie do programowania obiektowego...

(Ciąg dalszy ze strony 46)

(zgodziliśmy się już, że zawsze będą robić to samo niezależnie od rodzaju firgur). Korzystając z metody wirtualnej uprościmy sobie pracę.

Spróbujmy teraz opisać jak najbardziej szczegółowo sposób działania poszczególnych metod występujących w klasie C_Figures. Zacznijmy od metody Draw

```
{------ definicja metody Draw -------- }
procedure C_Figures.Draw;
begin
end;
```

Metoda ta dla klasy C_Figures jest metodą abstrakcyjną, tzn. występuje, ale nie możemy określić, jak ma być wykonywana. W klasach potommnych będzie musiała być *pokryta* metodami Draw odpowiednimi dla tych klas. Stąd powyższy zapis. Zauważmy, że po słowie **procedure** nie występuje samo słowo Draw, lecz jest połączone z nazwą klasy: C_Figures**.**Draw . W ten sposób Turbo Pascal jednoznacznie identyfikuje definicję metody Draw klasy C_Figures (przecież, jak wiemy, metody o nazwie Draw występować będą we wszystkich klasach potomnych). Jak łatwo zauważyć, pominięto parametr *color* występujący dla metody Draw w opisie klasy. Jest to dopuszczalne uproszczenie zapisu definicji.

Słowa **begin** i **end** stanowią rodzaj nawiasów dla instrukcji, sygnalizują początek (ang. *begin*) i koniec (ang. *end*) bloku instrukcji Pascala. Pomiędzy tymi słowami powinny znajdować się konkretne instrukcje opisujące sposób działania metody Draw. Co zatem robi opisana powyżej metoda? NIC. Dokładnie tak. Efektem jej działania będzie NIC, nie zostaną bowiem wykonane żadne instrukcje. Jest to działanie w pełni poprawne i pasujące do naszego obrazu klasy C_Figures - skoro nie możemy określić, jak ogólnie kreślić bliżej nieokreśloną figurę a musimy opisać sposób kreślenia, przyjmijmy, że kreślenie (Draw) po prostu nie robi NIC. COŚ będą robiły dopiero metody Draw w klasach konkretnych figur (np. trójkątów).

Mamy zatem zdefiniowaną (mimo wszystko) metodę Draw. Jak zatem powinny działać Show i Hide? Zakładamy, że rysujemy białą kreską po czarnym ekranie:

(Ciąg dalszy na stronie 48)

Wprowadzenie do programowania obiektowego...

(Ciąg dalszy ze strony 47)

```
{------ definicja metody Show --------}
procedure C_Figures.Show;
begin
  Draw(White)
end;

{------ definicja metody Hide --------}
procedure C_Figures.Hide;
begin
  Draw(Black)
end;
```

Powyższe definicje powinny być już oczywiste. Narysowanie dowolnej figury polega na wywołaniu metody Draw (kreślenie) właściwej dla tej figury. Ponieważ, jak już kilkakrotnie wspominaliśmy, metody Draw jako wirtualne powinny zostać zdefiniowane dla podklas klasy C_Figures, zarówno Hide jak i Show zadziałają poprawnie.

Pozostaje określić sposób tworzenia i usuwania obiektu należącego do klasy C_Figures. Oto odpowiednie metody:

```
{------ definicja metody tworzącej obiekt --------}
constructor C_Figures.Init;
begin
  Show
end;

{------ definicja metody usuwającej obiekt -------}
destructor C_Figures.Done;
begin
  Hide
end;
```

(Ciąg dalszy na stronie 49)

Wprowadzenie do programowania obiektowego...

(Ciąg dalszy ze strony 48)

Jest to najprostsza postać tych metod. Init po prostu wyświetla obiekt na ekranie, zaś Done usuwa go z ekranu. Komplikacje pojawią się przy definiowaniu klas potomnych, ale o tym w następnej części.

Pozostało nam jeszcze zdefiniowanie metody Move. Wykorzystamy w niej moduł biblioteczny MOUSE zawierający następujące potrzebne nam procedury obsługi myszki:

Show	pokaż kursor myszki
Hide	ukryj kursor myszki
Rightbutton	TRUE gdy naciśnięto prawy guzik myszki
Leftbutton	TRUE gdy naciśnięto prawy guzik myszki

Metoda Move może przyjąć poniższą postać:

```
{------- definicja metody poruszającej obiektem ------}
procedure C_Figures.Move;
begin {---------------------- C_Figures.Move }
 Mouse.Show;                 { pokaż kursor myszki       }
 while not Rightbutton do    { dopóki nie naciśnięto prawego guzika }
 begin
 if  Leftbutton then         { jeœli naciœniêto lewy guzik, to wtedy }
  begin
   Hide;                 { usuń ostatni obraz obiektu    }
   p0.x:=Xpos;           { wsp. x punktu p0 odpowiada wsp. x myszy }
   p0.y:=Ypos;           { wsp. y punktu p0 odpowiada wsp. y myszy }
   Show;                 { wyświetl obiekt na nowej pozycji }
  end
 end;
 Mouse.Hide                  { ukryj kursor myszki   }
end;   { ---------------------- C_Figures.Move }
```

Jak widać, po naciśnięciu lewego guzika myszki następuje wygaszenie obiektu, zmiana jego współrzędnych i wyświetlenie go na nowej pozycji. Powtarza się do momentu naciśnięcia

(Ciąg dalszy na stronie 50)

Wprowadzenie do programowania obiektowego...

