Wstęp

Być może stoisz właśnie w księgarni, trzymasz w ręku egzemplarz niniejszej książki i zadajesz sobie pytanie, czy rzeczywiście warto ją kupić. A może znalazłszy na półce biblioteczki w firmie, masz wątpliwości, czy warto poświęcić czas na jej lekturę. Wiem, że nie masz czasu, więc powiem krótko. Jeśli zastanawiasz się, jak podnieść poziom swoich programów w C++, jak radzić sobie z lawiną nieistotnych szczegółów panoszących się nawet w najlepiej zaprojektowanych systemach, jak tworzyć podprogramy, których będzie można używać wielokrotnie bez każdorazowego majstrowania w ich zakamarkach, to jest to książka dla Ciebie.

Wyobraź sobie następujący scenariusz. Wracasz z zebrania z plikiem schematów usianych dopiskami i notatkami. Typ `char` już nie wystarcza do przekazywania zdarzeń między obiektami; potrzeba `int`. Zmieniasz jeden wiersz kodu, inteligentne wskaźniki do obiektów typu `Widget` są zbyt wolne. Trzeba zrezygnować z kontroli poprawności wyłuskania. Zmieniasz jeden wiersz kodu. W sąsiednim dziale wprowadzono właśnie kod korzystający z obiektów typu `Gadget`; trzeba więc uzupełnić o nie fabrykę obiektów. Zmieniasz jeden wiersz kodu.

Kompilacja. Łączenie. Gotowe.

Hmm, coś jednak nie gra, prawda? O wiele bardziej prawdopodobny scenariusz wygląda tak: wracasz z zebrania w pośpiechu, bo masz masę roboty. Rzucasz się do sprawdzania wszystkiego. Tniesz i zszywasz na okrętkę. Dopisujesz kawałki kodu. Dodajesz nowe błędy..., na tym właśnie polega praca programisty, czyż nie? Choć książka ta nie zapewni Ci realizacji pierwszego scenariusza, to jednak umożliwi wykonanie pewnego kroku w jego kierunku. Stanowi próbę przedstawienia C++ jako nowo odkrytego języka dla architektów oprogramowania.

Tradycyjnie kod uważa się za najbardziej niejawną część systemu. Chociaż języki programowania dostarczają coraz silniejszego wsparcia dla metod projektowania (np. obiektowość między planami systemu a jego kodem istnieje wciąż olbrzymia przepaść, ponieważ kod musi specyfikować każdy szczegół. Zazwyczaj zaplanowana przez projektanta struktura rozmywa się w morzu szczegółów implementacyjnych.

W książce tej jest przedstawiony zbiór artefaktów konstrukcyjnych wielokrotnego użytku, zwanych *komponentami generycznymi* (ang. *generic component*) oraz techniki, dzięki którym można je tworzyć. Komponenty generyczne to odpowiednik bibliotek, ale na pozio-

mie architektury oprogramowania. Pokazane tu techniki kodowania i implemantacje są przedmiotem zadań i problemów tradycyjnie związanych z projektowaniem, czyli działań wykonywanych zazwyczaj *przed* kodowaniem. Ze względu na wysoki poziom abstrakcji komponenty generyczne umożliwiają odwzorowanie skomplikowanej architektury bezpośrednio za pomocą kodu, w wyjątkowo wyrazisty, zwięzły i łatwy do zmodyfikowania sposób.

Łączy się tu trzy elementy: wzorce projektowe, programowanie uogólnione oraz C++, by zmaksymalizować szanse wielokrotnego użycia (pionowego i poziomego). Jeśli chodzi o wykorzystanie poziome, kombinacje niewielkich kawałków przedstawionego kodu bibliotecznego dają możliwość zaimplementowania olbrzymiej, a w istocie nieograniczonej, liczby struktur i zachowań. W zastosowaniu pionowym zaś stopień ogólności komponentów czyni je zdatnymi do użycia w bardzo różnorodnych programach.

Wielki wpływ na tę książkę miały wzorce projektowe – skuteczne rozwiązania problemów często spotykanych w programowaniu obiektowym. Są one wyodrębnionymi fragmentami dobrych projektów – przepisami na solidne i nadające się do wielokrotnego użytku rozwiązania problemów. Wzorce projektowe zapewniają sugestywne słownictwo. Opisują problem, sprawdzone rozwiązania i ich warianty oraz konsekwnecje wyboru każdego z rozwiązań. Wykraczają poza wszystko, co może wyrazić dowolny język programowania. Oparte na wzorcach projektowych komponenty przedstawione w tej książce dają się zastosować do ogromnej klasy konkretnych problemów.

Programowanie uogólnione to paradygmat, w którym kładzie się nacisk na wyabstrahowanie typów do poziomu niewielkich zbiorów wymagań funkcjonalnych i na implementowanie algorytmów w kategoriach tych wymagań. Ponieważ takie algorytmy określają ścisły i zawężony interfejs do typów, z których korzystają, ten sam algorytm może być stosowany dla wartości różnych typów. W implementacjach omówionych w tej książce korzysta się z technik programowania uogólnionego, by jak najsłabiej związać się z konkretnymi rozwiązaniami, zapewnić zwięzłość oraz dużą wydajność mogącą konkurować z jakością kodu wycyzelowanego ręcznie.

