Java Swing

Tomasz Mularczyk
Luty 2005

Swing - Wprowadzenie

- Pierwotnym IU dla Javy był AWT (Abstract Windowing Toolkit)
- IU Swing dodano dopiero w wersji Javy 1.2 (przełom 1997/98)
- Dlaczego skonstruowano kolejny sposób tworzenia interfejsów ?

Swing - Wprowadzenie

- Główny powód: niezadowolenie osób tworzących w Javie z istniejącego rozwiązania
- Wygląd interfejsów stworzonych przy pomocy AWT zależało od systemu operacyjnego
- AWT oferowało słabe wsparcie dla rysowania po ekranie oraz nie zapewniało kompatybilności operacji Drag and Drop pod kontrolą różnych SO

Swing a AWT

AWT:

- Wygląd kontrolek specyficzny dla każdego SO

Swing:

- Wygląd kontrolek niezależny od SO
- Schematy wyglądu (look and feel)
- Bogate wsparcie dla rysowania 2d

Swing a AWT

AWT:

- Korzysta z wywołań funkcji systemowych przy tworzeniu i zarządzaniu wyglądem aplikacji
- Generalnie szybki, choć szybkość działania zależy od systemu operacyjnego

Swing:

- W całości napisany w Javie
- Nieco wolniejszy, ale mniejsze różnice w prędkości pomiędzy platformami

Swing a AWT

AWT:

- Z założenia ciężki typ kontrolek (lekkie dodano później w wersji 1.1)
- Wspierany od początku istnienia Javy

Swing:

- Lekki typ kontrolek
- Wymaga wirtualnej maszyny w wersji minimum 1.2 (dziś to już nie martwi)

Swing a AWT

AWT:

- Stosunkowo trudne tworzenie narzędzi do wizualizacji „w locie” (ciężkie kontrolki)

Swing:

- Łatwe tworzenie narzędzi do podglądu (lekkie kontrolki) oraz tworzenia IU – wspierane m.in. przez JBuildera, wtyczki do Eclipse

Swing - Założenia projektu

1. Rozszerzyć kontrolki AWT

- Wszystkie kontrolki AWT mają odpowiednik w Swingu
- Funkcjonalność wielu kontrolek rozszerzono (np. wszystkie kontrolki wyboru akceptują teraz zarówno tekst jak i obrazki jako element wyglądu)
- Dodano szereg nowych kontrolek (tabele, drzewa, paski stanu, chmurki podpowiedzi...)

Swing - Założenia projektu

2. Dodać sensowną obsługę 2D

- java.awt.Graphics był chyba najbardziej znienawidzonym pakietem
- Dodano możliwość rysowania niezależnie od rozdzielczości i urządzenia
- Wsparcie dla przezroczystości, ulepszenie wyświetlania tekstów
- Anti-aliasing i krzywe Beziera w standardzie

Swing - Założenia projektu

3. Wprowadzić schematy wyglądu

- Część użytkowników oczekuje tego samego wyglądu od aplikacji działającej w różnych środowiskach
- Inni wolą, aby program zachowywał się tak, jak resztą programów działających na jego platformie (na to pozwalał AWT)
- Swing ma zadowolić obie te grupy, a nawet pozwolić użytkownikowi wybrać wygląd programu

Swing - Założenia projektu

4. Obsłużyć Przeciąganie i Upuszczanie

- Dodano pomost między mechanizmami Javy a tymi wbudowanymi w konkretne systemy operacyjne, z których korzystają typowe aplikacje
- Dostrzeżono w ten sposób, że nie całe oprogramowanie jest pisane w Javie :)

Swing - Założenia projektu

5. Porządnie obsłużyć wyświetlanie tekstu

- JTextComponent potrafi wyświetlać tekst opisany przy pomocy języka podobnego do SGMLa.
- Obsługa wielu fontów, podkreśleń wielu linii, osadzonych obrazków
- Możliwe również wyświetlanie przeskalowanych, obróconych czy zacienionych tekstów

