17. Tworzenie własnych generycznych kontrolek GUI

Gdy poznaję nowe rzeczy i zdobywam nową wiedzę, największą satysfakcję sprawia mi świadomość, że są już one wystarczające, abym mógł wyjść poza narzucone ramy i wykorzystać je do kreowania własnych elementów programistycznych, zgodnych z moją wizją i działających tak, jak sobie tego życzę. Taki moment pojawił się również wówczas, gdy po raz pierwszy skompilowałem program wxWidgets z działającą kontrolką generyczną, którą w całości opracowałem i wykonałem samodzielnie.

W rozdziale zapoznam Cię z zasadami pisania własnych kontrolek generycznych, a także pokażę Ci kilka praktycznych przykładów, dzięki którym przekonasz się, że jest to zadanie niezbyt trudne. Mam cichą nadzieję, że wkrótce zaowocuje to wysypem ciekawych kontrolek GUI, które wyjdą spod Twojej ręki, co sprawi Ci ogromną satysfakcję i jeszcze bardziej zbliży do tego wspaniałego narzędzia, jakim jest wxWidgets.

17.1. Główne zasady tworzenia własnych kontrolek generycznych

Kontrolki generyczne pisze się zwykle na podstawie jednej z klas bazowych, czyli ***wxControl***, lub stojącej na nieco niższym poziomie klasy ***wxWindow***. Obie zawierają kompletne i dostosowane do różnych platform mechanizmy umożliwiające szybkie tworzenie kontrolek pracujących w programach pisanych z wxWidgets. W zasadzie można powiedzieć, że po stronie programisty pozostaje już tylko realizacja samej kontrolki, co należy rozumieć jako opracowanie jej wyglądu, interakcji, zdarzeń oraz ewentualnie klas towarzyszących, umożliwiających przetwarzanie jakichś specyficznych danych.

Proces tworzenia kontrolek generycznych, którym się teraz zajmiemy, obejmuje właśnie te czynności. Z technicznego i organizacyjnego punktu widzenia można je zebrać w pewien uogólniony i uniwersalny plan, który obejmuje siedem łatwych do zapamiętania kroków:

1. Opracowanie klas narzędziowych i struktur danych, które mają być wewnętrznie wykorzystywane przez kontrolkę.
2. Opracowanie trzonu klasy kontrolki obejmującego definicję klasy wraz z jej składowymi zmiennymi oraz metodami umożliwiającymi tworzenie kontrolki zgodnie z zasadami przyjętymi w wxWidgets (konstruktory, metoda *Create()*), a także zaplanowanie zdarzeń, jakie będzie emitować kontrolka.
3. Opracowanie klasy zdarzenia kontrolki wraz z uwzględnieniem ewentualnych, zależnych od charakteru kontrolki wartości dodatkowych, jakie będą przenoszone w pętli zdarzeń wraz z obiektami zdarzeń.
4. Opracowanie metody inicjującej wstępne wartości obiektu kontrolki.
5. Opracowanie metod odpowiedzialnych za inicjowanie rysowania kontrolki w przypadku wystąpienia zdarzenia ***wxEVT_PAINT***, a także opcjonalnie wydarzenia skalowania ***wxEVT_SIZE***.
6. Opracowanie metody rysującej kontrolkę z użyciem właściwego kontekstu.
7. Opracowanie metod obsługujących wewnętrzne zdarzenia kontrolki i inicjujących propagację właściwych zdarzeń kontrolki do pętli zdarzeń.

Każdy z tych kroków wymaga odrębnego podejścia i dokładnego planowania, ponieważ podczas realizacji rozbudowanych kontrolek wszelkie zmiany koncepcji mogą się wiązać z dość kłopotliwą koniecznością refaktoryzacji znacznej ilości kodu.

W przypadku punktu pierwszego należy przewidzieć wszystkie typowe i nietypowe zestawy danych, które mogą być niezbędne do poprawnej pracy kontrolki. Warto wówczas zastanowić się nad tym, czy wyposażyć klasę w jakieś wewnętrzne kontenery lub mechanizmy pozyskiwania danych ze źródeł zewnętrznych, jak choćby baza danych SQL. Jest to też dobry moment na zastanowienie się nad tym, czy nasza nowa klasa ma być znana w aplikacji przez *wxRTTI*.

Trzon klasy kontrolki, o którym mowa w punkcie drugim, w większości przypadków będzie taki sam. Zresztą chcąc być wiernym zasadom panującym w świecie wxWidgets, powinieneś zawsze dbać o opracowanie konstruktora domyślnego, opracowanie konstruktora zgodnego z ogólnie przyjętym schematem lub co najmniej z nim zbieżnego w zakresie umożliwiającym łatwe rozpoznanie jego elementów i bezproblemowe stosowanie przez innych programistów (jeśli oczywiście zechcesz z efektem swoich prac podzielić się z szerszym gronem), a także opracowanie metody *Create()*, która jest nieodłącznym elementem wszystkich klas kontrolek wxWidgets. Z tym etapem prac wiąże się również opracowanie wewnętrznej metody inicjującej startowe wartości danej kontrolki.

