



Wydanie III

Greg Perry, Dean Miller

# Programowanie dla początkujących w 24 godziny

SAMS

Helion 

Tytuł oryginału: Sams Teach Yourself Beginning Programming in 24 Hours, Third Edition

Tłumaczenie: Tomasz Walczak

Projekt okładki: Studio Gravite / Olsztyn  
Obarek, Pokoński, Pazdrijowski, Zaprucki

ISBN: 978-83-283-2939-3

Authorized translation from the English language edition: SAMS TEACH YOURSELF BEGINNING PROGRAMMING IN 24 HOURS, Third Edition, ISBN 0672337002; by Greg Perry; and Dean Miller; published by Pearson Education, Inc, publishing as SAMS Publishing. Copyright © 2014 by Pearson Education

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education Inc.

Polish language edition published by HELION S.A. Copyright © 2017.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji. Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock Images LLC.

Wydawnictwo HELION  
ul. Kościuszki 1c, 44-100 GLIWICE  
tel. 32 231 22 19, 32 230 98 63  
e-mail: [helion@helion.pl](mailto:helion@helion.pl)  
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:  
<ftp://ftp.helion.pl/przyklady/prpo24.zip>

Drogi Czytelniku!  
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres  
<http://helion.pl/user/opinie/prpo24>  
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

# Spis treści

---

O autorach .....	11
Podziękowania .....	12
Wprowadzenie .....	13

## **CZĘŚĆ I ZACZNIJ PROGRAMOWAĆ JUŻ DZIŚ**

<b>Godzina 1.</b>	<b>Praktyczne ćwiczenia z programowania .....</b>	<b>19</b>
	Przygotuj się do programowania .....	19
	Co robi program komputerowy? .....	20
	Często powtarzane mity na temat programowania .....	21
	Istnieje już wiele programów .....	23
	Programiści są poszukiwani na rynku pracy .....	23
	Prawdziwa wartość programów .....	23
	Użytkownicy zwykle nie są właścicielami programów .....	24
	Udostępnianie programów komputerowych .....	24
	Twój pierwszy program .....	26
	Komentarze objaśniające kod .....	27
	Wpisywanie własnego programu .....	29
	Podsumowanie .....	31
	Pytania i odpowiedzi .....	32
	Warsztaty .....	32
<b>Godzina 2.</b>	<b>Proces i techniki .....</b>	<b>35</b>
	Do czego potrzebne są programy? .....	35
	Programy, programy, wszędzie programy .....	38
	Programy jako wskazówki .....	38
	Podsumowanie .....	49
	Pytania i odpowiedzi .....	49
	Warsztaty .....	49

<b>Godzina 3.</b>	<b>Projektowanie programu</b> .....	<b>51</b>
	Dlaczego potrzebny jest projekt? .....	51
	Umowa między użytkownikiem a programistą .....	52
	Etapy projektowania .....	53
	Podsumowanie .....	65
	Pytania i odpowiedzi .....	65
	Warsztaty .....	66
<b>Godzina 4.</b>	<b>Pobieranie danych wejściowych i wyświetlanie danych wyjściowych</b> .....	<b>69</b>
	Wyświetlanie danych na ekranie za pomocą JavaScriptu .....	69
	Przechowywanie danych .....	71
	Przypisywanie wartości .....	73
	Pobieranie danych z klawiatury za pomocą metody prompt .....	75
	Podsumowanie .....	82
	Pytania i odpowiedzi .....	83
	Warsztaty .....	83
<b>Godzina 5.</b>	<b>Przetwarzanie danych z wykorzystaniem liczb i słów</b> .....	<b>85</b>
	Jeszcze o łańcuchach znaków .....	85
	Wykonywanie obliczeń matematycznych w JavaScriptcie .....	89
	W jaki sposób komputery wykonują obliczenia? .....	92
	Używanie tabeli kodów ASCII .....	95
	Przegląd metod .....	96
	Podsumowanie .....	100
	Pytania i odpowiedzi .....	100
	Warsztaty .....	101
<b>Godzina 6.</b>	<b>Sterowanie programami</b> .....	<b>103</b>
	Porównywanie danych za pomocą instrukcji if .....	103
	Pisanie warunków .....	106
	Pętle .....	108
	Podsumowanie .....	116
	Pytania i odpowiedzi .....	116
	Warsztaty .....	116

<b>Godzina 7.</b>	<b>Narzędzia do debugowania</b> .....	119
	Pierwszy błąd .....	119
	Wszystko zależy od precyzji .....	120
	Pisz przejrzyste programy .....	124
	Przećwicz debugowanie w JavaScriptcie .....	125
	Dziennik konsoli JavaScriptu .....	129
	Dodatkowe techniki debugowania .....	130
	Podsumowanie .....	131
	Pytania i odpowiedzi .....	131
	Warsztaty .....	132

## **CZĘŚĆ II    PODSTAWY PROGRAMOWANIA**

<b>Godzina 8.</b>	<b>Techniki programowania strukturalnego</b> .....	135
	Programowanie strukturalne .....	135
	Umieszczanie kodu w JavaScriptcie w funkcjach .....	141
	Testowanie programu .....	144
	Profilowanie kodu .....	145
	Wróćmy do programowania .....	146
	Podsumowanie .....	146
	Pytania i odpowiedzi .....	146
<b>Godzina 9.</b>	<b>Pisanie algorytmów</b> .....	149
	Liczniki i akumulatory .....	150
	Zmienne tablicowe .....	152
	Obliczanie łącznych wartości za pomocą akumulatorów .....	155
	Przestawianie wartości .....	156
	Sortowanie .....	157
	Przeszukiwanie tablic .....	161
	Więcej o funkcjach .....	166
	Pętle zagnieżdżone .....	170
	Podsumowanie .....	170
	Pytania i odpowiedzi .....	170
	Warsztaty .....	170

<b>Godzina 10. Radość z programowania .....</b>	<b>173</b>
Zmienianie zdjęć na stronie .....	173
Rejestrowanie pozycji kursora myszy .....	178
Dodawanie do witryny paska z powtarzаныmi informacjami .....	180
Podsumowanie .....	183
Pytania i odpowiedzi .....	183
Warsztaty .....	184
<b>Godzina 11. Zaawansowane techniki programowania .....</b>	<b>187</b>
Słaby punkt JavaScriptu .....	187
Zapisywanie plików cookie .....	190
Wczytywanie plików cookie .....	192
Usuwanie utworzonych plików cookie .....	193
Podsumowanie .....	197
Pytania i odpowiedzi .....	197
Warsztaty .....	198
<b>CZĘŚĆ III   PROGRAMOWANIE OBIEKTOWE Z UŻYCIEM JAVY</b>	
<b>Godzina 12. Programowanie w Javie .....</b>	<b>203</b>
Wprowadzenie do Javy .....	204
Java udostępnia zawartość wykonywalną .....	206
Automatyczne wykonywanie .....	208
Zawartość wykonywalna dostosowana do wielu systemów .....	208
Podsumowanie użytkowania Javy .....	210
Zacznij od niezależnego programu w Javie .....	211
Interfejs Javy .....	211
Kwestie bezpieczeństwa .....	212
Wypróbuj Javę .....	213
Mechanizmy języka Java .....	214
Przygotowania do rozpoczęcia .....	218
Podsumowanie .....	219
Pytania i odpowiedzi .....	219
Warsztaty .....	220

<b>Godzina 13. Szczegółowe omówienie Javy</b> .....	221
Definiowanie danych w Javie .....	221
Operatory .....	226
Sterowanie programem .....	230
Od szczegółów do ogólnego poziomu .....	234
Podsumowanie .....	235
Pytania i odpowiedzi .....	235
Warsztaty .....	235
<b>Godzina 14. Java ma klasę</b> .....	237
Używanie środowiska NetBeans do uruchamiania programów Javy ....	237
Przejsie do graficznego interfejsu użytkownika .....	241
Java i programowanie obiektowe .....	243
Omówienie klas .....	244
Czy rozumiesz programowanie obiektowe? .....	246
Za wykonywanie zadań w klasach odpowiadają metody .....	246
Podsumowanie .....	249
Pytania i odpowiedzi .....	249
Warsztaty .....	249
<b>Godzina 15. Aplety i strony internetowe</b> .....	251
O pisaniu apletów Javy .....	251
Tworzenie apletu Javy .....	252
Umieszczanie apletu na stronie internetowej .....	258
Wyświetlanie apletu na stronie internetowej .....	259
Podsumowanie .....	260
Pytania i odpowiedzi .....	261
Warsztaty .....	261

#### **CZĘŚĆ IV INNE JĘZYKI PROGRAMOWANIA**

<b>Godzina 16. HTML5 i CSS3</b> .....	265
Programowanie w HTML-u .....	265
Prostszy przykład .....	269
Szybkie wprowadzenie do HTML-a .....	271
Używanie stylów CSS do określania wyglądu tekstu .....	274
Dodawanie grafiki do witryn za pomocą HTML-a .....	276
Podsumowanie .....	277
Pytania i odpowiedzi .....	278
Warsztaty .....	278

<b>Godzina 17. JavaScript i AJAX .....</b>	<b>281</b>
Wprowadzenie do AJAX-a .....	281
Używanie obiektów typu XMLHttpRequest .....	285
Tworzenie prostej biblioteki AJAX-owej .....	287
Tworzenie quizu z wykorzystaniem AJAX-a i opisanej biblioteki .....	289
Podsumowanie .....	293
Pytania i odpowiedzi .....	293
Warsztaty .....	293
<b>Godzina 18. Skrypty w PHP .....</b>	<b>295</b>
Czego potrzebujesz do programowania w PHP? .....	295
Podstawowe struktury ze skryptów PHP .....	297
Pętle .....	301
Cegiełki języka PHP: zmienne, typy danych i operatory .....	303
Używanie i tworzenie funkcji w PHP .....	312
Praca z obiektami w języku PHP .....	316
Typowe zastosowania języka PHP .....	320
Podsumowanie .....	321
Pytania i odpowiedzi .....	321
Warsztaty .....	322
<b>Godzina 19. Programowanie w językach C i C++ .....</b>	<b>325</b>
Wprowadzenie do języka C .....	325
Czego potrzebujesz do programowania w językach C i C++? .....	326
Spojrzenie na kod w C .....	327
Dane w języku C .....	329
Funkcje w C .....	330
Operatory w C .....	336
Instrukcje sterujące w C są takie jak w JavaScriptcie .....	337
Nauka języka C++ .....	337
Terminologia obiektowa .....	338
Podstawowe różnice między językami C i C++ .....	338
Wprowadzenie do obiektów w języku C++ .....	340
Co dalej? .....	345
Podsumowanie .....	346
Pytania i odpowiedzi .....	346
Warsztaty .....	347