Swing - Założenia projektu

6. Ułatwić przeniesienie kodu z AWT

- Swing jako przyszłość, AWT tylko dla kompatybilności – stąd potrzeba migracji
- Większość kontrolek z AWT ma w Swingu odpowiednika o nazwie J<nazwa_AWT>, np. dla klasy Button odpowiednikiem jest JButton
- Swing implementuje wszystkie metody z AWT, więc migracja to tylko zmiana nazw

Swing - Założenia projektu

7. Zaprojektować w oparciu o MVC (Model, View, Controller)

Architektura MVC – założenia

Program kiedyś:

Wejście -> Przetwarzanie -> Wyjście

Input->Processing ->Output

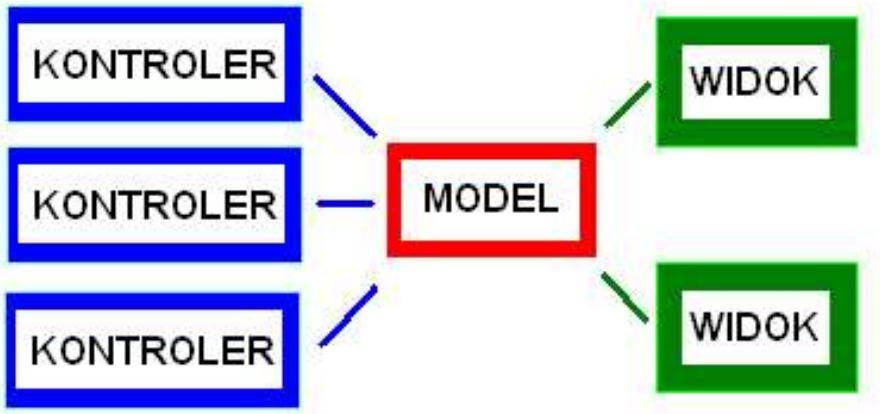
Program dziś:

Kontroler -> Model -> Widok

Controller->Model ->View

Architektura MVC - opis

- Podejście to pojawiło się w SmallTalk 80
- Kontrolery, model i widoki traktowane jako oddzielne byty
- Zmiany modelu natychmiast odzwierciedlane w widokach



Architektura MVC - opis

- Wszystkie komponenty w systemie muszą dać się określić jako kontroler, model, widok bądź część któregoś z nich
- Połączenie model <-> widok jest realizowane dynamicznie – w czasie działania programu, a nie w czasie kompilacji
- Jeżeli zaprojektujemy w MVC każdy komponent, to łatwe będzie ich łączenie w spójny projekt

Architektura MVC - opis

- Usuwanie bądź przebudowa komponentu nie pociąga za sobą potrzeby ingerencji w cały system
- Model nie wie, jakie są jego kontrolery, nie zna także korzystających z niego widoków – sam system zajmuje się ich spójnym połączeniem
- Kontrolery i Widoki znają skojarzony z nimi model

Architektura MVC - przykład

Załóżmy, że projektujemy wieloosobową, trójwymiarową grę, w której gracze dowodzą okrętami:

- Model – cały świat 3d wraz opisami statków graczy, lecących pocisków, własności morza
- Kontrolery – akcje podejmowane przez różnych graczy, np. rozkaz wystrzelenia pocisku
- Widoki – np. widok z mostku, rzut pionowy (mapa akwenu), czy patrzenie przez lornetkę

Architektura MVC w Javie

Java oferuje wsparcie dla MVC poprzez dwie klasy:

- Obserwator (Observer) – obiekt, który chce być powiadamiany o zmianach w innym obiekcie
- Obserwowany (Observable) – każdy obiekt, którego zmiana stanu może zainteresować inny obiekt