(Ciąg dalszy ze strony 49)

prawego guzika myszy. Oczywiście możnaby zamiast myszy używać klawiatury i procedury ReadKey z modułu CRT. Odpowiednia metoda mogłaby wyglądać następująco:

```
{------- definicja metody poruszającej obiektem ------}
{------------------    wersja bez myszki -----------------}
procedure C_Figures.Move;
var
  c: Char;
begin { ---------------------- C_Figures.Move }
 repeat
   c := UpCase(ReadKey);    { rozpoznajemy wielkie litery }
   if c='L' then            { jeśli naciśnięto L, to wtedy }
   begin
     Hide;                  { usuń ostatni obraz obiektu        }
     p0.x:=p0.x+10;         { wsp. x punktu p0 jest zwiększana o 10    }
     p0.y:=p0.y+10;         { wsp. y punktu p0 jest zwiększana o 10    }
     Show;                  { wyświetl obiekt na nowej pozycji }
   end
 until (c ='K')         { naciśnięcie K kończy poruszanie }
end;  { ---------------------- C_Figures.Move }
```

Oczywiście jest to tylko przykład. W dalszej analizie posługiwać się będziemy przykładem „myszkowatym”, gdyż pozwala na swobodniejsze manipulowanie obiektem - ot chociażby dzięki możliwości prostego ustawiania obiektu w punkcie wskazanym myszką.

```
Program Figures;
{
-------------------------------------------------------------
    Ilustracja do tekstu o programowaniu obiektowym
                        FIG-1.PAS
                   Witold Rudolf 2006
    Definicja klasy C_Figures
-------------------------------------------------------------
}
uses Mouse, Graph;
type
  C_Figures = object                  {jest to klasa obiektów    }
                p0 : PointType;       {każda figura musi zawierać}
                                      {co najmniej jeden punkt   }
                procedure Draw(color: Integer);    virtual;
                                      {kreśenie figury           }
                procedure Show;       {metoda narysowania        }
                procedure Hide;       {metoda zmazania           }
                procedure Move;       {metoda poruszania         }
                constructor Init;      {metoda tworząca obiekt   }
                                      {co najmniej jeden punkt   }
                procedure Draw(color: Integer);    virtual;
                                      {kreślenie figury          }
                procedure Show;       {metoda narysowania        }
                procedure Hide;       {metoda zmazania           }
                procedure Move;       {metoda poruszania         }
                constructor Init;      {metoda tworząca obiekt   }
                destructor  Done;      {metoda niszcząca obiekt  }
```

(Ciąg dalszy na stronie 51)

Wprowadzenie do programowania obiektowego...

(Ciąg dalszy ze strony 50)

Tu czas na pewną uwagę, która będzie istotna przy dalszych rozważaniach. Metoda Move modyfikuje współrzędne punktu p0. Jeżeli dla wszelkich figur potomnych dla C_Figures przyjmiemy, że p0 jest punktem szczególnym, to pozwoli nam to na używanie tej samej metody dla obiektów potomnych.

Pełny tekst deklaracji klasy znajduje się w pliku FIG-1.PAS

```
              end;                {koniec definicji klasy obiektów}
{------- DEFINICJA METOD DLA KLASY C_Figures --------------------}
procedure C_Figures.Draw;
begin   {---------------------------------------- C_Figures.Draw }
Figures.Draw }
end;    {---------------------------------------- C_Figures.Draw }
procedure C_Figures.Show;
begin   {---------------------------------------- C_Figures.Show }
  Draw(White)
end;    {---------------------------------------- C_Figures.Show }
procedure C_Figures.Hide;
begin   {---------------------------------------- C_Figures.Hide }
  Draw(Black)
end;    {---------------------------------------- C_Figures.Hide }
procedure C_Figures.Move;
begin   {---------------------------------------- C_Figures.Move }
  MOUSE.Show;
  while not RightButton do  {dopóki nie naciśnieto prawego guzika}
  begin
    if LeftButton then     {jeśli naciśnieto lewy guzik, to wtedy}
    begin
      Hide;                {usuń ostatni obraz obiektu           }
      p0.X := XPos;       {wsp. X punktu p0 odpowiada wsp. X myszy}
      p0.Y := YPos;       {wsp. Y punktu p0 odpowiada wsp. Y myszy}
      Show                {wyświetl obiekt na nowej pozycji      }
    end
  end;
  MOUSE.Hide
end;    {---------------------------------------- C_Figures.Move }
constructor C_Figures.Init;
begin   {---------------------------------------- C_Figures.Init }
  Show
end;    {---------------------------------------- C_Figures.Init }
destructor C_Figures.Done;
begin   {---------------------------------------- C_Figures.Done }
  Hide
end;    {---------------------------------------- C_Figures.Done }
{------------ KONIEC DEFINICJI METOD KLASY C_Figures ------------}
```

Klasa obiektów PUNKT

Dotychczas udało nam się ustalić pewne ogólne cechy i zasady obsługi obiektów - figur geometrycznych, które mają być poruszane na ekranie komputera. Pozwoliło to nam na zdefiniowanie abstrakcyjnej klasy C_Figures, która zawierała w sobie zestaw metod i atrybutów uniwersalnych dla każdej figury płaskiej.