<b>Godzina 20. Programowanie w języku Visual Basic 2012 .....</b>	<b>349</b>
Zawartość ekranu w środowisku Visual Basica .....	349
Tworzenie od podstaw prostej aplikacji .....	351
Inne uwagi związane z programowaniem w Visual Basicu .....	358
Następny krok .....	360
Podsumowanie .....	361
Pytania i odpowiedzi .....	361
Warsztaty .....	361
<b>Godzina 21. C# i platforma .NET .....</b>	<b>363</b>
Przeznaczenie platformy .NET .....	363
Środowisko CLR .....	364
Biblioteka FCL .....	365
Platforma przetwarzania równoległego .....	366
Środowisko DLR .....	366
Język C# .....	366
Podsumowanie .....	374
Pytania i odpowiedzi .....	374
Warsztaty .....	374
<b>CZĘŚĆ V     BRANŻA PROGRAMISTYCZNA</b>	
<b>Godzina 22. Programowanie w firmach .....</b>	<b>379</b>
Działy przetwarzania danych i IT .....	379
Stanowiska związane z komputerami .....	383
Nazwy stanowisk .....	383
Ustrukturyzowane przeglądy .....	389
Przenoszenie programu do środowiska produkcyjnego .....	390
Konsulting .....	392
Podsumowanie .....	393
Pytania i odpowiedzi .....	393
Warsztaty .....	393
<b>Godzina 23. Rozpowszechnianie aplikacji .....</b>	<b>395</b>
Kwestie związane z rozpowszechnianiem aplikacji .....	395
Korzystanie z systemu kontroli wersji .....	398
Podsumowanie .....	399
Pytania i odpowiedzi .....	399
Warsztaty .....	399

<b>Godzina 24. Przyszłość programowania .....</b>	<b>401</b>
Przydatne narzędzia .....	401
Czy programowanie przestanie być potrzebne? .....	404
Wymóg ciągłego dokształcania się .....	407
Podsumowanie .....	408
Pytania i odpowiedzi .....	408
Warsztaty .....	409
Skorowidz .....	411

## Godzina 3.

# Projektowanie programu

Programiści już na początku kariery uczą się cierpliwości. Zauważają, że aby program stał się sukcesem, musi być właściwie zaprojektowany. Możliwe, że zetknąłeś się już z określeniem **analiza i projektowanie systemów**. Jest to nazwa nadana procesowi analizowania problemu i projektowania na tej podstawie systemów. Z pewnością chcesz wrócić do praktycznych ćwiczeń z programowania — wkrótce będziesz mieć ku temu okazję. Aby jednak móc produktywnie programować, musisz najpierw zrozumieć znaczenie projektu. W tej godzinie staramy się omówić najważniejsze aspekty projektowania programów i pokazać Ci, przez co przechodzą wydajni programiści przed rozpoczęciem pisania programów.

Oto najważniejsze zagadnienia omawiane w tej godzinie:

- ▶ znaczenie projektu programu;
- ▶ trzy kroki potrzebne do pisania programów;
- ▶ definicja danych wyjściowych;
- ▶ projektowanie w modelach „od ogółu do szczegółu” i „od szczegółu do ogółu”;
- ▶ wyjaśnienie, w jaki sposób schematy blokowe i pseudokod pozwalają stosować narzędzia RAD;
- ▶ przygotowywanie się do ostatniego kroku w procesie programowania.

## Dlaczego potrzebny jest projekt?

Gdy budowlańcy zaczynają stawiać dom, nie biorą młotka i nie zaczynają od zbijania mebli kuchennych. Zanim będzie można zacząć prace, projektant musi zaprojektować nowy dom. Wkrótce zobaczysz, że także program przed napisaniem trzeba zaprojektować.

Wykonawca musi najpierw ustalić, czego oczekuje inwestor, który zleca budowę. Niczego nie można zbudować, dopóki kierownik budowy nie będzie miał w umyśle wyniku końcowego. Dlatego inwestor musi się spotkać z architektem, by przedstawić mu swoje oczekiwania co do wyglądu domu. Architekt pomaga klientowi podejmować decyzje, informując go, co jest możliwe, a co nie jest. Na tym początkowym etapie zawsze istotna jest też cena, ponieważ projektant i inwestor muszą osiągnąć w tej kwestii kompromis.

Gdy architekt opracuje już plany domu, wykonawca musi zaplanować, jakie zasoby będą potrzebne do postawienia domu. Dopiero po ukończeniu projektu, uzyskaniu pozwoleń, zapewnieniu finansowania, zakupieniu materiałów i zatrudnieniu pracowników można rozpocząć fizyczny proces budowy. Im więcej wysiłku wykonawca włoży we wstępne prace, tym szybciej będzie mógł potem zakończyć budowę domu.

Problem z budową domu przed przygotowaniem odpowiedniego projektu związany jest z tym, że przyszli właściciele mogą chcieć wprowadzić poprawki na etapie, gdy zmiany są już niemożliwe. Bardzo trudno jest dodać łazienkę pomiędzy dwiema sypialniami *po* ukończeniu budowy. Celem jest więc uzgodnienie ostatecznej wersji domu przez właściciela i wykonawcę przed rozpoczęciem budowy. Gdy wszystkie zainteresowane strony uzgodnią specyfikację, ryzyko późniejszych sporów jest małe. Im bardziej przejrzyste są początkowe plany, tym mniej problemów pojawia się później, ponieważ wszystkie strony wyraziły zgodę na te same projekty domu.

Oczywiście nie jest to książka poświęcona budowaniu domów, ale gdy piszesz większe aplikacje, pamiętaj o podobieństwach między omawianymi dziedzinami. Nie należy siadać do klawiatury i rozpoczynać wpisywania instrukcji w edytorze przed zaprojektowaniem programu; podobnie budowniczy nie powinien podnosić młotka przed ukończeniem projektu domu.

### Wskazówka Wskazówka

Im więcej wstępnych prac projektowych wykonasz, tym szybciej ukończysz program.

Dzięki technologiom informatycznym programy komputerowe są łatwiejsze w modyfikacji niż budynki. Jeśli pominiemy procedurę, na której zależy użytkownikowi, będziesz mógł dodać ją później w łatwiejszy sposób niż pokój do gotowego domu. Mimo to dodawanie elementów do programu nigdy nie jest tak łatwe jak pisanie kodu po poprawnym zaprojektowaniu aplikacji już za pierwszym razem.

## Umowa między użytkownikiem a programistą

Załóżmy, że przyjmujesz zlecenie od małej firmy, która chce ulepszyć swoją witrynę (po ukończeniu wszystkich 24 lekcji będziesz lepiej rozumieć programowanie i nauczysz się nawet tego, jak pisać programy sieciowe w Javie). Oczekiwane przez klienta zmiany w witrynie wydają się proste. Firma chce, żebyś napisał interaktywne procedury w Javie umożliwiające użytkownikom sprawdzenie przez internet stanu magazynu i wydrukowanie zamówienia z listą produktów, którą można przynieść do sklepu w celu dokonania zakupu.

Po wysłuchaniu oczekiwań klienta uzgadniasz cenę za usługę, otrzymujesz zaliczkę, pobierasz pliki z kodem obecnych stron i udajesz się na kilka dni do domu, gdzie pracujesz. Po tych kilku dniach wyczerpującej pracy przynosisz wspaniale przygotowane strony internetowe do pokazania klientowi.

— Wyglądają dobrze — mówi klient. — Ale gdzie jest pole do wpisywania numerów kart kredytowych? W jaki sposób użytkownik może zamówić produkty przez internet? Dlaczego strona nie wyświetla produktów, które firma dopiero zamówiła i które nie są jeszcze dostępne? I dlaczego są podane tylko ceny netto?

Właśnie otrzymałeś bolesną lekcję związaną z umowami między użytkownikiem a programistą. Użytkownicy kiepsko poradzi sobie z wyjaśnieniem swoich oczekiwań. Na ich obronę można powiedzieć, że Ty też nie postarałeś się wydobyć z nich informacji o tym, czego naprawdę potrzebują. Obie strony uznały, że wiesz, co masz zrobić, ale okazało się to nieprawdą. Teraz widzisz, że pierwotna cena, jakiej zażądałeś, pokrywa tylko około 10% rzeczywistej pracy niezbędnej przy tym projekcie.

Zanim zaczniesz pracę i ją wycenisz, musisz ustalić, czego użytkownicy oczekują. Nauczenie się tego to jeden z aspektów nabywania doświadczenia w zakresie projektowania programów. Musisz ustalić każdy szczegół, zanim będziesz mógł precyzyjnie wycenić swoją pracę i zadowolić klientów.

### Uwaga

Odpowiednia umowa między użytkownikiem a programistą jest niezbędna we wszystkich obszarach programowania. Dotyczy to nie tylko programistów kontraktowych. Także jeśli pracujesz dla korporacji, przed rozpoczęciem prac musisz uzyskać szczegółową specyfikację. Inni pracownicy korporacji, którzy będą korzystać z danego systemu, muszą zatwierdzić spisane oczekiwania, tak by wszyscy od początku wiedzieli, czego można się spodziewać. Jeżeli użytkownik później przyjdzie do Ciebie i zapyta, dlaczego nie dodałeś jakiejś funkcji, będziesz mógł odpowiedzieć: „Ponieważ nigdy na jej temat nie rozmawialiśmy; zatwierdziłeś specyfikację, w której nie ma mowy o tej funkcji”.

Konserwacja programu, mająca miejsce po jego napisaniu, przetestowaniu i udostępnieniu, to jeden z najbardziej czasochłonnych aspektów procesu programowania. Programy są nieustannie aktualizowane, by odzwierciedlić nowe oczekiwania użytkowników. Czasem, jeśli program nie został poprawnie zaprojektowany przed jego napisaniem, użytkownik nie będzie chciał z niego korzystać, dopóki ów program nie zacznie wykonywać potrzebnych zadań.

Konsultanci z branży informatycznej szybko się uczą, aby przed rozpoczęciem programowania uzyskać akceptację projektu przez użytkownika (najlepiej na piśmie z podpisem). Uzgodnienie przez użytkownika i programistów tego, co trzeba zrobić, zmniejsza ryzyko sporów po zaprezentowaniu gotowego programu. Zasoby firmy są ograniczone. Nie ma czasu na późniejsze dodawanie funkcji, które od początku powinny znajdować się w systemie.

## Etapy projektowania

Są trzy podstawowe etapy, przez które należy przejść w procesie pisania programu:

1. Definiowanie danych wyjściowych i przepływu danych.
2. Opracowanie logiki potrzebnej do uzyskania tych danych wyjściowych.
3. Napisanie programu.

Zauważ, że pisanie programu to *ostatni* krok w procesie tworzenia aplikacji. Nie jest to tak dziwne, jak może się wydawać. Pamiętaj, że fizyczne stawianie domu to ostatni etap w jego budowaniu. Przed jego rozpoczęciem bardzo istotne jest odpowiednie zaplano-

wanie prac. Odkryjesz, że samo wpisywanie wierszy programu to jeden z najłatwiejszych aspektów procesu programowania. Jeśli projekt jest dobrze przemyślany, program niemal „sam się pisze”. Wprowadzanie kodu staje się jedynie formalnością.