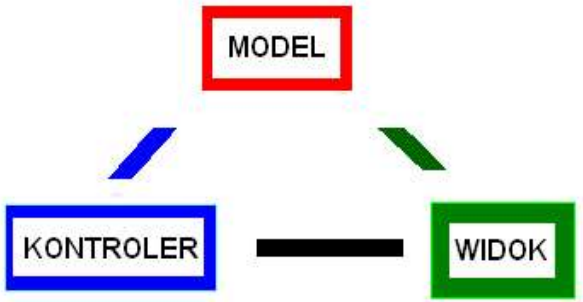
Architektura MVC w Javie

- MODEL jest klasy „Obserwowany”
- WIDOK jest klasy „Obserwator”
- WIDOK i KONTROLER posiadają referencję do MODELU i potrafią się z nim komunikować poprzez wywoływanie jego metod

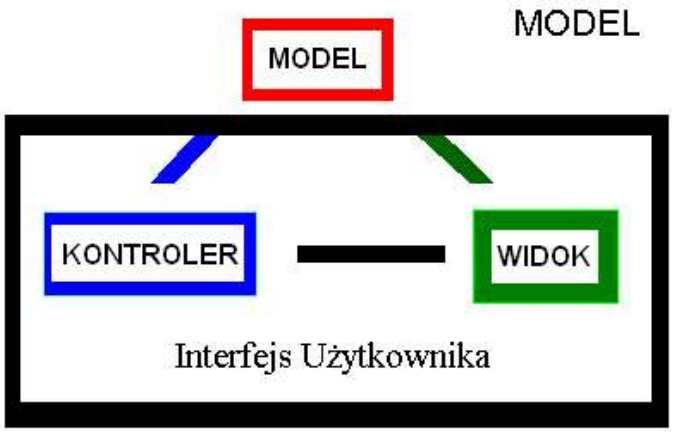
Architektura MVC w Swingu

Swing a architektura MVC:

Ogólny model MVC



MVC w Swingu



Architektura MVC w Swingu - przykład

Zajmijmy się prostym Przyciskiem (JButton)

- Model przechowuje stan, pozwala na jego modyfikację i odczytywanie. Dodatkowo pozwala dodawać/usuwać słuchaczy(Listeners).

- Widok i kontroler zajmują się wyświetlaniem stanu i przycisku na ekranie. Tutaj także pamiętamy, w którym miejscu na ekranie znajduje się przycisk. Ta część zajmuje się również obsługą zdarzeń AWT.

Słuchacze (Listeners)

- W modelu, z którego korzysta Swing, akcje podejmowane są w odpowiedzi na zajście konkretnych zdarzeń.

- Aby powiązać zdarzenie z określoną akcją, każdej kontrolce musimy przyporządkować specjalną klasę, która będzie reagowała na zajście tego zdarzenia. Tę klasę nazywamy słuchaczem.

Słuchacze jako klasy wewnętrzne

- Koszt zaprojektowania dla każdego elementu interfejsu oddzielnej klasy obsługującą jego zdarzenia jest bardzo duży. Nie jest to konieczne!

- Idealnie sprawdzają się tutaj klasy wewnętrzne Javy, gdyż od razu uzyskują dostęp do zmiennych i metod kontrolki.