Zdefiniujemy teraz najprostszą klasę rzeczywistych figur geometrycznych. Są to oczywiście punkty. Definicja klasy C_Point wyglądać będzie następująco:

```
C_Point = object (C_Figures)     { jest to klasa potomna klasy
  C_Figures }
                constructor Init(x, y: Integer);
                        { metoda tworząca obiekt w punkcie  }
                        { o współrzędnych (x, y)    }
                procedure Draw(color: Integer);    virtual;
                        { kreślenie punktu }
        end;
```

(Ciąg dalszy na stronie 52)

Wprowadzenie do programowania obiektowego...

(Ciąg dalszy ze strony 51)

Dokładnie tyle. Punkt jest opisany przez atrybut p0 odziedziczony po swoim przodku (C_Figures) opisujący współrzędnie obiektu. Będzie to punkt charakterystyczny określający położenie obiektu. Podobnie dziedziczy metody Show, Hide, Move i Done. Zmiany wymaga jedynie konstruktor Init, który powinien ustalić współrzędne punktu oraz metoda Draw kreśląca punkt. Może to wyglądać np. tak:

```
{----- definicja metody tworzącej obiekt --------}
constructor C_Point.Init;
begin
  p0.X := x;            { ustalenie współrzędnej X }
  p0.Y := y;            { ustalenie współrzędnej Y }
  inherited Init        { wywo³anie Init przodka  }
end;

{------ definicja metody Draw --------}
procedure C_Point.Draw;
begin
  PutPixel(p0.X, p0.Y, color);
  { zapalenie na ekranie punktu (X,Y) w kolorze COLOR }
end;
```

Procedura PutPixel jest standardową procedurą graficzną umieszczoną w module GRAPH. Pewnego wyjaśnienia wymaga użycie odwołania **inherited** w konstruktorze. Pozwala to na wywołanie metody o podanej nazwie używanej przez przodka bez względu na to, jak nazywa się klasa - przodek.

Zauważmy, że klasa C_Point posiada dokładnie te same atrybuty (p0) i metody (Init, Done, Draw, Show, Hide i Move) co jej nad-klasa (czyli przodek) C_Figures. Przy tym atrybut (atrybuty nazywamy czasami *polami*) p0 oraz metody Done, Show, Hide i Move są odziedziczone po przodku. Konstruktor Init jest własną metodą C_Point zaś przedefiniowanie wirtualnej metody Draw gwarantuje poprawne działanie odziedziczonych metod Show i Hide.

(Ciąg dalszy na stronie 53)

Wprowadzenie do programowania obiektowego...

(Ciąg dalszy ze strony 52)

Właśnie tutaj wirtualność gwarantuje, że metoda Show wywołana przez obiekt z klasy C_Point wywoła metodę Draw z klasy C_Point. Gdybyśmy pominęli słowo *virtual*, metoda Show skorzystałaby z najbliższej jej metody Draw, czyli z klasy C_Figures (bo w niej jest zdefiniowana metoda Show). Zatem pominięcie w opisie metody Draw słowa *virtual* spowodowałoby konieczność ponownego definiowania identycznych jak w C_Figures metod Show, Hide i Move w klasie C_Point.

Utworzyliśmy zatem wstępną hierachię klas obiektów. Na razie (będziemy stopniowo ją rozbudowywać) wygląda to tak:

```
C_Figures
   |
   |
C_Point
```

Najwyższy czas zobaczyć efekty naszej pracy. Dopóki mieliśmy tylko abstrakcyjną klasę C_Figures, nie bardzo można było obejżeć obiekt tej klasy. Obecnie dysponujemy już klasą obiektów, które można narysować (punkty). Spróbujmy zatem uruchomić program obsługujący obiekt z klasy C_Point. Program (w Turbo Pascalu) wyglądać będzie następująco:

```
Program Figury1;
uses Graph,          { podłączenie biblioteki graficznej }
     Mouse;          { podłączenie biblioteki obsługi myszki }

{ definicja klasy C_Figures  }
{ definicja klasy C_Point    }
{ deklaracja metod klasy C_Figures  }
{ deklaracja metod klasy C_Point (czyli Init i Draw)  }

var                { deklaracja zmiennych, których będziemy używać }
   dr, gr: Integer;          { zmienne do inicjowania grafiki }
   obiekt: C_Point;          { konkretny obiekt klasy C_Point }
begin  { ------ część główna programu ----- }
```

(Ciąg dalszy na stronie 54)

Wprowadzenie do programowania obiektowego...

(Ciąg dalszy ze strony 53)

```
{ włączenie trybu graficznego }
   dr := 0;  gr := 0;
   InitGraph(gr, dr, '');

{ obsługa obiektu  }
   obiekt.Init(300, 200);   { zainicjowanie obiektu - punkt (300, 200-
  200)  }
   obiekt.Move;             { poruszanie obiektem }
   obiekt.Done;             { usunięcie obiektu }

{ wyłączenie trybu graficznego  }
   CloseGraph
end.      { koniec programu }
```

UWAGA: w bieżącej kartotece powinien znajdować się plik EGAVGA.BGI

Uruchamiamy program - i obserwujemy efekty. Na początku zostaje zapalony punkt na ekranie o współrzędnych (300, 200). Możemy nim poruszać (pamiętacie metodę Move?), poruszanie kończymy naciśnięciem klawisza [Esc].

