Greg Perry, Dean Miller

Programowanie
dla początkujących
w 24 godziny

SAMS

Helion

Tytuł oryginału: Beginning Programming in 24 Hours, Sams Teach Yourself (4th Edition)

Tłumaczenie: Tomasz Walczak

ISBN: 978-83-283-6797-5

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *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/prp244.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/prp244
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

CZĘŚĆ V INNE JĘZYKI PROGRAMOWANIA

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

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

Dzięki technologiom informatycznym programy komputerowe są łatwiejsze w modyfikacji niż budynki. Jeśli pominiesz 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 opracować oprogramowanie do zarządzania sprzedażą i magazynem (po ukończeniu wszystkich 24 lekcji będziesz lepiej rozumieć programowanie i nauczysz się nawet tego, jak pisać programy w Pythonie i używać innych języków). Oczekiwane przez klienta rozwiązania wydają się proste. Firma chce, żebyś napisał interaktywne procedury w Pythonie umożliwiające sprawdzenie stanu magazynu i wyświetlenie listy produktów, które zostały sprzedane w ostatnim dniu, tygodniu, miesiącu lub roku.

Po wysłuchaniu oczekiwań klienta uzgadniasz cenę za usługę, otrzymujesz zaliczkę, przygotowujesz projekt oprogramowania i udajesz się do domu, gdzie pracujesz. Po kilku miesiącach wyczerpującej pracy przynosisz fantastyczny program do pokazania klientowi.

— Wygląda dobrze — mówi klient. — Ale gdzie jest raport z podziałem na płatność gotówką i kartą? Gdzie można sprawdzić, jakie produkty są dostępne w sklepie, a jakie w magazynie? Gdzie program wyświetla produkty, które firma dopiero zamówiła i które nie są jeszcze dostępne? I dlaczego program nie potrafi dodać do ceny podatku VAT?

Właśnie otrzymałeś bolesną lekcję związaną z umowami między użytkownikiem a programistą. Użytkownicy kiepsko poradzili 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 zaplanowanie 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 Pythonie program, który umożliwia małej firmie zapisywanie i składowanie danych kontaktowych od klientów. 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
Usuń	Przycisk polecenia	Umożliwia użytkownikowi usunięcie danych istniejącej osoby kontaktowej

Wiele pól wymienionych w **definicji danych wyjściowych** może być oczywistych. Pole o nazwie `Nazwisko` 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 wejściowe, na przykład ekran do wprowadzania danych w programie do obsługi kontaktów, to także część definicji danych wyjściowych. Może się to wydawać sprzeczne z intuicja, ale ekrany na dane wejściowe wymagają wyświetlania przez program pól na ekranie, dlatego należy zaplanować, gdzie umieścić te pola.

Definicja danych wyjściowych to coś więcej niż wstępny projekt takich danych. Dzięki takiej definicji masz wgląd w to, jakie elementy danych program powinien rejestrować, obliczać i generować. Definiowanie danych wyjściowych pomaga też określić 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 pominiesz 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 piszesz kod w Pythonie, tworząc program do zarządzania danymi kontaktowymi klientów, musisz wiedzieć, jakie dane właściciele witryny chcą pobierać od użytkowników. Zdefiniuj, jakie są to dane. Możliwe, że właściciele zamierzają pytać użytkowników o to, czy zechcą podać nazwisko i adres e-mail, pod który będzie można przesyłać 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 postępami 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 bezproblemowy 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.