## Etap 1. Definiowanie danych wyjściowych i przepływu danych

Przed rozpoczęciem programowania musisz mieć dobre pojęcie o tym, co program powinien generować i jakie dane wejściowe są niezbędne do uzyskania potrzebnych danych wyjściowych. Podobnie jak budowniczy musi przed rozpoczęciem budowy domu znać jego docelowy wygląd, programista przed przystąpieniem do pisania kodu musi wiedzieć, jakie dane wyjściowe są oczekiwane. Wszystko, co program generuje i co widzi użytkownik, jest uważane za dane wyjściowe, które należy zdefiniować. Musisz wiedzieć, jak ma wyglądać każdy ekran programu i co znajdzie się na poszczególnych stronach wszystkich drukowanych raportów.

Niektóre programy są małe, ale bez wiedzy o tym, dokąd zmierzasz, ukończenie programu może zająć Ci więcej czasu, niż gdybyś najpierw szczegółowo określił dane wyjściowe. Załóżmy, że chcesz napisać w JavaScriptcie program, który umożliwi użytkownikom witryny podanie danych kontaktowych. Najpierw należy przygotować listę wszystkich pól, które program ma wyświetlać na ekranie. Trzeba nie tylko wymienić wszystkie pola, ale też je opisać. W tabeli 3.1 znajdziesz szczegóły dotyczące pól z okna omawianego programu.

TABELA 3.1. Pola, które może wyświetlać program do zarządzania danymi kontaktowymi

Pole	Typ	Opis
Lista kontaktów	Lista przewijana	Wyświetla listę osób kontaktowych
Nazwisko	Pole tekstowe	Zawiera nazwisko osoby kontaktowej
Adres	Pole tekstowe	Zawiera adres osoby kontaktowej
Miejscowość	Pole tekstowe	Zawiera miejscowość osoby kontaktowej
Województwo	Pole tekstowe	Zawiera województwo osoby kontaktowej
Kod pocztowy	Pole tekstowe	Zawiera kod pocztowy osoby kontaktowej
Telefon domowy	Pole tekstowe	Zawiera numer telefonu domowego osoby kontaktowej
Telefon komórkowy	Pole tekstowe	Zawiera numer telefonu komórkowego osoby kontaktowej
Adres e-mail	Pole tekstowe	Zawiera adres e-mail osoby kontaktowej
Etap	Stałe wartości, lista przewijana	Wyświetla listę możliwych etapów, na jakich może znajdować się dana osoba kontaktowa (możliwe, że dopiero nawiązano pierwszy kontakt z nią lub zaproponowano jej specjalną dodatkową rozmowę)
Uwagi	Pole tekstowe	Różne uwagi na temat osoby kontaktowej (na przykład o tym, czy zakupiła coś wcześniej od firmy)

TABELA 3.1. Pola, które może wyświetlać program do zarządzania danymi kontaktowymi — ciąg dalszy

Pole	Typ	Opis
Filtrowanie kontaktów	Stałe wartości, lista przewijana	Umożliwia użytkownikowi wyszukiwanie grup osób kontaktowych na podstawie etapu (pozwala to na przykład wyświetlić listę wszystkich osób, do których wysłano e-mail)
Edytuj	Przycisk polecenia	Umożliwia użytkownikowi modyfikację istniejących danych osoby kontaktowej
Dodaj	Przycisk polecenia	Umożliwia użytkownikowi dodanie nowej osoby kontaktowej
OK	Przycisk polecenia	Umożliwia użytkownikowi zamknięcie okna z osobami kontaktowymi

Wiele pól wymienionych w **definicji danych wyjściowych** może być oczywistych. Pole o nazwie Nazwi sko oczywiście zawiera i pozwala wyświetlać nazwisko osoby kontaktowej. Nie ma nic złego w tym, że coś jest oczywiste. Pamiętaj, że jeśli piszesz programy dla innych osób (co zdarza się często), musisz uzyskać akceptację parametrów aplikacji. Jedną z najlepszych metod jest zacząć od listy wszystkich pól tworzonego programu i upewnić się, że użytkownik potwierdza, iż obejmuje ona wszystkie potrzebne pozycje. Możliwe, że klient ma nietypowe potrzeby — na przykład chce dostępu do kontaktów z Twittera. Dzięki rozmowie z klientem lepiej zrozumiesz, co powinieneś dodać do programu.

W dalszej części tej godziny, w podrozdziale „Narzędzia RAD”, zobaczysz, jak za pomocą programów zbudować model ekranu z danymi wyjściowymi, który można pokazać użytkownikom. Ten model razem z listą pól pozwala dwa razy sprawdzić, czy program zawiera dokładnie to, czego użytkownik potrzebuje.

Okna na dane wyjściowe, na przykład ekran do wprowadzania danych w programie do obsługi kontaktów, to także część definicji danych wyjściowych. Może się to wydawać sprzeczne z intuicją, ale ekrany na dane wyjściowe wymagają wyświetlania przez program pól na ekranie, dlatego należy zaplanować, gdzie umieścić te pola.

Tworzenie definicji danych wyjściowych to coś więcej niż etap wstępny projektowania danych wyjściowych, ponieważ pozwala uzyskać wgląd w to, jakie elementy danych należy rejestrować, obliczać i generować w programie. Ten etap pomaga też ustalić wszystkie dane wejściowe potrzebne do wygenerowania danych wyjściowych.

### Ostrzeżenie

Niektóre programy generują bardzo dużą ilość danych wyjściowych. Nie pomijaj omawianego tu pierwszego i niezwykle ważnego etapu procesu projektowania tylko dlatego, że ilość danych wyjściowych jest duża. Im więcej jest danych wyjściowych, tym ważniejsze jest ich zdefiniowanie. Jest to stosunkowo prosty proces (czasem nawet nudny), ale czasochłonny. Czas potrzebny na zdefiniowanie danych wyjściowych może być porównywalny z czasem wpisywania programu. Jeśli jednak pominiemy początkowe definiowanie danych wyjściowych, możesz stracić o wiele więcej czasu.

Definicja danych wyjściowych obejmuje wiele stron szczegółowych informacji. Musisz móc określić specyfikację wszystkich szczegółów problemu przed określeniem, jakich danych wyjściowych będziesz potrzebować. Nawet przycisk poleceń i pola z przewijanymi listami to dane wyjściowe, ponieważ program je wyświetla.

Z godziny 1., „Praktyczne ćwiczenia z programowania”, wiesz, że dane trafiają do programu, a ten zwraca znaczące informacje. Powinieneś przygotować spis wszystkich danych przekazywanych do programu. Jeśli dodajesz kod w JavaScriptcie do witryny, by zwiększyć jej interaktywność, musisz wiedzieć, czy właściciele witryny chcą pobierać dane od użytkowników. Zdefiniuj, jakie są to dane. Możliwe, że witryna umożliwi użytkownikowi podanie nazwiska i adresu e-mail, pod który będzie można przysyłać cotygodniowe wiadomości z ofertami. Czy firma chce pobierać od użytkowników dodatkowe dane, na przykład adres zamieszkania, wiek lub dochód?

### Projektowanie obiektowe

Z tego 24-godzinnego samouczka dowiesz się, czym jest **programowanie obiektowe**. Podejście to polega na przekształcaniu wartości (na przykład nazw i cen) w obiekty, które istnieją jako samodzielne jednostki w programach. Podstawy programowania obiektowego opisaliśmy w części III, „Programowanie obiektowe z użyciem Javy”.

Parę lat temu kilku ekspertów od programowania obiektowego opracowało proces projektowania programów obiektowych. Proces ten nazwano **projektowaniem obiektowym**. Projektowanie obiektowe to zaawansowana nauka określania danych, które są potrzebne w programie, i definiowania tych danych w sposób odpowiedni dla potrzeb programistów obiektowych. Jednym ze słynnych twórców projektowania obiektowego był Grady Booch. To jego specyfikacje sprzed dwudziestu lat nadal pomagają programistom obiektowym określać dane dla planowanych aplikacji i przekształcać te dane w obiekty programów.

W godzinie 4., „Pobieranie danych wejściowych i wyświetlanie danych wyjściowych”, dowiesz się, jak uwzględnić zdobytą wiedzę w programie. Zobaczysz, w jaki sposób program żąda podania danych i wyświetla informacje na ekranie. Przetwarzanie **wejścia – wyjścia** to najbardziej krytyczny aspekt aplikacji. Ważne jest, by pobrać wszystkie potrzebne dane i zrobić to precyzyjnie.

W tej dyskusji na temat projektowania nadal czegoś brakuje. Znasz już wagę pobierania danych. Masz świadomość, jak ważne jest zaprojektowanie danych wyjściowych w celu ustalenia, co jest potrzebne. Jak jednak przejść od danych wejściowych do danych wyjściowych? To następny krok w procesie projektowania. Musisz określić, jaki proces przetwarzania jest potrzebny do wygenerowania danych wyjściowych na podstawie danych wejściowych. Musisz przygotować właściwy przepływ danych i odpowiednie obliczenia, tak by program operował danymi wejściowymi i generował poprawne dane wyjściowe. W ostatnich podrozdziałach z tej godziny opisaliśmy, jak opracować najważniejszą część programu — logikę.



Wszystkie ekrany z danymi wyjściowymi, drukowane raporty i ekrany do wprowadzania danych muszą być zdefiniowane wcześniej, abyś dokładnie wiedział, co jest potrzebne w programach. Musisz też zdecydować, jakie dane zachować w plikach i jaki będzie ich format. Wraz z postęпами w edukacji programistycznej poznasz sposoby zapisywania plików danych w potrzebnych formatach.

W trakcie zbierania danych należy przyjmować je od użytkowników w sensowny sposób wymagający niewiele czasu. Prośby o dane powinny być przyjazne i niekłopotliwe. W ich przygotowaniu pomocne mogą być tworzenie prototypów (opisane w następnym punkcie) i narzędzia RAD.