Pełny tekst programu znajduje się w pliku FIG-2.PAS

```
Program Figures;
{
-----------------------------------------------------------------
    Ilustracja do tekstu o programowaniu obiektowym
                        FIG-2.PAS
                   Witold Rudolf 2006
    Definicja klasy C_Figures
                   C_Point
-----------------------------------------------------------------
}
uses Mouse, Graph;
type
  C_Figures = object                  {jest to klasa obiekt˘w    }
                p0 : PointType;       {ka§da figura musi zawieraŤ}
                                      {co najmniej jeden punkt   }
                procedure Draw(color: Integer);    virtual;
                                      {krežlenie figury          }
                procedure Show;       {metoda narysowania        }
                procedure Hide;       {metoda zmazania           }
                procedure Move;       {metoda poruszania         }
                constructor Init;      {metoda tworzłca obiekt   }
                destructor  Done;      {metoda niszczłca obiekt  }
              end;                 {koniec definicji klasy obiekt˘w}
  C_Point = object (C_Figures)    {Klasa potomna klasy C_Figures  }
                procedure Draw(color: Integer);    virtual;
                                      {krežlenie punktu          }
                constructor Init(x, y: Integer);
            end;                   {koniec definicji klasy obiekt˘w}
{------- DEFINICJA METOD DLA KLASY C_Figures --------------------}
procedure C_Figures.Draw;
begin   {---------------------------------------- C_Figures.Draw }
end;    {---------------------------------------- C_Figures.Draw }
procedure C_Figures.Show;
begin   {---------------------------------------- C_Figures.Show }
  Draw(White)
end;    {---------------------------------------- C_Figures.Show }
procedure C_Figures.Hide;
begin   {---------------------------------------- C_Figures.Hide }
  Draw(Black)
end;    {---------------------------------------- C_Figures.Hide }
procedure C_Figures.Move;
begin   {---------------------------------------- C_Figures.Move }
  MOUSE.Show;
  while not RightButton do  {dopoki nie nacisnieto prawego guzika}
  begin
    if LeftButton then     {jesli nacisnieto lewy guzik, to wtedy}
    begin
```

Wprowadzenie do programowania obiektowego...

(Ciąg dalszy ze strony 54)

```
      Hide;                  {usun ostatni obraz obiektu          }
      p0.X := XPos;        {wsp. X punktu p0 odpowiada wsp. X myszy}
      p0.Y := YPos;        {wsp. Y punktu p0 odpowiada wsp. Y myszy}
      Show                 {wyswietl obiekt na nowej pozycji      }
    end
  end;
  MOUSE.Hide
end;    {--------------------------------------- C_Figures.Move }
constructor C_Figures.Init;
begin   {--------------------------------------- C_Figures.Init }
  Show
end;    {--------------------------------------- C_Figures.Init }
destructor C_Figures.Done;
begin   {--------------------------------------- C_Figures.Done }
  Hide
end;    {--------------------------------------- C_Figures.Done }
{------------ KONIEC DEFINICJI METOD KLASY C_Figures ------------}
{--------- DEFINICJA METOD DLA KLASY C_Point --------------------}
procedure C_Point.Draw;
{Zapala na ekranie punkt p0 w kolorze COLOR}
begin   {---------------------------------------- C_Point.Draw }
  PutPixel(p0.X, p0.Y, color);
end;    {---------------------------------------- C_Point.Draw }
constructor C_Point.Init;
begin   {---------------------------------------- C_Point.Init }
  p0.X := x;
  p0.Y := y;
  Show
end;    {---------------------------------------- C_Point.Init }
{------------- KONIEC DEFINICJI METOD KLASY C_Point -------------}
var
  F: C_Point;                       {zmienna - konkretny obiekt klasy }
  dr, gr: Integer;                  {zmienne inicjujace tryb graficzny}
begin
  dr := Detect; gr := 0;
  InitGraph(dr, gr, '');            {zainicjowanie trybu graficznego}
  F.Init(100, 100);                 {utworzenie obiektu}
  F.Move;                           {poruszanie obiektem}
  F.Done;                           {likwidacja obiektu}
  CloseGraph                        {wylaczenie trybu graficznego}
end.
```

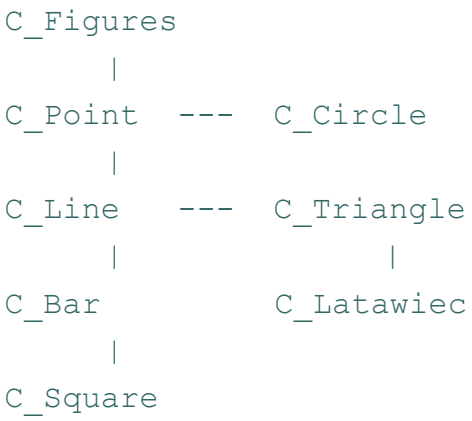
Hierarchia klas

Po tych wstępnych rozważaniach zbudujemy większy zestaw obiektów. Zdefiniujemy nowe klasy:

C_Circle
C_Line
C_Triangle
C_Bar
C_Square

oraz C_Latawiec

tworząc następującą hierarchię klas (czyli strukturę *przodek-potomek*)



Przyjęta hierarchia pozwala na budowanie obiektów potomnych jako przodka obdarzonego pewnymi dodatkowymi cechami (np. kwadrat to specyficzny rodzaj prostokąta, latawiec będzie zbudowany z dwóch trójkątów równoramiennych).