Gdy obraz kontrolki jest już zaplanowany, a po wszelkich jego korektach, zmianach i rozwianiu wszelkich wątpliwości wiemy, jakiego rodzaju dane będzie ona wewnętrznie przetwarzać i czy dane te będziemy chcieli przekazywać dalej, dobrze jest zastanowić się nad klasą zdarzenia naszej kontrolki. Opracowanie klasy zdarzenia to proces dość skomplikowany i obwarowany pewnymi wymaganiami, który oprócz opisania samej klasy wymaga również zastosowania pewnych dodatkowych makr i konstrukcji, które muszą wystąpić w odpowiednim porządku. Dalej opowiem Ci o tym, jak przejść przez ten etap bezboleśnie i bezproblemowo.

Na koniec pozostaje wykonanie rzeczy, które w istocie są najbardziej pracochłonne i podczas pisania własnych kontrolek mogą wywoływać wiele frustracji. Chodzi oczywiście o właściwe i poprawne wykonanie metod rysujących kontrolkę oraz metod emitujących jej zdarzenia.

17.1.1. Program przykładowy

Proces tworzenia kontrolek generycznych, który ogólnie i w formie jednej z możliwych koncepcji przedstawiłem wcześniej, przekujemy w praktykę, pisząc razem przykładowy program, który będzie obrazować tematyczne zagadnienia wraz z ich stopniowym uszczegóławianiem i komplikowaniem wykonywanych rzeczy.

Na początek pokażę Ci, w jaki sposób wykonać panel, który w tle będzie mieć grafikę skalowaną wraz ze zmianą wielkości okna programu. Co prawda wxWidgets ma już klasę pozwalającą wykonać podobne rzeczy, jednak są one ograniczone jedynie do wyświetlania grafiki tła w trybie kafelkowym.

Dalej przedstawię Ci sposób wykonania prostego przycisku graficznego o trzech stanach, którego działanie będzie podobne do działania przycisków znanych z witryn internetowych. Będzie on nie tylko obsługiwać stany przycisku: zwykły, zawieszony (*hover*) oraz aktywowany, ale także będzie pozwalał na zastosowanie dowolnej grafiki, w tym obsługującej przezroczystość. Wraz z realizacją przycisku poznasz podstawowe zasady rysowania kontrolek oraz propagacji elementarnych zdarzeń kontrolki.

Kolejną kontrolką, jaką razem wykonamy, będzie własny pasek postępu. Na jego przykładzie zobaczysz nieco bardziej zaawansowane techniki rysowania kontrolki, a także sposób, w jaki można sprzęgnąć kontrolkę z działaniami prowadzącymi do zmian wartości, jakie ona reprezentuje i obrazuje.

Na koniec wykonamy prosty kalendarz, który będzie łączyć wszystkie te zagadnienia, a także pokażę Ci, jak wykorzystać klasy dodatkowe do zarządzania wewnętrznymi danymi kontrolki.

Kompletny kod programu możesz znaleźć w materiałach dodatkowych do książki, w katalogu *Rozdział 17/Własne kontrolki*.

Podczas lektury rozdziału skorzystaj z kodu programu przykładowego, aby prowadzić własne testy i eksperymenty. Możesz go również użyć jako bazy do wykonania własnych modyfikacji i stworzenia nowych kontrolek.

Rysunek 17.1. Okno przykładowego programu obrazującego techniki realizacji własnych kontrolek generycznych. U góry po lewej jest widoczny graficzny przycisk o trzech stanach, po prawej zaś własny graficzny pasek postępu. Znaczny fragment dolnej części okna jest zajęty przez własną kontrolkę kalendarza. W tle widoczny jest panel ze skalowalną grafiką. W interfejsie tej aplikacji nie występują żadne standardowe kontrolki wxWidgets. Rysunek przedstawia okno programu uruchomionego w systemie operacyjnym Mint Linux

Jeszcze lepszym rozwiązaniem będzie samodzielne wykonanie przykładowych kontrolek. W takiej sytuacji utwórz domyślny projekt aplikacji wxWidgets. Gdy konieczne będzie dodanie do niego jakichś specyficznych plików nagłówkowych lub bibliotek, niezwłocznie Cię o tym poinformuję.

Wiem, że pisanie własnych kontrolek jest zajęciem, do którego aż się palisz, dlatego nie będę Cię dłużej zamęczał. Zajmijmy się praktyką...

17.2. Skalowany panel z obrazkowym tłem

Zadanie, którym się teraz zajmiemy, będzie polegać na wykonaniu kontenera dla kontrolek GUI, który będzie miał tło w postaci zdefiniowanej grafiki, skalowalnej wraz ze zmianą wielkości panelu.

Zanim zaczniemy, winien Ci jestem informację, że biblioteka wxWidgets ma już podobne rozwiązanie, jednak pozwala ono jedynie na wyświetlanie panelu z tłem graficznym wyświetlanym w formie kafelków. Jest nim klasa szablonowa ***wxCustomBackgroundWindow*** dostarczana wraz z plikiem nagłówkowym *wx/custombgwin.h*. Korzystanie z niej ogranicza się do wyprowadzenia z niej własnej klasy i zastosowania jej jako kontenera (tła) dla innych kontrolek:

```
class MyPanel : public wxCustomBackgroundWindow<wxPanel>
{
   public:
      MyPanel(wxWindow* parent, wxBitmap bitmap)
      {
         Create(parent, wxID_ANY);
         SetBackgroundBitmap(bitmap);
      }
};
```

Dlaczego jednak nie pójść nieco dalej? Przecież panel, jakim się za chwilę zajmiemy, będziesz mógł rozbudować o podobne opcje, a także wzbogacić o kolejne tryby wyświetlania grafiki.