## Tworzenie prototypów

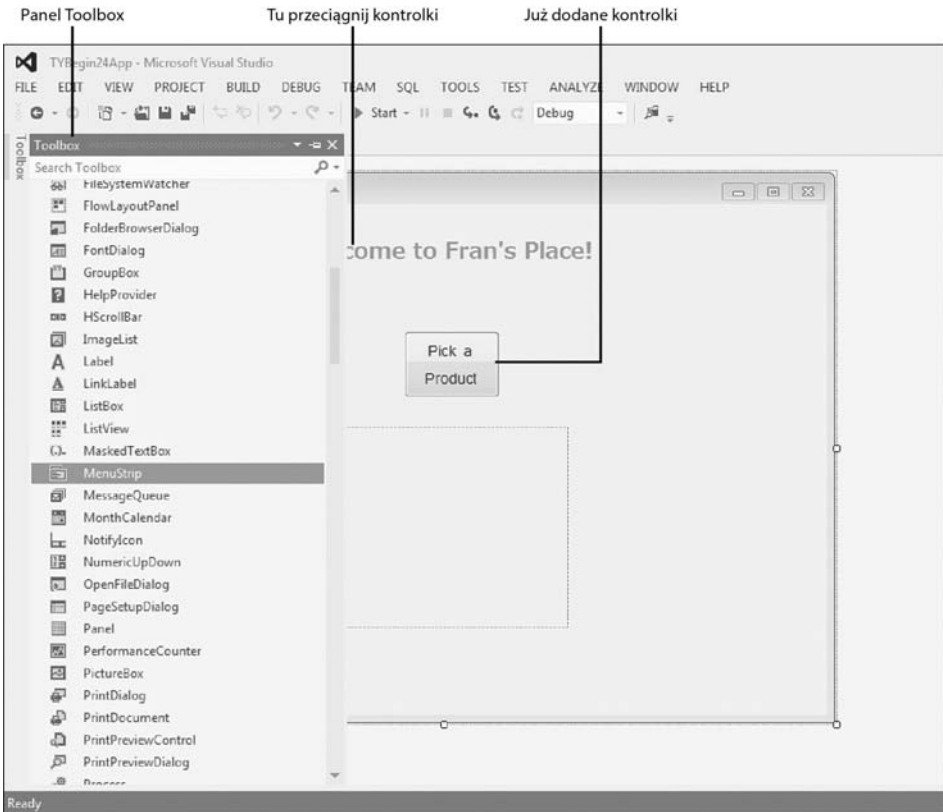
W czasach, gdy sprzęt i czas korzystania z komputerów były kosztowne, proces projektowania systemów pod niektórymi względami był ważniejszy niż obecnie. Im więcej czasu poświęcano na zaprojektowanie kodu, tym bardziej płynny był kosztowny proces programowania. Obecnie jest to prawdą w mniejszym stopniu, ponieważ komputery są tańsze i programiści mają dużo większą niż wcześniej możliwość zmiany zdania i dodania opcji programu. Mimo to w pierwszej części tej godziny szczegółowo wyjaśniliśmy, dlaczego przygotowanie projektu przed przystąpieniem do programowania jest tak ważne.

Podstawowy problem w podejściu wielu początkujących programistów polega na tym, że w ogóle pomijają etap projektowania. Jest to przyczyną wielu kłopotów, takich jak wspomniany wcześniej w tej godzinie, związany z tym, że firma oczekiwała od witryny znacznie więcej, niż programista mógł sobie wyobrazić.

Choć projektowanie danych wyjściowych, danych wejściowych, a nawet logiki programu jest znacznie prostsze dzięki dostępnym obecnie narzędziom informatycznym i ich niskim cenom, nadal trzeba zadbać o przygotowanie wstępnego projektu z danymi wyjściowymi uzgodnionymi z użytkownikami. Przed przystąpieniem do pisania kodu musisz też określić wszystkie dane, jakie program ma zbierać. Jeśli tego nie zrobisz, narazisz się na frustrację (niezależnie od tego, czy będziesz programistą kontraktowym czy etatowym w korporacji), ponieważ będziesz musiał stale dodawać funkcje, których użytkownicy oczekują, ale o których nie poinformowali.

Jedną z zalet systemu operacyjnego Windows jest jego wizualny charakter. Przed powstaniem tego systemu narzędzia programistyczne umożliwiały tylko tekstowe projektowanie i implementowanie rozwiązań. Obecnie, gdy projektujesz ekran dla klienta, możesz wybrać język programowania taki jak Visual Basic, narysować ekran i przeciągnąć na niego obiekty (na przykład przycisk OK), z którymi użytkownik będzie wchodził w interakcje. Pozwala to szybko zaprojektować **ekrany prototypowe**, które można przesłać użytkownikowi. Prototyp to model, a prototypowy ekran jest modelem tego, jak będzie wyglądał ekran w gotowym programie. Gdy użytkownik zobaczy ekran, z którym ma wchodzić w interakcję, będzie mu znacznie łatwiej ustalić, czy rozumiesz wymagania stawiane programowi.

JavaScript (podobnie jak języki programowania dla systemu Windows takie jak Visual C++ i Visual Basic) udostępnia narzędzia do tworzenia prototypów. Na rysunku 3.1 pokazaliśmy ekran do tworzenia programu w języku Visual Basic. Ekran zawiera wiele



**RYSUNEK 3.1.** Systemy do tworzenia programów, na przykład Visual Basic, udostępniają narzędzia pozwalające w wizualny sposób tworzyć definicje danych wyjściowych

elementów; zwróć uwagę na panel *Toolbox* i projekt okna wyjściowego. Aby umieścić kontrolki (na przykład przyciski poleceń i pola tekstowe) na formularzu reprezentującym okno wyjściowe, programista musi tylko przeciągnąć daną kontrolkę z panelu *Toolbox* na formularz. Tak więc by zbudować okno wyjściowe programu, programista musi jedynie przeciągnąć potrzebne kontrolki na formularz. Nie musi w tym celu napisać ani jednego wiersza kodu.

Po umieszczeniu kontrolki w oknie formularza za pomocą narzędzia programistycznego takiego jak Visual Basic można wyjść poza wyświetlanie formularza użytkownikom. Formularz można skompilować (podobnie jak cały program) i umożliwić użytkownikom interakcję z kontrolkami. Gdy użytkownik może korzystać z kontrolki, to nawet jeśli nie prowadzi to do żadnych efektów, pozwala klientowi stwierdzić, czy programista zrozumiał wymagania stawiane programowi. Użytkownik często zauważa, czy w programie czegoś nie brakuje. Może też przedstawić sugestie pozwalające ułatwić poruszanie się po programie z perspektywy użytkownika.

### Ostrzeżenie

Prototyp często jest tylko pustą powłoką, które nie robi nic oprócz symulowania interakcji z użytkownikiem. Dopiero później formularz jest łączony z kodem. Wraz z uzyskaniem akceptacji ekranów zadanie programisty dopiero się zaczyna, jednak ekrany to punkt wyjścia, ponieważ musisz ustalić, czego użytkownicy oczekują, zanim będziesz mógł pójść dalej.

## Narzędzia RAD

Bardziej zaawansowanym produktem do projektowania programów, umożliwiającym definiowanie danych wyjściowych, przepływu danych i samej logiki, są **narzędzia RAD** (ang. *Rapid Application Development*). Choć te narzędzia znajdują się jeszcze w powijakach, zapewne w ciągu swej kariery będziesz z nich korzystać — zwłaszcza gdy staną się bardziej popularne i tańsze.

RAD to proces szybkiego rozmieszczania kontrolki na formularzu (przebiega to podobnie jak pokazaliśmy wcześniej na podstawie Visual Basica), łączenia tych kontrolki z danymi i generowania fragmentów gotowego kodu. Pozwala to uzyskać w pełni funkcjonalną aplikację bez napisania choćby jednego wiersza kodu. Pod pewnymi względami systemy programistyczne takie jak Visual Basic realizują wiele celów stawianych narzędziom RAD. Gdy umieszczasz kontrolkę na formularzu — co szczegółowo przedstawiliśmy w godzinie 20., „Programowanie w języku Visual Basic 2012” — Visual Basic obsługuje wszystkie aspekty programistyczne potrzebne dla danej kontrolki. Nigdy nie musisz pisać żadnego kodu, by przycisk polecenia działał tak, jak powinien. Jedynym celem programisty jest ustalenie, ile przycisków poleceń potrzeba w programie i gdzie należy je umieścić.

Omawiane narzędzia nie potrafią jednak czytać w myślach. Narzędzia RAD nie wiedzą, że kliknięcie przez użytkownika określonego przycisku ma prowadzić do wydrukowania raportu. Programiści są potrzebni do łączenia poszczególnych elementów ze sobą i z danymi, a także do pisania szczegółowej logiki pozwalającej poprawnie przetwarzać dane. Przed pojawieniem się tego rodzaju narzędzi do budowania programów programiści musieli pisać tysiące wierszy kodu (często w języku C), aby utworzyć prostą aplikację dla systemu Windows. Teraz można przynajmniej szybko przygotować kontrolki i interfejs. Możliwe, że któregoś dnia narzędzia RAD będą na tyle zaawansowane, że będą potrafiły generować także logikę. Ale do tego czasu nie rzucaj pracy programisty, ponieważ wciąż będzie zapotrzebowanie na Twoje usługi.

### Wskazówka

Nauucz użytkowników, jak tworzyć prototypy ekranów! Do projektowania ekranów nie jest potrzebna wiedza z zakresu programowania. Dzięki prototypom użytkownicy będą mogli dokładnie pokazać Ci, czego chcą. Ponadto prototypy ekranów są interaktywne. Oznacza to, że użytkownicy będą mogli klikać przyciski i wprowadzać wartości w polach, choć w efekcie nic się nie stanie. Pomysł polega na tym, by umożliwić użytkownikom wypróbowanie ekranów i upewnić się w ten sposób, że klientom odpowiada rozmieszczenie i wygląd kontrolki.

## Projektowanie programów metodą od ogółu do szczegółu

W dużych projektach wielu programistów stwierdza, że projektowanie metodą od ogółu do szczegółu pomaga skoncentrować się na tym, co w aplikacji jest potrzebne, i ułatwia szczegółowe opracowanie logiki koniecznej do uzyskania wyników programu. **Projektowanie od ogółu do szczegółu** (ang. *top-down design*) to proces rozbijania ogólnego problemu na coraz mniejsze części do momentu określenia wszystkich szczegółów. W tym modelu określane są szczegóły potrzebne do wykonania zadania programistycznego.

Problem z tym podejściem polega na tym, że programiści zwykle go nie stosują. Przeważnie posługują się odwrotnym modelem, czyli **projektowaniem od szczegółu do ogółu** (ang. *bottom-up design*). Gdy ignorujesz projektowanie od ogółu do szczegółu, obciążasz się koniecznością zapamiętania wszystkich potrzebnych szczegółów. Jeśli stosujesz podejście od ogółu do szczegółu, szczegóły same się pojawiają. Nie musisz wtedy martwić się o drobiazgi, ponieważ uzyskujesz szczegóły w wyniku stosowania omawianego procesu.

### Wskazówka

Jednym z ważnych aspektów projektowania od ogółu do szczegółu jest to, że zmusza do odłożenia szczegółów na później. Proces ten sprawia, że programista przez możliwie długi czas myśli w kategoriach ogólnego problemu. Projektowanie od ogółu do szczegółu pozwala utrzymać koncentrację. Jeśli stosujesz projektowanie od szczegółu do ogółu, narażasz się na to, że przestaniesz widzieć las zza drzew. Zbyt szybko zajmiesz się wtedy szczegółami i utracisz z oczu podstawowe wymagania stawiane programowi.

Oto trzyetapowy proces niezbędny w trakcie projektowania od ogółu do szczegółu:

1. Ustalenie ogólnego celu.
2. Rozbicie celu na bardziej szczegółowe elementy (dwa, trzy lub więcej). Zbyt duża liczba szczegółów spowoduje, że niektóre z nich zostaną pominięte.
3. Odkładaj zajmowanie się szczegółami tak długo, jak to możliwe. Powtarzaj kroki 1. i 2. do momentu, w którym nie zdołasz już rozbić podproblemów na mniejsze części.

Łatwiej zrozumiesz projektowanie od ogółu do szczegółu, jeśli przed przyjrzeniem się problemowi informatycznemu zastosujesz je do zadania z rzeczywistego świata. Omawiane tu podejście jest przeznaczone nie tylko dla problemów programistycznych. Gdy już opanujesz projektowanie od ogółu do szczegółu, będziesz mógł zastosować ten model do dowolnego aspektu życia, który musisz szczegółowo zaplanować. Prawdopodobnie najbardziej szczegółowym wydarzeniem, które człowiek może zaplanować, jest ślub. Dlatego jest to doskonały przykład odzwierciedlający, jak wygląda projektowanie od ogółu do szczegółu w praktyce.