Język C++ to jedyne używane tutaj narzędzie implementacyjne. Nie ma tu implementacji zmyślnego systemu okienkowego, złożonych bibliotek sieciowych ani sprytnego mechanizmu rejestrowania wykonania programu. Znajdziesz natomiast podstawowe komponenty ułatwiające implementację tego wszystkiego i nie tylko. Jezyk C++ ma mechanizmy, dzięki którym to wszystko jest osiągalne praktycznie. Model pamięci odziedziczony po C zapewnia wysoką wydajność. Wsparcie dla polimorfizmu umożliwia korzystanie z technik obiektowych. Szablony uruchamiają niesamowitą maszynerię do generowania kodu. Szablony przenikają cały kod tej książki, ponieważ umożliwiają ścisłą współpracę między użytkownikiem a biblioteką. Użytkownik biblioteki dosłownie steruje tym, w jaki sposób (w granicach wytyczonych przez bibliotekę) jest generowany kod. Biblioteka komponentów generycznych umożliwia tworzenie struktury przez łączenie typów i funkcjonalności zaprojektowanych przez użytkownika z komponentami biblioteki. Ze względu na statyczny charakter stosowanych technik błędy powstałe przy łączeniu i mieszaniu komponentów są zazwyczaj wykrywane podczas kompilacji.

W książce tej chodzi przede wszystkim o tworzenie komponentów generycznych – prefabrykowanych kawałków programu, których głównymi cechami są: elastyczność, wszechstronność i wygoda użycia. Komponenty generyczne nie stanowią litego zrębu (ang. *framework*). W istocie spełniają funkcję uzupełniającą. O ile zrąb definiuje powiązane ze so-

bą klasy, narzucające konkretny model obiektowy, o tyle komponenty generyczne są lekkimi artefaktami projektowymi, które są od siebie niezależne, ale mogą być dowolnie dopasowywane i łączone. Mogą też być bardzo pomocne przy *implemantacji* zrębów.

Adresaci książki

Książka ta jest przeznaczona dla dwóch grup odbiorców. Pierwsza to programiści doświadczeni w posługiwaniu się C++, którzy chcieliby opanować większość współczesnych technik tworzenia kodu bibliotecznego. Znajdą tu oni omówienie nowych, potężnych idiomów języka C++ o zaskakujących możliwościach, które jeszcze do niedawna uważano za nieosiągalne. Stanowią one nieocenioną pomoc przy pisaniu bibliotek wysokiego poziomu. Na przeczytaniu tej książki skorzystają też z pewnością programiści średnio zaawansowani w C++, którzy chcieliby zrobić krok do przodu, zwłaszcza jeśli wykażą się pewną dozą uporu. Przedstawiony kod w C++ jest miejscami naprawdę trudny, ale jest wyczerpująco wyjaśniony. Druga grupa to zapracowani programiści, którzy muszą szybko uporać się z robotą, a nie mają czasu na uczenie się nowych technik. Oni zapewne przekartkują najbardziej zawiłe szczegóły implementacyjne, a skupią się na wykorzystaniu przedstawionej biblioteki. Każdy rozdział rozpoczyna się krótkim wyjaśnieniem, a kończy zwartym podsumowaniem. Układ ten jest bardzo pomocny w szybkim przyswojeniu działania opisanych składników biblioteki. Składniki można poznawać pojedynczo. Każdy z osobna stanowi bardzo silne, a mimo to bezpieczne i przyjemne narzędzie.

Będzie Ci potrzebne solidne doświadczenie w pracy z C++, a przede wszystkim chęć do uczenia się czegoś nowego. Przyda się też znajomość szablonów (ang. *templates*) oraz standardowej biblioteki szablonów (ang. *Standard Template Library*, STL).

Znajomość wzorców projektowych (Gamma i in. 2005) jest zalecana, ale niekonieczna. Stosowane tutaj wzorce i idiomy są szczegółowo opisane. Nie jest to jednak książka o wzorcach; dlatego nie są one w niej potraktowane całościowo. Wzorce są pokazane z punktu widzenia twórcy kodu bibliotecznego, a zatem nawet Czytelnicy zainteresowani głównie wzorcami mogą odkryć w tym zawężonym podejściu coś nowego.

Loki

W książce jest przedstawiona biblioteka C++ o nazwie Loki. Loki to w mitologii nordyckiej bóg sprytu i podstępu. Autor ma nadzieję, że oryginalność i elastyczność biblioteki będą przypominać Czytelnikowi owego figlarnego boga. Wszystkie elementy biblioteki są umiejscowione w przestrzeni nazw `Loki`. Przestrzeń nazw została pominięta w przykładach, gdyż niepotrzebnie zwiększyłoby to wcięcia i rozmiar kodu. Biblioteka Loki jest dostępna za darmo pod adresem: http//www.awl.com/cseng/titles/0-201-70431-5.