(Ciąg dalszy na stronie 56)

Wprowadzenie do programowania obiektowego...

(Ciąg dalszy ze strony 55)

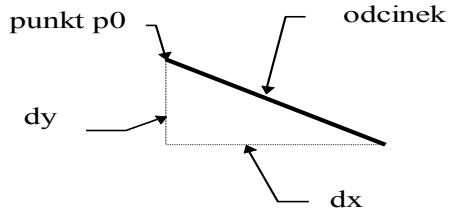
Jak pamiętamy, dla każdej klasy obiektów musimy określić punkt charakterystyczny p0, którego przesunięcie zagwarantuje nam poprawne przesunięcie całej figury. Przyjmijmy, że będzie to dla odpowiednich figur:

okrąg	- jego środek
odcinek	- jego lewy początek
prostokąt	- jego lewy górny róg
kwadrat	- jego lewy górny róg
trójkąt	- jego górny wierzchołek
latawiec	- jego górny wierzchołek

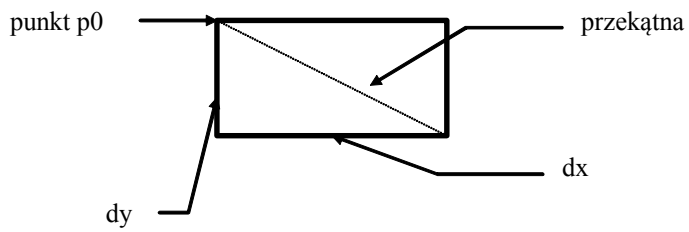
Ponadto dodać trzeba będzie nowe atrybuty - pola przechowujące dane pozwalające na kreślenie obiektu. Będą to (dla odpowiednich obiektów):

okrąg - *r* : promień okręgu

odcinek - *dx* (długość odcinka po wsp. X) i *dy* (długość po wsp. Y)



prostokąt - jak dla odcinka z tym, że będą to długości przekątnej

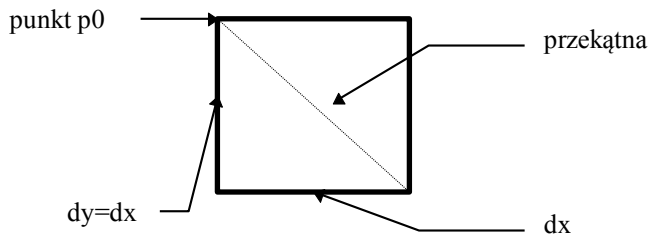


kwadrat - jak dla prostokąta, ale konstruktor wymagać będzie podawania *bx* - długości boku, bo dla kwadratu *dx = dy = bx*

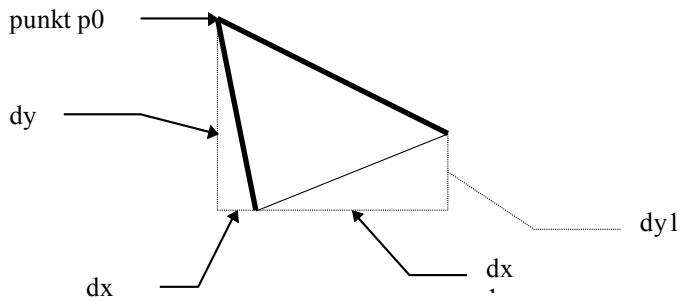
(Ciąg dalszy na stronie 57)

Wprowadzenie do programowania obiektowego...

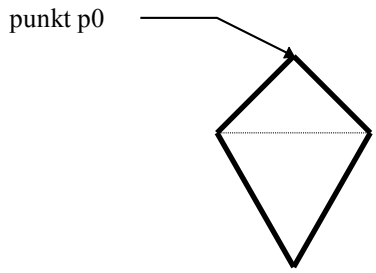
(Ciąg dalszy ze strony 56)



trójkąt - jak dla odcinka plus *dx1* i *dx2* określające drugi bok - trzeci już będzie można obliczyć



latawiec - jak dla trójkąta, figura ta jest budowana na równoramiennych trójkątach, z których drugi jest 3-krotnie wyższy.



W każdej klasie konieczne będzie zredefiniowane konstruktora Init oraz wirtualnej metody Draw.

W części głównej programu konieczne będzie ustalanie typu zmiennej (klasy obiektu) i parametrów konstruktora Init zależnie od klasy. Poniżej trzy przykłady:

RYSOWANIE OKRĘGU

```
var
   F      : C_Circle;
   dr,gr  : Integer; { zmienne inicjujace tryb graficzny }
```

(Ciąg dalszy na stronie 58)

Wprowadzenie do programowania obiektowego...