Co jest pierwszą rzeczą potrzebną, by wziąć ślub? Najpierw należy znaleźć potencjalną partnerkę lub potencjalnego partnera (aby uzyskać pomoc w tym zakresie, będziesz potrzebować innej książki). Gdy przychodzi czas na planowanie ślubu, projektowanie od ogółu do szczegółu jest najlepszym sposobem podejścia do zadania. Metodą, której *nie* należy stosować, jest martwienie się najpierw szczegółami. A jednak wiele osób zaczyna

właśnie od nich. Narzeczeni myślą wtedy najpierw o strojach, zespole, wystroju sali i potrawach serwowanych gościom na przyjęciu weselnym. Największy problem pojawiający się przy próbie uwzględniania od początku wszystkich tego typu szczegółów polega na tym, że wiele rzeczy przestaje być widocznych. Zbyt łatwo jest wtedy zapomnieć o niektórych szczegółach i przypomnieć sobie o nich dopiero wtedy, gdy jest już za późno. Dzieje się tak, ponieważ uwaga jest zajęta innymi detalami.

Jaki jest ogólny cel ślubu? W najbardziej ogólnym ujęciu można powiedzieć, że po prostu „wziąć ślub”. Jeśli odpowiadasz za zaplanowanie ślubu, ogólny cel „wziąć ślub” zapewni Ci właściwy kierunek. Załóżmy, że na najwyższym poziomie cel to właśnie „wziąć ślub”.

### Uwaga

Ogólny cel pomaga Ci utrzymać koncentrację. Mimo swej nadmiarowej natury cel „wziąć ślub” pozwala zrezygnować z uwzględniania takich szczegółów jak planowanie miesiąca miodowego. Jeśli nie odgraniczysz rozwiązywanego problemu od innych zadań, możesz zacząć zajmować się niepotrzebnymi drobiazgami i, co ważniejsze, pominąć istotne szczegóły. Jeżeli planujesz zarówno ślub, jak i miesiąc miodowy, opracuj dwa projekty od ogółu do szczegółu lub uwzględnij podróz poślubną w najbardziej ogólnym celu. Plan ślubu dotyczy tylko tego wydarzenia (ceremonii zaślubin i wesela), natomiast nie powinien uwzględniać szczegółów miesiąca miodowego. Zajęcie się szczegółami podróży poślubnej możesz pozostawić partnerce lub partnerowi, dzięki czemu może czekać Cię niespodzianka. W końcu masz wystarczająco dużo pracy z planowaniem ślubu, prawda?

Teraz, gdy wiesz już, dokąd zmierzasz, zacznij rozbijać ogólny cel na dwa lub trzy bardziej szczegółowe punkty. W jakich kolorach zrobić wystrój sali, kogo zaprosić na wesela, co z opłatami dla księdza... — eh, szczegółów jest za dużo! W projektowaniu od ogółu do szczegółu chodzi o to, by jak najdłużej odkładać zajmowanie się drobiazgami. Nie spiesz się. Gdy zauważysz, że rozbijasz problem z danego poziomu na więcej niż trzy lub cztery części, zapewne zanedbujesz się spieszyć. Poczekaj ze szczegółami. Cel „wziąć ślub” można rozbić na dwa podstawowe komponenty — ceremonię zaślubin i przyjęcie weselne.

Następny krok projektowania od ogółu do szczegółu polega na podzieleniu nowych komponentów na części. W ramach ceremonii należy uwzględnić ludzi i miejsce. W przypadku wesela należy ustalić menu, ludzi i miejsce. Ludzie związani z ceremonią to goście, narzeczeni i obsługa (ksiądz, organista i tak dalej, ale tymi szczegółami zajmiesz się później).

### Wskazówka

Na razie nie martw się kwestiami związanymi z czasem. Celem projektowania od ogółu do szczegółu jest uzyskanie (ostatecznie) wszystkich potrzebnych szczegółów, a nie uporządkowanie ich w kolejności. Musisz wiedzieć, dokąd zmierzasz, a także co jest potrzebne. Dopiero potem możesz zastanowić się nad tym, jak szczegóły są powiązane ze sobą i jaki jest ich porządek chronologiczny.

Ostatecznie uzyskasz kilka stron szczegółów, których nie da się już podzielić. Prawdopodobnie otrzymasz szczegóły związane między innymi z potrawami weselnymi (na przykład orzeszkami do podgryzania). Jeśli zaczniesz od razu wypisywać takie szczegóły, wiele z nich może Ci umknąć.

Teraz przejdź do bardziej informatycznego problemu. Załóżmy, że otrzymałeś zadanie napisania dla firmy programu kadrowo-płacowego. Czego wymaga taki program? Możesz zacząć od wymienienia szczegółów takiej aplikacji:

- ▶ drukowanie czeków z wypłatami,
- ▶ obliczanie podatków państwowych,
- ▶ obliczanie podatków lokalnych.

Co złego jest w tym podejściu? Jeśli odpowiedziałeś, że zbyt wczesne przechodzenie do szczegółów, masz rację. Najlepiej zacząć od ogólnego poziomu. Najbardziej ogólnym celem programu kadrowo-płacowego może być „opracować listę płac”. Ten ogólny cel pozwala pominąć inne szczegóły programu (nie trzeba uwzględniać przetwarzania księgi głównej, chyba że jakaś część systemu płacowego aktualizuje plik z księgą główną) i skoncentrować się na rozwiązywanym problemie.

Przyjrzyj się rysunkowi 3.2. Może to być pierwsza strona projektu związanego z listą płac i uzyskanego metodą od ogółu do szczegółu. Każdy program płacowy musi obejmować mechanizmy do wprowadzania, usuwania i modyfikowania informacji o pracownikach — imienia i nazwiska, adresu, dni zwolnienia itd. Jakie jeszcze szczegółowe dane o pracownikach są potrzebne? Na tym etapie nie należy się tym zajmować. Projekt nie jest jeszcze do tego gotowy.



**RYSUNEK 3.2.** Pierwsza strona uzyskanego metodą od ogółu do szczegółu projektu programu kadrowo-płacowego obejmuje szczegóły z najwyższego poziomu

Czeka Cię jeszcze długa droga do ukończenia projektu związanego z płacami. Rysunek 3.2 to pierwszy etap. Musisz kontynuować dzielenie poszczególnych komponentów do momentu pojawienia się szczegółów.

Dopiero gdy razem z użytkownikami ustalicie wszystkie potrzebne szczegóły (za pomocą projektowania od ogółu do szczegółu), możesz zdecydować, jak powinny one wyglądać.

## Etap 2. Tworzenie logiki

Gdy razem z użytkownikami uzgodnicie cele i dane wyjściowe programu, reszta zależy od Ciebie. Twoje zadanie polega na tym, by na podstawie definicji danych wyjściowych ustalić, jak sprawić, aby komputer wygenerował te dane. Przyjrzałeś się ogólnemu problemowi i rozbiłeś go na szczegółowe instrukcje, które komputer może wykonać. Nie oznacza to, że jesteś już gotowy do pisania programu — wprost przeciwnie. Teraz przyszedł czas na opracowanie logiki, która wygeneruje dane wyjściowe.

Definicja danych wyjściowych dobrze opisuje, *co* program ma robić. Teraz musisz zdecydować, *jak* ma się to odbywać. Musisz uporządkować ustalone szczegóły, by operacje odbywały się w określonym porządku chronologicznym. Ponadto musisz ustalić, jakie decyzje program musi podejmować i jakie operacje mają wynikać z poszczególnych decyzji.

W pozostałej części tego 24-godzinnego samouczka opanujesz dwa ostatnie kroki rozwijania programów. Zdobędziesz wgląd w to, jak programiści piszą i testują programy po opracowaniu definicji danych wyjściowych i zatwierdzeniu specyfikacji przez użytkowników.

### Ostrzeżenie

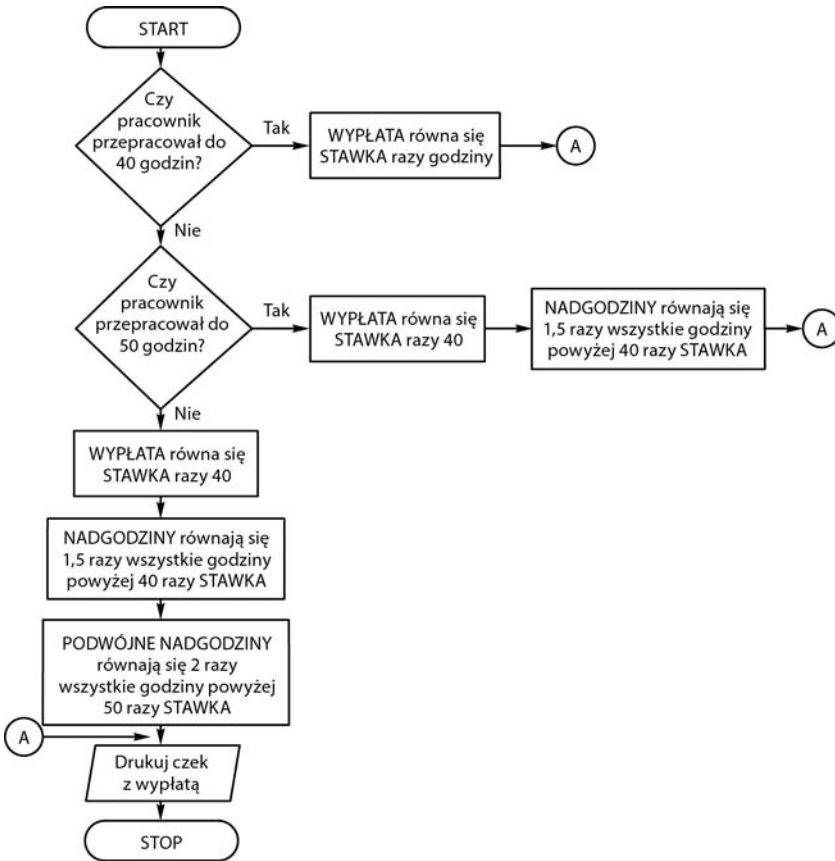
Ostrzeżenie

Dopiero gdy opanujesz programowanie, będziesz mógł nauczyć się wbudowywać logikę w program. Musisz jednak przygotować logikę przed napisaniem programu, by móc przejść od etapu definicji danych wyjściowych i wejściowych do kodu aplikacji. Jest to problem „jajka i kury”, z którym zmagają się wielu początkujących programistów. Gdy zaczniesz pisać własne programy, znacznie lepiej zrozumiesz proces rozwijania logiki.

W przeszłości użytkownicy posługiwali się narzędziami takimi jak **schematy blokowe** i **pseudokod** do tworzenia logiki programu. Schemat blokowy pokazaliśmy na rysunku 3.3. Mówi się, że obraz jest wart tysiąc słów. Pokazany tu schemat blokowy reprezentuje ogólny przepływ logiki w programie. Jeśli schemat blokowy jest poprawnie narysowany, pisanie samego programu staje się bardzo proste. Po ukończeniu programu schemat blokowy może pełnić funkcję dokumentacji.