Biblioteka jest napisana wyłącznie w standardowym C++, z wyjątkiem fragmentu dotyczącego wątków. Mimo to wiele współczesnych kompilatorów nie radzi sobie z jej fragmentami. Ja sam implementowałem i testowałem Loki za pomocą kompilatorów Metrowerks Code Warrior Pro 6. oraz Comeau C++ 4.2.38 w systemie Windows. Prawdopodobnie kompilator KAI C++[1] też nie miałby problemów. W miarę powstawania nowych, do-

[1] KAI jest obecnie częścią firmy Intel (też dostawcy kompilatorów C++). (Przyp. tłum.)

skonalszych wersji kompilatorów C++, będzie można coraz pełniej wykorzystywać bibliotekę Loki[1].

Kod Loki oraz przykłady w tekście są w stylu zaproponowanym przez Herba Suttera. Jestem pewien, że szybko zrozumiesz, o co chodzi. W skrócie można powiedzieć tak:

- klasy, funkcje i typy wyliczeniowe wyglądają `TakJakTo`;
- zmienne i wartości typów wyliczeniowych wyglądają `takJakTo`;
- składowe danych wyglądają `takJakTo_`;
- argumenty wzorców są deklarowane za pomocą `class`, jeśli można na nie podstawiać wyłącznie typy zdefiniowane przez użytkownika, a za pomocą `typename`, jeśli dopuszcza się także typy podstawowe[2].

Układ książki

Książka składa się z dwóch zasadniczych części dotyczących – odpowiednio – technik C++ oraz składników biblioteki. Część I (rozdz. 1–4) przybliża techniki języka C++ stosowane w progrmowaniu uogólnionym, ze szczególnym uwzględnieniem tworzenia generycznych składników bibliotek. Jest w niej przedstawionych mnóstwo chwytów charakterystycznych dla C++: projektowanie szablonów opartych na wytycznych, częściowa specjalizacja szablonów, listy typów, klasy lokalne itd. Można przeczytać te rozdziały po kolei, a potem wracać do nich w miarę potrzeby.

Część II jest oparta na części I. Zawiera implementacje generycznych składników biblioteki, przy czym nie są to wydumane przykłady. Zaprezentowane składniki to kod o jakości przemysłowej, używany w prawdziwych programach. Zagadnienia powracające wielokroć w codziennej pracy programisty C++, dotyczące np. inteligentnych wskaźników (ang. *smart pointers*), fabryk obiektów czy funktorów, są dokładnie omówione i zaimplementowane w sposób generyczny. W przedstawionych implementacjach są uwzględnione podstawowe wymagania i rozwiązane zasadnicze problemy. Zamiast wyjaśnień, co robi kawałek kodu znajdziesz tu opisy problemu i podjętych decyzji projektowych oraz kolejne fazy ich implementacji.

Zawartość rozdziałów jest następująca:

1 – wytyczne (idiom języka C++, pomagający tworzyć systemy o elastycznej strukturze);
2 – podstawowe techniki programowania uogólnionego w C++;
3 – lista typów, niezwykle silna struktura do manipulowania typami;
4 – ważne narzędzie dodatkowe: przydzielacz małych obiektów;
5 – pojęcie funktorów uogólnionych, użyteczne w programach korzystających ze wzorca projektowego Polecenie;
6 – implementacja wzorca Singleton;
7 – implementacja inteligentnych wskaźników;
8 – fabryki obiektów;

[1] W czerwcu 2004 r. biblioteka Loki działała w pełni na kompilatorach GCC 2.95.3, Microsoft Visual C++ 7.0, Borland C++ Builder 6.0 oraz częściowo na kompilatorach Code Warrior 6.0 i Microsoft Visual 6.0. (Przyp. tłum.)

[2] Obecnie (2004 r.) autor zaleca odejście od tej praktyki i jednorodne korzystanie ze słowa kluczowego `class`. (Przyp. tłum.)

9 – wzorzec Fabryka Abstrakcyjna oraz jego implementacje;
10 – kilka wariantów generycznej implementacji wzorca Odwiedzający;
11 – kilka implementacji mechanizmu wielometod (ang. *multimethods*).

Zagadnienia projektowania dotyczą wielu sytuacji, z którymi w codziennej pracy styka się programista tworzący kod w C++. Osobiście uważam fabryki obiektów (rozdz. 8) za podwalinę każdego rozsądnego projektu, w którym korzysta się z polimorfizmu. Również inteligentne wskaźniki (rozdz. 7) są istotnym składnikiem wielu dużych i małych programów w C++. Funktory uogólnione (rozdz. 5) mają ogromny zakres zastosowań. Istotnie upraszczają wiele skomplikowanych problemów projektowych. Inne, bardziej wyspecjalizowane składniki biblioteki, takie jak Odwiedzający (rozdz. 10) czy wielometody, (rozdz. 11) mają ważne zastosowania niszowe, a praktycznie rozszerzają język o nowe mechanizmy.