(Ciąg dalszy ze strony 57)

```
begin
   dr:=detect;gr:=0;
   InitGraph(dr,gr,'');     { zainicjowanie trybu graficznego }

   F.Init(100,100,50);
```

... (ciąg dalszy programu)

RYSOWANIE PROSTOKĄTA

```
var
   F     : C_Bar;
   dr,gr : Integer; { zmienne inicjujące tryb graficzny }
begin
  dr:=detect;gr:=0;
  InitGraph(dr,gr,'');      { zainicjowanie trybu graficznego }

  F.Init(100, 100, 90, 50);
```

... (ciąg dalszy programu)

RYSOWANIE LATAWCA

```
var
   F     : C_Latawiec;
   dr,gr : Integer; { zmienne inicjujace tryb graficzny }
begin
   dr:=detect;gr:=0;
   InitGraph(dr,gr,'');     { zainicjowanie trybu graficznego }

   F.Init(100, 100, 30, 50);
```

... (ciąg dalszy programu)

Pełny tekst programu znajduje się w pliku FIG-3.PAS

Wprowadzenie do programowania obiektowego...

```
Program Figures;
{-----------------------------------------
Ilustracja do tekstu o programowaniu obiektowym
           Fig-3.pas
       Witold Rudolf 2006
       Definicja klasy C_Figures
                       C_Point
                       C_Circle
                       C_Line
                       C_Triangle
                       C_Bar
                       C_Square
             oraz      C_Latawiec

     Hierarchia klas:  C_Figures
                          |
                       C_Point  ---  C_Circle
                          |
                       C_Line   ---  C_Triangle
                          |             |
                       C_Bar         C_Latawiec
                          |
                       C_Square

-----------------------------------------------}
Uses Mouse,graph;
Type
 C_Figures=object                                {jest to klasa obiektów}
             p0:PointType;                       {każda figura musi zawierać}
                                                 {co najmniej jeden punkt}
             Procedure Draw(color:Integer);virtual; {kreślenie figury}
             Procedure Show;                     {metoda narysowania}
             Procedure Hide;                     {metoda zmazania}
             Procedure Move;                     {metoda poruszania}
             Constructor Init;                   {metoda tworząca obiekt}
             Destructor Done;                    {metoda niszcząca obiekt}
           end;                                  {koniec definicji klasy obiektow}
 C_Point=object(C_Figures)                       {klasa potomna klasy C_Figures}
           Procedure Draw(color:Integer);virtual;  {kreślenie punktu}
           Constructor Init(x,y:Integer);
         end;                                    {koniec definicji klasy obiektów}
 C_Circle=object(C_Point)
            r:Integer;
            Procedure Draw(color:Integer);virtual;
            Constructor Init(x,y,promien:Integer);
          end;
 C_Line=object(C_Point)
             dx, dy:Integer;
             Procedure Draw(color:Integer);virtual;
             Constructor Init(x,y,px,py:Integer);
            end;
 C_Triangle=object(C_Line)
             dx1, dy1: Integer;
             Procedure Draw(color:Integer);virtual;
             Constructor Init(x,y,ox1,oy1,ox2,oy2:Integer);
            end;
 C_Bar=object(C_Line)
             Procedure Draw(color:Integer);virtual;
             Constructor Init(x,y,bx,by:Integer);
            end;
 C_Square=object(C_Bar)
             Procedure Draw(color:Integer);virtual;
             Constructor Init(x,y,bx:Integer);
            end;
 C_Latawiec=object(C_Triangle)
             Procedure Draw(color:Integer);virtual;
             Constructor Init(x, y, ox, oy:Integer);
            end;
{-----------DEFINICJA METOD DLA KLASY C_FIGURES------------}
Procedure C_Figures.Draw;
begin {----------------------C_Figures.Draw}
end;  {----------------------C_Figures.Draw}
Procedure C_Figures.Show;
begin {----------------------C_Figures.Show}
 Draw(white);
end;  {----------------------C_Figures.Show}
```

(Ciąg dalszy na stronie 60)

Wprowadzenie do programowania obiektowego...