Schematy blokowe składają się z symboli standardowych dla branży. Istnieją programy, które wspomagają tworzenie schematów blokowych i pozwalają je wydrukować.

Choć niektóre osoby wciąż posługują się schematami blokowymi, narzędzia RAD i inne narzędzia programistyczne prawie wyeliminowały takie schematy. Schematy stosuje się głównie do ilustrowania izolowanych części logiki programu na potrzeby dokumentacji. Nawet w okresie ich największej popularności (w latach 60. i 70. ubiegłego wieku) schematy blokowe nie były powszechnie używane. Niektóre firmy wołały stosować opisy logiki w **pseudokodzie** (nazywanym czasem **ustrukturyzowanym angielskim**), co polega na reprezentowaniu logiki za pomocą zdań, a nie przy użyciu potrzebnych w schematach blokowych diagramów.



RYSUNEK 3.3. Schemat blokowy przedstawia graficznie logikę programu kadrowo-płacowego

W pseudokodzie nie występują żadne instrukcje z języka programowania, ale nie jest on też pisany zwykłym językiem naturalnym. Obejmuje on zestaw ściśle określonych słów, który umożliwia opis logiki. Słowa te (w języku angielskim) często występują na schematach blokowych i w językach programowania. Pseudokod (podobnie jak schematy blokowe) można pisać dla dowolnych zadań, nie tylko dla programów komputerowych. W wielu instrukcjach obsługi jakaś odmiana pseudokodu jest używana do ilustrowania kroków potrzebnych do składania części. Pseudokod umożliwia ścisły opis logiki bez pozostawiania miejsca na wieloznaczność.

Poniżej pokazaliśmy logikę programu kadrowo-płacowego przedstawioną za pomocą pseudokodu. Zauważ, że możesz przeczytać i zrozumieć tekst, bo nie jest on zapisany w języku programowania. Wcięcia pomagają określić, które zdania są ze sobą połączone. Ten pseudokod jest czytelny dla każdego (nawet dla osób, które nie znają symboli ze schematów blokowych).

Dla każdego pracownika:

- Jeśli pracownik przepracował od 0 do 40 godzin, to
  - płaca brutto równa się przepracowane godziny razy stawka.
- W przeciwnym razie



jeśli pracownik przepracował od 40 do 50 godzin, to  
płaca brutto równa się 40 razy stawka;  
plus (godziny przepracowane - 40) razy stawka razy 1,5.

W przeciwnym razie  
płaca brutto równa się 40 razy stawka;  
plus 10 razy stawka razy 1,5;  
plus (godziny przepracowane - 50) razy stawka razy 2.

Odejmij od płacy brutto podatek.

Wydrukuj czek z wypłatą.

## Etap 3. Pisanie kodu

Nauka pisania programu zajmuje najwięcej czasu. Gdy jednak już się tego nauczysz, sam proces programowania zajmuje mniej niż projektowania (jeśli projekt jest precyzyjny i kompletny). Natura programowania wymaga opanowania pewnych nowych umiejętności. Dzięki kilku następnym godzinnym lekcjom nauczysz się wiele o językach programowania i wykonasz ćwiczenia, by stać się lepszym programistą. To sprawi, że tworzone przez Ciebie programy nie tylko będą realizować stawiane im cele, ale też będą proste w konserwacji.

## Podsumowanie

Budowniczy nie stawia domu przed jego zaprojektowaniem. Programista też nie powinien pisać programu przed opracowaniem projektu. Programiści zbyt często siadają do klawiatury bez wcześniejszego przemyślenia logiki. Źle zaprojektowany program skutkuje wieloma błędami i długą konserwacją. W tej godzinie opisaliśmy, jak zapewnić, by projekt programu był dopasowany do oczekiwań użytkowników. Po przygotowaniu definicji danych wyjściowych możesz uporządkować logikę programu za pomocą projektu od ogółu do szczegółu, schematów blokowych i pseudokodu.

Następna godzina przede wszystkim pozwoli Ci przećwiczyć korzystanie z pierwszego języka programowania — JavaScriptu.

## Pytania i odpowiedzi

**P: W jakim momencie tworzenia projektu w modelu od ogółu do szczegółu należy zacząć dodawanie szczegółów?**

**O:** Odkładaj to najdłużej jak to możliwe. Jeśli projektujesz program do generowania raportów o sprzedaży, nie przechodź do etapu drukowania ostatecznego podsumowania raportu, jeśli nie masz ukończonych wszystkich innych zadań. Szczegóły pojawiają się same, gdy nie da się już rozbić zadania na co najmniej dwa inne.

**P: Czy po rozbiciu projektu na szczegóły z najniższego poziomu nie uzyskują też szczegółów pseudokodu?**

**O:** Projekt w modelu od ogółu do szczegółu to narzędzie do ustalania wszystkich szczegółów potrzebnych w programie. Projekt ten nie określa jednak logicznej kolejności przetwarzania tych szczegółów. Pseudokod wyznacza logikę wykonywania programu i określa,

kiedy operacje są przeprowadzane, w jakiej kolejności i kiedy należy je zakończyć. Projekt w modelu od ogółu do szczegółu obejmuje wszystko, co może się zdarzyć w programie. Zamiast pisać pseudokod rozważ zastosowanie narzędzia RAD, ponieważ pomoże Ci ono szybciej przejść od projektu do gotowego działającego programu. Obecnie systemy RAD są jeszcze stosunkowo prymitywne, dlatego dużą część kodu będziesz musiał dodać samodzielnie.

## Warsztaty

Pytania quizowe mają pomóc Ci w lepszym zrozumieniu materiału.

### Quiz

1. Dlaczego przygotowanie poprawnego projektu często zajmuje więcej czasu niż samo pisanie programu?
2. Na jakim etapie programista zaczyna określać wymagania użytkowników?
3. Poprawne projektowanie w modelu od ogółu do szczegółu wymaga odkładania ustalania szczegółów najdłużej jak to możliwe – prawda czy fałsz?
4. Czym projekt w modelu od ogółu do szczegółu różni się od pseudokodu?
5. Jakie jest przeznaczenie narzędzi RAD?
6. Do systemu projektowanego za pomocą narzędzi RAD nie trzeba dodawać żadnego kodu – prawda czy fałsz?
7. Symbole stosowane są w schematach blokowych czy w pseudokodzie?
8. Na schematach blokowych można przedstawiać zarówno logikę programu, jak i procesy z rzeczywistego świata.
9. Jeśli pozwolisz użytkownikom wypróbować prototyp danych wyjściowych, pomogą Ci w opracowaniu danych wyjściowych programu – prawda czy fałsz?
10. Jaki jest ostatni etap procesu programowania (przed testami końcowych wyników)?

### Odpowiedzi

1. Im dokładniejszy jest projekt, tym szybciej programiści mogą napisać program.
2. Programista często robi to wtedy, gdy zaczyna definiować dane wyjściowe proponowanego systemu.
3. Prawda.
4. Projektowanie w modelu od ogółu do szczegółu umożliwia projektantowi stopniowe określanie wszystkich aspektów wymagań stawianych programowi. Pseudokod to sposób reprezentowania logiki programu w momencie, gdy projekt programu został już przygotowany (za pomocą narzędzi takich jak projektowanie od ogółu do szczegółu).

5. Narzędzia RAD umożliwiają szybkie tworzenie systemów i przechodzenie od etapu projektu do gotowego produktu. Te narzędzia nie są jeszcze na tyle zaawansowane, by wykonywać większość zadań z zakresu programowania, choć mogą ułatwiać projektowanie systemów.
6. Fałsz. Po zakończeniu zadania przez narzędzia RAD w wielu sytuacjach potrzebne są jeszcze pewne prace programistyczne.
7. Symbole są używane w schematach blokowych.
8. Prawda.
9. Prawda.
10. Ostatnim etapem programowania jest pisanie kodu.



# Skorowidz

---

## A

- abstrakcja, 338
- adres
  - URL, 286
  - zmiennej, 72
- agile, 48
- AJAX, Asynchronous JavaScript and XML, 211, 281
- akumulator, 150, 155
- algorytm, 149
- ampersand, 240
- analityk systemów, 387
- analityk-programista, 384
- analiza
  - kodu, 216
  - sortowania bąbelkowego, 160
- API, Application Programming Interface, 129, 212
- apka, 205
- aplet, 205, 213, 251, 258
- aplikacje
  - Javy, 205, 211
  - okienkowe, 370
  - sieciowe, 283
- argument, 240, 247, 312
- ASCII, 87
- atrybut
  - CODE, 259
  - HEIGHT, 259
  - WIDTH, 259
- automatyczne
  - testy, 403
  - uruchamianie apletów, 209
  - wykonywanie, 208

## B

- Backbone.js, 284
- back-end, 282
- bajt, 87
- BCL, Base Class Library, 365

- bezpieczeństwo, 212
  - ze względu na typ, 366
- biblioteka
  - AJAX-owa, 287
  - BCL, 365
  - FCL, 364, 365
  - Graphics, 253
  - javax.swing, 243
  - jQuery, 284
  - Prototype, 284
  - Swing, 212, 241
- binarna reprezentacja liczby, 94
- bit, 87
- blok kodu, 230
- błędy, bugs, 27, 119
  - czasu wykonania, 124
  - logiczne, 121
  - składniowe, 121
- breakpoints, 131

## C

- C#, 363, 366
- CASE, Computer-Aided Software Engineering, 405
- certyfiakat, 384, 385
  - witryny, 213
- CGI, Common Gateway Interface, 208
- chmura, 374
- ciasteczka, 188
- ciągła integracja, 404
- ciągłe wdrażanie, 404
- CIL, Common Intermediate Language, 364
- CLR, Common Language Runtime, 364
- CRUD, 321
- CSS, Cascading Style Sheets, 269, 274
- CSS3, 265
- CTS, Common Type System, 364
- czujki, watch variables, 131

**D**

dane, 20, 33  
 wyjściowe, 43, 54  
 debugger, 122  
 debugowanie, 119, 130  
 w JavaScriptcie, 125  
 decyzje, 138  
 definicja danych wyjściowych, 55  
 definiowanie  
 danych, 221  
 danych wyjściowych, 54  
 funkcji, 313  
 klas, 340  
 przepływu danych, 54  
 deklaracja  
 funkcji, 313  
 klasy, 340  
 obiektu, 340  
 tablicy, 152  
 zmiennych obiektowych, 340  
 dekrementacja, 308  
 DLR, Dynamic Language Runtime, 364  
 dodatek Firebug, 122  
 dodawanie  
 grafiki, 276  
 komentarzy, 299  
 instrukcji, 357  
 kontrolek, 373  
 kontrolek do formularza, 353  
 operacji do obiektów, 342  
 DOM, Document Object Model, 282  
 dostęp  
 do plików cookie, 189  
 do składowych, 341  
 do właściwości klasy, 319  
 do zmiennej, 315  
 dyplom, 384  
 dyrektor, 389  
 dyrektywa #include, 346  
 dyrektywy preprocesora, 328  
 dział IT, 380  
 dziedziczenie klas, 347, 338  
 dziennik konsoli JavaScriptu, 129  
 dźwięk, 212