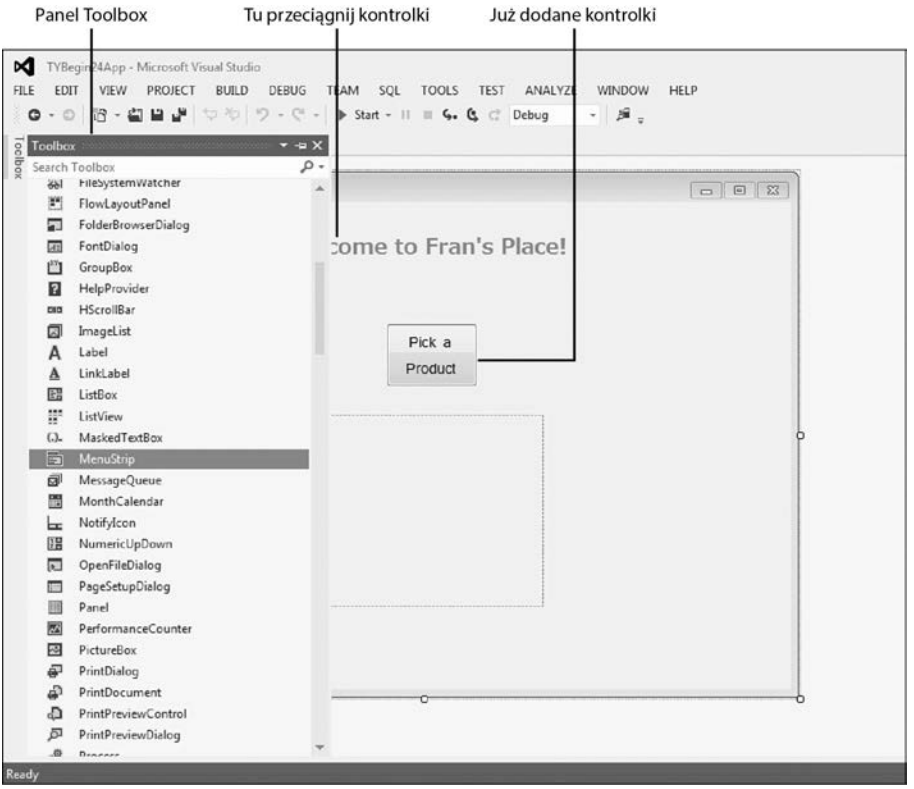
Podstawowy problem w podejściu wielu początkujących programistów polega na tym, że w ogóle pomijają etap projektowania. Jest to przyczyną wielu kłopotów, takich jak wspomniany wcześniej w tej godzinie, związany z tym, że firma oczekiwała od witryny znacznie więcej, niż programista mógł sobie wyobrazić.

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 interakcje, będzie mu znacznie łatwiej ustalić, czy rozumiesz wymagania stawiane programowi.

Wiele języków programowania dla systemu Windows, na przykład Visual C++ i Visual Basic, udostępnia narzędzia do tworzenia prototypów. Na rysunku 3.1 pokazaliśmy ekran do tworzenia programu w języku Visual Basic. Język omawiany w początkowych rozdziałach, Python, lepiej nadaje się do pracy na zapleczu — do przetwarzania danych i ich analizowa-

nia, gdy jest to potrzebne. Oczywiście możesz za pomocą Pythona pobierać i wyświetlać dane, jeśli jednak piszesz aplikację dla systemu Windows, lepsze będą inne języki, na przykład język użyty na rysunku 3.1. Ekran z rysunku zawiera wiele 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.



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

Po umieszczeniu kontrolek 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 kontrolek, 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*). RAD to proces szybkiego rozmieszczania kontrolek na formularzu (przebiega to podobnie jak pokazaliśmy wcześniej na podstawie Visual Basica), łączenia tych kontrolek 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

> Naucz 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 kontrolek.

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 podróż 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 wesele, co z opłatami dla księdza... — *ups*, 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 zanadto się spieszysz. 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.

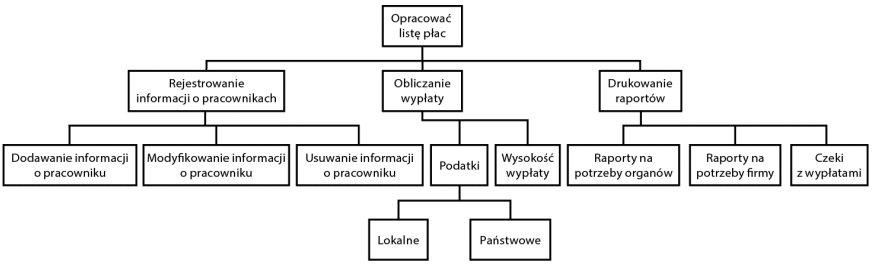
Ostatecznie uzyskasz kilka stron szczegółów, których nie da się już podzielić. Prawdopodobnie otrzymasz szczegóły związane między innymi z potrawami weselnymi (na przykład orzeszkami do podgryzania). Jeśli zaczniesz od razu wypisywać takie szczegóły, wiele z nich może Ci umknąć.

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 — adresu, miasta, kodu pocztowego, 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.