(Ciąg dalszy ze strony 59)

```
Procedure C_Figures.Hide;
begin {-----------------------C_Figures.Hide}
 Draw(black);
end;  {-----------------------C_Figures.Hide}
Procedure C_Figures.Move;
begin {-----------------------C_Figures.Move}
 Mouse.Show;
 While not Rightbutton do     {dopóki nie naciśnieto prawego guzika}
 begin
  if Leftbutton then          {jeśli naciśnieto lewy guzik, to wtedy}
  begin
   Hide;                      {usuń ostatni obraz obiektu}
   p0.x:=Xpos;                {wsp. x punktu p0 odpowiada wsp. x myszy}
   p0.y:=Ypos;                {wsp. y punktu p0 odpowiada wsp. y myszy}
   Show;                      {wyświetl obiekt na nowej pozycji}
  end
 end;
 Mouse.Hide
end;  {-----------------------C_Figures.Move}
Constructor C_Figures.Init;
begin {-----------------------C_Figures.Init}
 Show;
end;  {-----------------------C_Figures.Init}
Destructor C_Figures.Done;
begin {-----------------------C_Figures.Done}
 Hide;
end;  {-----------------------C_Figures.Done}
{-----------KONIEC DEFINICJI METOD DLA KLASY C_FIGURES------------}
{---------------DEFINICJA METOD DLA KLASY C_POINT----------------}

Procedure C_Point.Draw;
{zapala na ekranie punkt p0 w kolorze COLOR}
begin {-----------------------C_Point.Draw}
 PutPixel(p0.x,p0.y,color);
end;  {-----------------------C_Point.Draw}
Constructor C_Point.Init;
begin {-----------------------C_Point.Init}
 p0.x:=x;
 p0.y:=y;
 Show
end;  {-----------------------C_Point.Init}
{-------------KONIEC DEFINICJI METOD DLA KLASY C_POINT--------------}
{---------------DEFINICJA METOD DLA KLASY C_Circle----------------}

Constructor C_Circle.Init;
begin {-----------------------C_Circle.Init}
 p0.x:=x;
 p0.y:=y;
 r:=promien;
 Show
end;  {-----------------------C_Circle.Init}
Procedure C_Circle.Draw;
{zapala na ekranie okrąg w kolorze COLOR}
begin {-----------------------C_Circle.Draw}
 SetColor(Color);
 circle(p0.x,p0.y,r);
end;  {-----------------------C_Circle.Draw}
{-------------KONIEC DEFINICJi METOD DLA KLASY C_Circle--------------}
{---------------DEFINICJA METOD DLA KLASY C_Line----------------}

Constructor C_Line.Init;
begin {-----------------------C_Line.Init}
 p0.x:=x;
 p0.y:=y;
 dx:=px;
 dy:=py;
 Show
end;  {-----------------------C_Line.Init}
Procedure C_Line.Draw;
{zapala na ekranie odcinek w kolorze COLOR}
begin {-----------------------C_Line.Draw}
 SetColor(color);
 line(p0.x,p0.y,p0.x+dx,p0.y+dy);
end;  {-----------------------C_Line.Draw}
{-------------KONIEC DEFINICJi METOD DLA KLASY C_Line--------------}
{---------------DEFINICJA METOD DLA KLASY C_Triangle----------------}
```

(Ciąg dalszy na stronie 61)

Wprowadzenie do programowania obiektowego...

(Ciąg dalszy ze strony 60)

```
Constructor C_Triangle.Init;
begin {-----------------------C_Triangle.Init}
 dx1:=ox2;
 dy1:=oy2;
 Inherited Init(x, y, ox1, oy1);
end;  {-----------------------C_Triangle.Init}
Procedure C_Triangle.Draw;
{zapala na ekranie trójkąt w kolorze COLOR}
begin {-----------------------C_Triangle.Draw}
 SetColor(color);
 Line(p0.x, p0.y, p0.x+dx, p0.y+dy);
 Line(p0.x, p0.y, p0.x+dx1, p0.y+dy1);
 Line(p0.x+dx, p0.y+dy, p0.x+dx1, p0.y+dy1);
end;  {-----------------------C_Triangle.Draw}
{-------------KONIEC DEFINICJi METOD DLA KLASY C_Triangle--------------}
{---------------DEFINICJA METOD DLA KLASY C_Bar----------------}

Constructor C_Bar.Init;
begin {-----------------------C_Bar.Init}
 Inherited Init(x, y, bx, by)
end;  {-----------------------C_Bar.Init}
Procedure C_Bar.Draw;
{zapala na ekranie prostokat w kolorze COLOR}
begin {-----------------------C_Bar.Draw}
 SetFillStyle(SolidFill, color);
 Bar(p0.x,p0.y,p0.x-dx,p0.y-dy);
end;  {-----------------------C_Bar.Draw}
{-------------KONIEC DEFINICJi METOD DLA KLASY C_Bar--------------}
{---------------DEFINICJA METOD DLA KLASY C_Square----------------}

Procedure C_Square.Draw;
{zapala na ekranie kwadrat w kolorze COLOR}
begin {-----------------------C_Square.Draw}
 SetFillStyle(SolidFill, color);
 Bar(p0.x, p0.y, p0.x-dx, p0.y-dy);
end;  {-----------------------C_Square.Draw}
Constructor C_Square.Init;
begin {-----------------------C_Square.Init}
 Inherited Init(x, y, bx, bx)
end;  {-----------------------C_Square.Init}
{-------------KONIEC DEFINICJI METOD DLA KLASY C_Square--------------}
{---------------DEFINICJA METOD DLA KLASY C_Latawiec----------------}

Procedure C_Latawiec.Draw;
{zapala na ekranie latawiec w kolorze COLOR}
begin {-----------------------C_LatawiecTriangle.Draw}
 SetColor(color);
 Line(p0.x,p0.y,p0.x-dx,p0.y+dy);
 Line(p0.x,p0.y,p0.x+dx,p0.y+dy);
 Line(p0.x-dx,p0.y+dy,p0.x,p0.y+(3*dy));
 Line(p0.x+dx,p0.y+dy,p0.x,p0.y+(3*dy));
end;  {-----------------------C_Latawiec.Draw}
Constructor C_Latawiec.Init;
begin {-----------------------C_Latawiec.Init}
 C_Line.Init(x, y, ox, oy)
end;  {-----------------------C_Latawiec.Init}
{-------------KONIEC DEFINICJi METOD DLA KLASY C_Latawiec--------------}

var
 F     : C_Circle;   {zmienna-konkretny obiekt klasy potomnej do C_Figures}
                     {tutaj można modyfikować typ obiektu}
 dr,gr : Integer; {zmienne inicjujące tryb graficzny}

BEGIN
 dr:=detect;gr:=0;
 InitGraph(dr,gr,''); {zainicjowanie trybu graficznego}
 F.Init(100,100,50);
                                  {utworzenie obiektu }
                                  {trzeba zmienać w zależności od typu}
 F.Move;                         {poruszanie obiektem}
 F.Done;                         {likwidacja obiektu}
 CloseGraph;             {wyłączenie trybu graficznego}
END.
```