**E**

edytor zasobów, 402  
 ekran, 57  
 z formularzem, 352  
 ekrany prototypowe, 57  
 Ember.js, 284  
 etapy projektowania, 53

**F**

FCL, Framework Class Library, 364, 365  
 format  
 instrukcji, 230  
 stron internetowych, 271  
 swobodny, 29  
 formatowanie tekstu, 272  
 formularz, 289, 320, 352  
 freeware, 395  
 front-end, 282  
 FTP, File Transfer Protocol, 297  
 funkcja, 141, 166, 217, 312, *Patrz także* metoda  
 ajaxRequest, 288  
 ajaxResponse, 288  
 checkAnswer(), 292  
 getQuestions(), 292  
 main(), 249, 328, 336  
 Math.atn(), 99  
 Math.exp(), 100  
 Math.floor(), 98  
 Math.log(), 99  
 Math.round(), 98  
 nextQuestion(), 292  
 printf(), 330  
 scanf(), 333  
 scrollingMsg, 182  
 setType(), 305  
 strcpy(), 333, 341  
 zwracająca wartość, 314  
 funkcje  
 do sterowania przepływem, 299  
 liczbowe, 98  
 okienkowe, 242  
 składowe, 342  
 w C, 330  
 wbudowane, 312, 330

**G**

gra Autocrazy, 213  
 graficzny interfejs użytkownika, GUI, 241, 402  
 grafika, 276  
 2D i 3D, 212  
 gramatyka języka, 24  
 GUI, Graphical User Interface, 402

**H**

hardware, 24  
 hiperłącze, 265  
 HTML, Hypertext Markup Language, 205, 265  
 HTML5, 265

**I**

IaaS, Infrastructure as a Service, 374  
IDE, Integrated Development Environment, 218, 402  
informacje, 20, 33  
  o błędzie, 127  
inkrementacja, 150, 308  
instancja klasy, 340  
instrukcja, 25, 27, 39, 42, 297  
  #include, 328  
  break, 300  
  console.error(), 130  
  console.log(), 130  
  console.warn(), 130  
  cout, 339  
  default, 300  
  do...while, 302  
  document.write, 71  
  echo, 297  
  End Sub, 360  
  for, 303  
  function, 313  
  global, 315  
  goto, 137  
  if, 103, 104, 230  
  if...else, 337  
  if-else, 232  
  print(), 297  
  prompt, 77  
  przypisania, 73  
  return, 314  
  while, 301  
instrukcje sterujące w C, 337  
interakcje z bazami danych, 320  
interaktywność zdjęć, 176  
interfejs  
  API, 129, 212  
  CGI, 208  
  Javy, 211  
  programowania aplikacji, 212  
  Windows API, 255  
interpretowanie danych, 287  
iteracje, 139, 301

**J**

Java, 203  
JavaScript, 24, 25, 27, 69, 281  
JDBC, Java Database Connectivity, 212  
język programowania, 20, 24, 33, 46, 263  
  C, 325  
  C#, 363, 366  
  C++, 325

CIL, 364  
COBOL, 119  
HTML, 265  
Java, 203  
  maszynowy, 26  
  PHP, 295  
  UML, 406  
  Visual Basic, 57, 349  
  Visual C#, 367

jQuery, 284  
JSON, JavaScript Object Notation, 282  
JVM, Java Virtual Machine, 209

**K**

kaskadowe arkusze stylów, CSS, 269  
katalog Temporary Internet Files, 188  
kierownik, 389  
klasa, 237, 317, 338, 340  
  bazowa, 347  
  Box, 245  
  Color, 256  
  Employee, 243  
  JOptionPane, 243  
  pochodna, 244, 347  
  String, 346  
  z metodą, 319  
klauzula  
  case, 300  
  elseif, 300  
  if...elseif...else, 300  
klient, 282  
  FTP, 297  
  SCP, 297  
klucz  
  prywatny, 213  
  publiczny, 213  
kod, 24  
  bajtowy, 208  
  kontrolki, 357, 373  
  spaghetti, 137  
  w C, 327  
  w HTML-u, 268  
  wywołujący, 248  
  źródłowy, 26  
kody  
  ASCII, 87  
  EBCDIC, 88  
kolekcje, 212  
komentarze, 27, 239, 299  
  lokalizacja, 28  
  stosowanie, 28  
kompilator, 26, 326

kompilowanie aplikacji, 396  
 konkatenacja, 85  
 konserwacja programu, 124  
 konsola JavaScriptu, 122, 123  
 konstrukt, 137  
 konsulting, 392  
 kontekst określający urządzenie graficzne, 255  
 kontrolka, 58, 353
 

- Label, 354
- MenuStrip, 359
- PictureBox, 354

 kończenie apletu, 253  
 kropka, 306

## L

licencja, 36
 

- na program, 24

 licencje grupowe, 24  
 liczniki, 150  
 lider zespołu, 389  
 literały, 90, 221, 269, 329
 

- całkowitoliczbowe, 221
- logiczne, 222
- zmiennoprzecinkowe, 221

 logika programu, 63, 64  
 lokalizacja komentarzy, 28

## Ł

łańcuch znaków, 77, 85, 332  
 łączenie
 

- HTML-a z PHP, 298
- łańcucha znaków, 77

## M

maszyna JVM, 209  
 mechanizm
 

- interaktywny, 179
- zmieniania zdjęć, 173

 mechanizmy języka Java, 214  
 metaznacznik, 271  
 metoda, 96, 217, 317, 338, *Patrz także* funkcja
 

- checkCookie(), 197
- document.getElementById, 175
- document.write, 84
- doubleIt(), 248
- drawString(), 218, 256
- escape, 190
- GET, 283
- init(), 217
- od ogółu do szczegółu, 60
- onclick(), 177
- onmouseover, 179

open(), 286  
 paint(), 218, 255  
 parseFloat(), 90  
 parseInt(), 90  
 POST, 283  
 prompt, 75, 78  
 resize(), 217, 253, 262  
 send(), 286  
 setColor(), 218, 256  
 setTimeout, 182  
 String.fromCharCode, 95  
 metody
 

- obiektów, 319
- tekstowe, 97
- uzupełnień do dwóch, 93

 metodyki programowania zwinnego, 48  
 młodszy programista, 386  
 modelowanie danych, 406  
 moduł kodu, 360  
 modyfikacja przycisku, 82

## N

narzędzia, 401
 

- CASE, 405
- dla programistów, 126
- do debugowania, 119
- projektowe, 42
- RAD, 59, 67

 nazwa
 

- klasy, 240
- pliku, 240
- zmiennej, 72, 303

 negacje liczb, 93  
 NetBeans, 219, 237  
 niestandardowe komunikaty, 130  
 nowy projekt, 238, 350

## O

obciążenia zwrotne, chargeback, 382  
 obiekt, 316, 338, 340
 

- cin, 339
- cout, 339
- typu XMLHttpRequest, 285

 obiekty strumienia, 339  
 obliczenia, 92
 

- matematyczne, 89

 obsługa
 

- baz danych, 212
- kasy, 78
- mysz, 80
- plików cookie, 189
- sieci, 212
- zdarzeń, 361



obszar aktywny, 265  
 oczekiwanie na odpowiedź, 286  
 odsyłacze, 277  
 odzyskiwanie pamięci, 365  
 ograniczenia AJAX-a, 284  
 okno
 

- dialogowe, 84
- formularza, 58, 352
- projektu, 350
- Properties, 354, 356
- Toolbox, 353, 371
- zapytania, 75

 OOP, Object-Oriented Programming, 214  
 open source, 37  
 operacje, 342
 

- arytmetyczne, 92

 operator
 

- dekrementacji, 227
- inkrementacji, 227
- new, 249
- pobierania, 339
- postdekrementacji, 308
- postinkrementacji, 308
- przypisania, 198, 228, 306
- scalania, 306
- scalania łańcuchów, 185
- trójargumentowy, 301
- warunkowy, 229, 235
- wstawiania, 339

 operatory
 

- arytmetyczne, 228, 306
- logiczne, 310
- matematyczne, 89, 226
- priorytety, 310
- porównania, 309
- porównywania, 228
- relacji, 106
- w C, 336
- złożone przypisania, 307

 oprogramowanie typu open source, 37  
 organizacja W3C, 266  
 osobołata, 37  
 otwarty dostęp do kodu źródłowego, 397  
 otwieranie stron, 286

## P

pakiet, 216
 

- instalacyjny, 396
- MAMP, 296
- WAMP, 296
- XAMPP, 296

 pamięć RAM, 42

panel Toolbox, 58  
 pasek z powtarzаныmi informacjami, 180  
 pętla, 108, 301
 

- do...while, 116, 302
- for, 108, 116, 233, 303
- while, 114, 116, 232, 301

 pętle
 

- nieskończone, 140, 302
- zagnieżdżone, 170

 PHP, 295  
 pierwszy program, 26, 215  
 pisanie
 

- algorytmów, 149
- apletów, 251
- kodu, 65
- kodu obiektowego, 257
- warunków, 106

 platforma
 

- .NET, 363
- Backbone.js, 284
- Ember.js, 284
- przetwarzania równoległego, 366

 plik
 

- HTML, 289
- HTML z apletem, 258
- HTML z PHP, 298
- iostream.h, 338
- nagłówkowy, 333
- XML, 290
- z funkcją, 142
- z kodem w JavaScriptcie, 291

 pliki
 

- .cpp, 337
- .htm, 278
- .html, 175
- .js, 143
- .txt, 278
- cookie, 189
  - usuwanie, 193
  - wczytywanie, 192
  - właściwości, 190
  - zapisywanie, 190
- nagłówkowe, 328

 pobieranie danych wejściowych, 69, 333
 

- z klawiatury, 75

 podpis cyfrowy, 213  
 podprocedura Sub, 359  
 podstawy programowania, 133  
 polecenia
 

- HTML-a, 270
- znaczników, 269

 polecenie document.write, 70, 74

polimorfizm, 338, 345  
 powtórzenia, 139  
 precyzja, 120  
 precyzowanie instrukcji, 39  
 priorytety operatorów, 90, 310  
 procedura, 359  
 procesor, 43  
   tekstu, 44  
 profiler, 145, 401  
 profilowanie kodu, 145  
 program, 20  
   Calculator, 241  
   kadrowo-płacowy, 64  
   raport o sprzedaży, 155  
   w języku C#, 367  
   wykorzystujący funkcję, 167  
   wpisywanie ocen, 114  
   zarządzanie danymi kontaktowymi, 54  
 programista, 52, 386  
   full stack, 388  
   kontraktowy, 382  
 programowanie, 386  
   obiektywne, OOP, 45, 56, 201, 214, 246  
   sterowane testami, 403  
   strukturalne, 45, 135  
   w firmach, 379  
   w HTML-u, 265  
   w Javie, 203  
   w PHP, 295  
   w Visual Basicu, 358  
 programy  
   jako wskazówki, 38  
   pozyskiwanie, 36  
 projekt, 51, 350  
 projektowanie programu, 51  
   obiektywne, 56  
   od ogółu do szczegółu, 60  
   od szczegółu do ogółu, 60  
 prototyp, 57, 284, 342  
 przechowywanie danych, 71  
 przeciąganie, 212  
 przeciążanie operatorów, 345  
 przeglądarka apletów, 251  
 przekazywanie danych do metod, 248  
 przepływ danych, 54  
 przedstawianie wartości, 156  
 przesyłanie żądania, 286  
 przeszukiwanie tablic, 161  
 przetwarzanie  
   danych, 85  
   równoległe, 366  
   wejścia – wyjścia, 56

przycisk Toolbox, 354  
 przypisanie, 228  
 przypisywanie wartości, 73  
 pseudokod, 63  
 punkt przerwania, breakpoint, 131