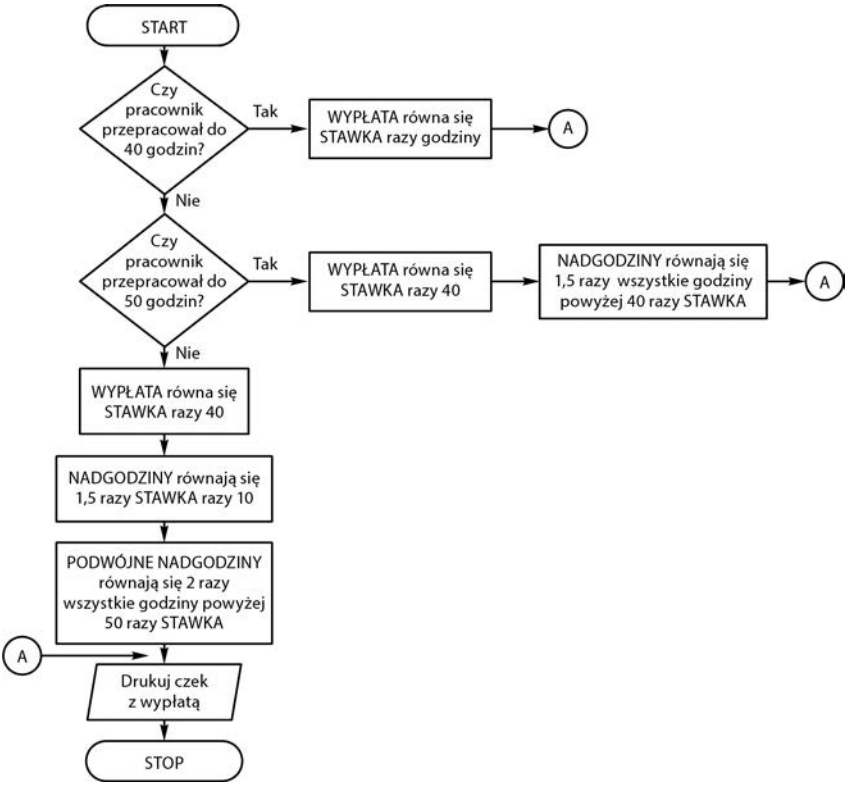
Ostrzeżenie

> 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 zmaga 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. W sklepach z przyborami biurowymi nadal dostępne są plastikowe *szablony symboli schematów blokowych*, dzięki którym schematy są bardziej atrakcyjne, niż gdyby tworzyć je ręcznie. Istnieją też 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 wolały 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 — Pythona.

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 uzyskuję 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.

Warto nauczyć się programowania! Poza możliwością znalezienia ciekawej i dobrze płatnej pracy czy pasjonującego hobby umiejętność programowania bywa niezwykle przydatna w rozwiązywaniu różnych problemów. Paleta języków programowania i narzędzi programistycznych jest niezwykle szeroka i praktycznie każdy znajdzie coś dla siebie. Zanim to jednak nastąpi, trzeba zdobyć trochę wiedzy i umiejętności. Ale bez obaw! W nauce programowania najtrudniejszy bywa pierwszy krok, jednak ta książka sprawi, że wykonasz go bez trudu i dumnie wkroczysz w świat kodowania!

To kolejne wydanie lubianego samouczka, dzięki któremu w ramach 24 godzinnych lekcji przyswoisz solidne podstawy programowania. Zrozumiesz, jak działają programy, i nauczysz się reguł stosowanych przez profesjonalistów przy ich projektowaniu. Dowiesz się, jak wygląda świat programistów i na czym dokładnie polega programowanie w korporacjach. Znajdziesz tutaj także wprowadzenie do kilku najpopularniejszych języków programowania, co pozwoli na ich porównanie i ułatwi wybór języka do dalszej nauki. Każdy z 24 rozdziałów zawiera materiał, który można opanować w ciągu godziny. Naukę ułatwiają instrukcje krok po kroku, quizy, ćwiczenia i praktyczne przykłady. Dzięki tej książce zdobędziesz najlepsze podstawy, aby stać się dobrym programistą. Przygotujesz się też do świadomego kształtowania swojej dalszej ścieżki zawodowej!

W książce między innymi:

- przygotowanie narzędzi do pracy — sprzęt i oprogramowanie
- podstawowe aspekty programowania i projektowania programów
- algorytmy, interaktywność, zmienne, funkcje
- debugowanie kodu
- programowanie obiektowe i korzystanie z baz danych
- planowanie kariery programisty

Greg Perry jest programistą i doświadczonym szkoleniowcem, znanym z wyjątkowo zrozumiałego prezentowania zagadnień z obszaru programowania. Jest też autorem kilkudziesięciu książek informatycznych. Często występuje na prestiżowych konferencjach programistycznych.

Dean Miller jest autorem książek i redaktorem z ponad dwudziestoletnim doświadczeniem. Współtworzył tak uznane serie wydawnicze jak *Dla każdego, W 24 godziny* i *Księga eksperta*.

Zacznij programować. Najlepiej od razu!

SAMS

Helion

helion.pl

HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

ISBN 978-83-283-6797-5

9 788328 367975

INFORMATYKA W NAJLEPSZYM WYDANIU

Cena: 69,00 zł