UWAGA. Ciąg dalszy artykułu: **Wprowadzenie do programowania obiektowego na przykładzie języka Turbo Pascal** zostanie umieszczone w kolejnym numerze kwartalnika.

mgr inż. Wojciech Błaszczyk

Konkursy informatyczne i techniczne w roku szkolnym 2008/2009

WKT – Wojewódzki Konkurs Techniczny R. Ratuś, W. Bartoszek (wbartoszek@wodip.opole.pl) i Z. Kucik

Konkurs dla uczniów zainteresowanych współczesną techniką, polega na wykazaniu się umiejętnościami z zakresu stosowania wiedzy technicznej.

eliminacje szkolne	– do 30.11.2008
eliminacje gminne	– 02.02.2009 godz. 8^{00}
finał dla szkół podstawowych	– 02.03.2009 godz. 9^{30} WODIiP
finał dla gimnazjów	– 16.03.2009 godz. 9^{30} WODIiP

Wystawa: R. Ratuś, W. Bartoszek (wbartoszek@wodip.opole.pl) i Z. Kucik

Celem wystawy jest prezentacja twórczości technicznej uczniów szkół podstawowych i gimnazjów. W konkursie prezentowane są prace techniczne wykonane dowolną techniką.

dostarczenie prac	– 03.04.2009
wystawa	– 06–13.04.2009
ogłoszenie wyników	– 13.04.2009

WKI – Wojewódzki Konkurs Informatyczny – W. Rudolf (wrudolf@wodip.opole.pl)

Konkurs dla uczniów szkół podstawowych i gimnazjów zainteresowanych algorytmiką i na rozwiązywaniu zadań w językach wyższego poziomu. Wymaga od uczestników umiejętności programowania.

zgłoszenia szkół	– 17.10.2008
eliminacje szkolne	– 30.10.2008
eliminacje wojewódzkie	– 05.01.2009
finał	– 16.02.2009

OTI – Opolski Turniej Informatyczny – W. Rudolf (wrudolf@wodip.opole.pl)

Konkurs dla uczniów szkół ponadgimnazjalnych zainteresowanych algorytmiką i na rozwiązywaniu zadań w językach wyższego poziomu. Wymaga od uczestników umiejętności programowania.

zgłoszenia szkół	– 30.10.2008
zawody szkolne	– 30.11.2008
zawody rejonowe	– 05.01.2009
finał	– 16.02.2009

Opolszczyzna Wczoraj, Dziś i Jutro – „Z plecakiem po Opolszczyźnie" – K. Pędziwiatr (kpedziwiatr@wodip.opole.pl)

Konkurs dla uczniów wszystkich etapów kształcenia polegający na przygotowaniu prac narzędziami TI o tematyce propagującej różne formy turystyki na Opolszczyźnie.

eliminacje szkolne	– 13.02.2009
eliminacji rejonowe	– 16.02.2009
finał	– 09.03.2009

(Ciąg dalszy na stronie 63)

Konkursy informatyczne i techniczne w roku szkolnym 2008/2009

(Ciąg dalszy ze strony 62)

Opolski Konkurs Technologii Informacyjnej – A. Koj (akoj@wodip.opole.pl)

Konkurs dla uczniów szkół podstawowych i gimnazjalnych z wiedzy i umiejętności stosowania technologii informacyjnej. Uczniowie rozwiązują umieszczony test na platformie internetowej, w części finałowej sprawdzane są umiejętności praktyczne przy komputerze.

zgłoszenia uczestników – 10.04.2009
etap pierwszy – 27.04.2009
etap – 4.05.2009

Wojewódzki Konkurs Informatyczny „Na Program Uczniowski" J. Szymczyna (jszymczyna@wodip.opole.pl) i Z. Kucik (zkucik@wodip.opole.pl)

Konkurs polega na samodzielnym tworzeniu przez uczniów programów komputerowych na dowolny temat związany z dydaktyką. Programy muszą być tworzone w językach programowania wysokiego poziomu.

eliminacje szkolne – 31.03.2009
zgłoszenie prac – 06.04.2009
eliminacje rejonowe – 20.04.2009
finały – 11.05.2009

Konkurs na Pomoc Dydaktyczną „Uczniowie-Uczniom" – T. Marenin (tmarenin@wodip.opole.pl)

Konkurs polega na przygotowaniu materiałów dydaktycznych w formie e-lekcji, w standardzie HTML (strony internetowe) lub SCORM (specjalne edytory treści).

eliminacje szkolne – 30.03.2009
zgłaszanie prac do etapu rejonowego – 10.04.2009
kwalifikacja prac do finału – 18.5.2009
finały – 8.06.2009

Szczegółowe informacje na stronie http://konkursy.wodip.opole.pl