## Q

quiz, 289, 291

## R

RAD, Rapid Application Development, 59  
 RAM, Random Access Memory, 42  
 rejestrowanie  
   pozycji kursora, 178  
   zdarzeń, 80  
 rodzaje stanowisk, 383  
 rollover, 80  
 rozmieszczanie kontrolek, 371  
 rozpowszechnianie aplikacji, 395  
   na urządzenia przenośne, 397  
 rozszerzenie Java Plug-in, 213  
 rzutowanie, 305

## S

SaaS, Software as a Service, 374  
 schemat blokowy, 63, 104  
   procedura sortowania bąbelkowego, 160  
   wyszukiwanie binarne, 165  
   wyszukiwanie sekwencyjne, 162  
 SCP, Secure Copy, 297  
 sekwencja, 138  
   ucieczki, 222, 235  
 serwer, 282  
 serwlet, 205  
 shareware, 395  
 składnia, 24  
 składowa, 243, 245, 340  
 skrypt, 82, 282  
   instalacyjny, 396  
   w PHP, 295  
 słaba kontrola typów, 305  
 słowa kluczowe w języku C, 326  
 słowo kluczowe  
   boolean, 225  
   class, 340  
   goto, 137  
   Private, 359  
   public, 344  
   String, 225  
   var, 73  
 software, 24

- sortowanie, 157
  - bąbelkowe, 158
- stała, 311
- stanowisko, 383
- starszy
  - analityk systemów, 388
  - programista, 384, 387
- sterowanie
  - pętlą for, 112
  - programem, 103, 230
  - przepływem, 299
- stos wywołań, 131
- stosowanie
  - komentarzy, 28
  - odstępów, 125
  - odsyłaczy, 277
- strona internetowa, 251
- struktura, 137
- strumień
  - wejścia, 339
  - wyjścia, 339
- style CSS, 274
- Swing, 212, 241
- symbole konwersji, 331, 332
- system kontroli wersji, 398
  - Git, 398
  - Subversion, 398
- system operacyjny, 44
- systemy MIS, 404
- szkielet
  - apletu, 252
  - kodu, 239
- sztuczna inteligencja, 46

## Ś

- średnik, 303
- środowisko
  - CLR, 364
  - DLR, 366
  - IDE, 219
  - IDE, 402
  - produkcyjne, 390

## T

- tabela kodów
  - ASCII, 87
  - EBCDIC, 88
- tablice, 96, 226
  - równoległe, 154
  - znaków, 96

- techniki
  - debugowania, 130
    - programowania, 187
    - programowania strukturalnego, 135

### technologia

- .NET, 363
- AJAX, 211
- CSS3, 260
- HTML5, 260
- JDBC, 212

- terminator instrukcji, 303
- terminologia obiektowa, 338

### testowanie

- oprogramowania, 144
- quizu, 292

### testy

- równoległe, 145
- wersji beta, 144

- tryb tekstowy, 242

### tworzenie

- apletu, 252
- aplikacji, 351
- biblioteki AJAX-owej, 287
- funkcji, 141, 142
- funkcji w PHP, 312
- klasa, 317
- logiki, 63
- nowego projektu, 238
- obiektów, 249
- obiektu, 317
- pakietu instalacyjnego, 396
- prototypów, 57
- quizu, 289
- typów danych, 346
- własnych stylów, 274
- złożonych wyrażeń, 309
- żądania, 285

- typ danych, 223, 305

- array, 305
- boolean, 225, 305
- byte, 223
- char, 225
- double, 224, 305
- float, 224
- int, 223
- integer, 305
- long, 223
- object, 305
- resource, 305
- short, 223
- string, 305
- String, 225

- typy zmiennych całkowitoliczbowych, 223

**U**

udostępnianie programów, 24  
 układ pamięci, 44  
 UML, Unified Modeling Language, 406  
 umowa, 52  
 Unicode, 88  
 uruchomienie programu, 78, 237  
 usuwanie plików cookie, 193  
 uzupełnienie do dwóch, 93  
 użytkowanie Javy, 210  
 użytkownik, 20, 52

**V**

Visual Basic, 349  
 Visual C#, 367

**W**

wartość  
   bezwzględna, 99  
   NaN, 77  
   null, 77, 332  
   programów, 23  
 warunek, 106  
 watch variables, 131  
 wczytywanie plików cookie, 192, 320  
 wersja beta, 144  
 wielkość liter, 97  
 wielokrotne użycie, 338  
 właściciel programu, 24  
 właściwości  
   obiektów, 318  
   plików cookie, 190  
 właściwość document.cookie, 189  
 wpisywanie programu, 29  
 wprowadzanie  
   danych, 385  
   łańcuchów znaków, 77  
 wspólny  
   język pośredni, CIL, 365  
   systemów typów, CTS, 364  
 wykonywanie metod, 246  
 wyrażenie, 98, 309  
 wyszukiwanie  
   binarne, 163  
   sekwencyjne, 161, 162  
 wyświetlanie  
   apletu, 259  
   danych na ekranie, 69  
   danych wyjściowych, 69  
   informacji w konsoli, 129  
   komunikatu, 111

składowych, 341  
 wartości, 110  
 właściwości obiektu, 318  
 wyników, 90  
 wyników obliczeń, 71  
 wywołanie funkcji, 313  
   przez wartość, 247

**X**

XML, Extensible Markup Language, 283

**Z**

zaawansowane techniki, 187  
 zapisywanie plików cookie, 190  
 zarządzanie  
   danymi kontaktowymi, 54  
   plikami, 187  
 zasięg  
   klasy, 343  
   zmiennych, 303, 315  
 zasób, 402  
 zastępowanie fragmentu łańcucha, 97  
 zastosowania  
   AJAX-a, 283  
   języka PHP, 320  
 zawartość wykonywalna, 206, 208  
 zdalne skrypty, 282  
 zdarzenia związane z myszą, 80  
 zdarzenie  
   onmouseover, 84, 179  
   onreadystatechange, 286  
   przeniesienia kursora, 80  
 zegary, 212  
 zintegrowane środowisko programowania,  
   IDE, 218  
 zlecenie, 52  
 złożone operatory przypisania, 307  
 zmienianie  
   wielkości liter, 97  
   zdjęć, 173  
 zmienna, 98, 221  
 \$\_COOKIE, 304  
 \$\_ENV, 304  
 \$\_FILES, 304  
 \$\_GET, 304  
 \$\_POST, 304  
 \$\_REQUEST, 304  
 \$\_SERVER, 304  
 \$\_SESSION, 304  
 zmienne  
   całkowitoliczbowe, 223  
   globalne, 223, 304  
   logiczne, 225

- lokalne, 223, 304
- pomocnicze, 157
- składowe, 245
- superglobalne, 304
- tablicowe, 152
- tymczasowe, 157
- zdefiniowane poza funkcją, 315
- zmiennoprzecinkowe, 224
- znakowe, 225
- znacznik, 258, 269
  - <!DOCTYPE html>, 271
  - </script>, 70
  - <a>, 277
  - <APPLET>, 258, 259
  - <BODY>, 258
  - <EMBED>, 258
  - <h>, 273
  - <H1>, 259
  - <HEAD>, 258
  - <HTML>, 258
  - <OBJECT>, 258
  - <questions>, 290
  - <script>, 70, 289
  - <span id="question">, 290
- znak
  - #, 346
  - null, 332
  - równości, 73
- znaki ASCII, 95
- zwracanie wartości, 314

## Ż

- żądanie
  - GET, 283
  - POST, 283



# PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW  
w działający bankomat!

**Dowiedz się więcej i dołącz już dzisiaj!**

<http://program-partnerski.helion.pl>

**Dziś mało kto potrafi obejść się bez komputera.** Te niezwykle pożyteczne urządzenia służą do pracy i rozrywki, komunikowania się, diagnozowania chorób, zarządzania firmą, a nawet prowadzenia wojen. Właściwie trudno byłoby wyobrazić sobie dziedzinę, w której komputery i to, co potrafią, byłyby zbędne. Właśnie dlatego we współczesnym świecie umiejętność programowania jest bardzo cennym atutem. Zapotrzebowanie na programistów wciąż rośnie, a najlepsi w tym fachu mogą liczyć na niezwykle atrakcyjne warunki pracy.

**Niniejszy samouczek pozwala** przyswoić podstawy programowania w ciągu 24 godzinnych lekcji, umożliwi solidne opanowanie podstaw i uczy poprawnego projektowania programów. Nie pominięto tu szczególnie ważnego przygotowania się do programowania ani zasad tworzenia oprogramowania w korporacjach, a równocześnie pokazano sposoby korzystania z najważniejszych technik programistycznych oraz kluczowe cechy najczęściej wykorzystywanych języków programowania.

### W książce omówiono:

- sprzęt i narzędzia, w tym środowiska programistyczne
- zarys historii i języki programowania
- programowanie aplikacji internetowych
- programowanie w korporacji i obowiązki programisty
- dobre praktyki projektowe
- perspektywy programowania

**Greg Perry** — jest programistą i nauczycielem programowania od ponad 20 lat. Słynie z tego, że uczy solidnych podstaw programowania i potrafi mówić o tej dziedzinie w sposób niezwykle przystępny.

**Dean Miller** — od ponad 20 lat jest autorem i redaktorem książek. Przyczynił się do wydania wielu bestsellerowych pozycji i serii w wydawnictwie Sams Publishing.



**Programowania można się nauczyć,  
wystarczają 24 godziny!**

sięgnij po **WIĘCEJ**



**KOD KORZYŚCI**

**Helion**

księgarnia Internetowa



<http://helion.pl>

zamówienia telefoniczne



**0 801 339900**



**0 601 339900**

**SAMS**

Helion SA  
ul. Kościuszki 1c, 44-100 Gliwice  
tel.: 32 230 98 63  
e-mail: [helion@helion.pl](mailto:helion@helion.pl)  
<http://helion.pl>

Sprawdź najnowsze promocje:  
📍 <http://helion.pl/promocje>  
Książki najchętniej czytane:  
📍 <http://helion.pl/bestsellery>  
Zamów informacje o nowościach:  
📍 <http://helion.pl/nowosci>

ISBN 978-83-283-2939-3



9 788328 329393

Informatyka w najlepszym wydaniu

cena: 59,00 zł