Anonimowi słuchacze

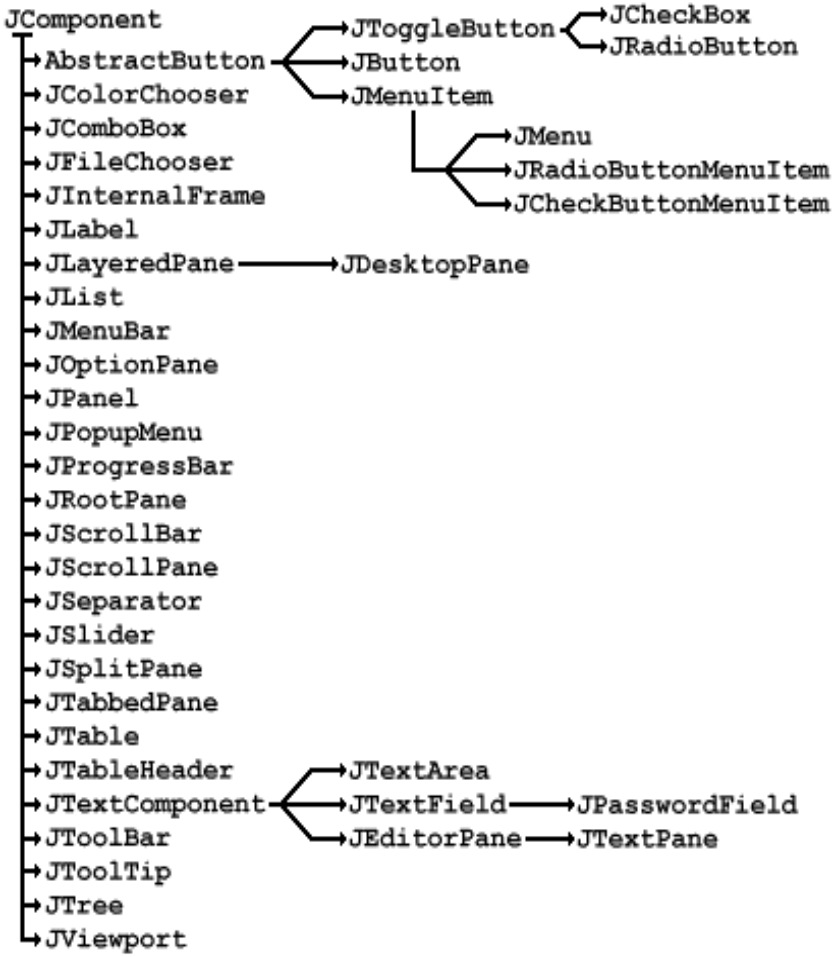
- Java raczy nas kolejnym udogodnieniem w takim modelu obsłudze zdarzeń – umożliwia tworzenie klas anonimowych.

- Stąd kod dodający przyciskowi obsługę zdarzenia może wyglądać tak:

```
Button.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent evt) {
m_iCount++;
}});
```

Zarys klas i pakietów

Kilknaście wybranych klas Swing (większość pochodzi z pakietu javax.swing)



Zarys klas i pakietów

javax.swing	- podstawowa funkcjonalność i kontrolki
javax.swing.border	- style obramowań
javax.swing.colorchooser	- menu wyboru kolorów
javax.swing.event	- zdarzenia i klasy słuchaczy
javax.swing.filechooser	- menu wyboru plików
javax.swing.plaf	- schematy wyglądu
javax.swing.plaf.basic	- bazowa klasa z wyglądem interfejsu
javax.swing.plaf.metal	
javax.swing.plaf.multi	
javax.swing.table	- tabele
javax.swing.text	
javax.swing.text.html	
javax.swing.text.html.parser	
javax.swing.text.rtf	
javax.swing.tree	- pozwala przechowywać i wizualizować dane
javax.swing.undo	- pomaga obsługiwać operacje Cofnij/Powtórz

Wtyczki interfejsu użytkownika

Schematy wyglądu dostarczane wraz ze Swingiem:

- GTKLookAndFeel (Linuks z GTK)
- MetalLookAndFeel (typowy dla Javy)
- MotifLookAndFeel
- WindowsLookAndFeel (typowy dla Windows)

Niestety, nie każdy styl jest już obsługiwany na każdej platformie.

Wtyczki interfejsu użytkownika

MetalLookAndFeel

MotifLookAndFeel

WindowsLookAndFeel

DefaultLookAndFeel

Przykłady kodu i wyglądu okien

- Wygląd pokazanych okien uzyskaliśmy poprzez poinformowanie menedżera wyglądu o tym, który chcielibyśmy uzyskać:

```
UIManager.setLookAndFeel(
"javax.swing.plaf.metal.MetalLookAndFeel"  );
```

- Istnieje także możliwość pozostawienia wyboru, który wygląd wybrać, użytkownikowi (program wykorzysta typowe ustawienia pobrane z lokalnego pliku <JAVA>/lib/swing.properties)

Przykłady kodu i wyglądu okien

- Stworzenie samego okienka wraz z napisem oraz przyciskiem zajmuje tylko chwilę:

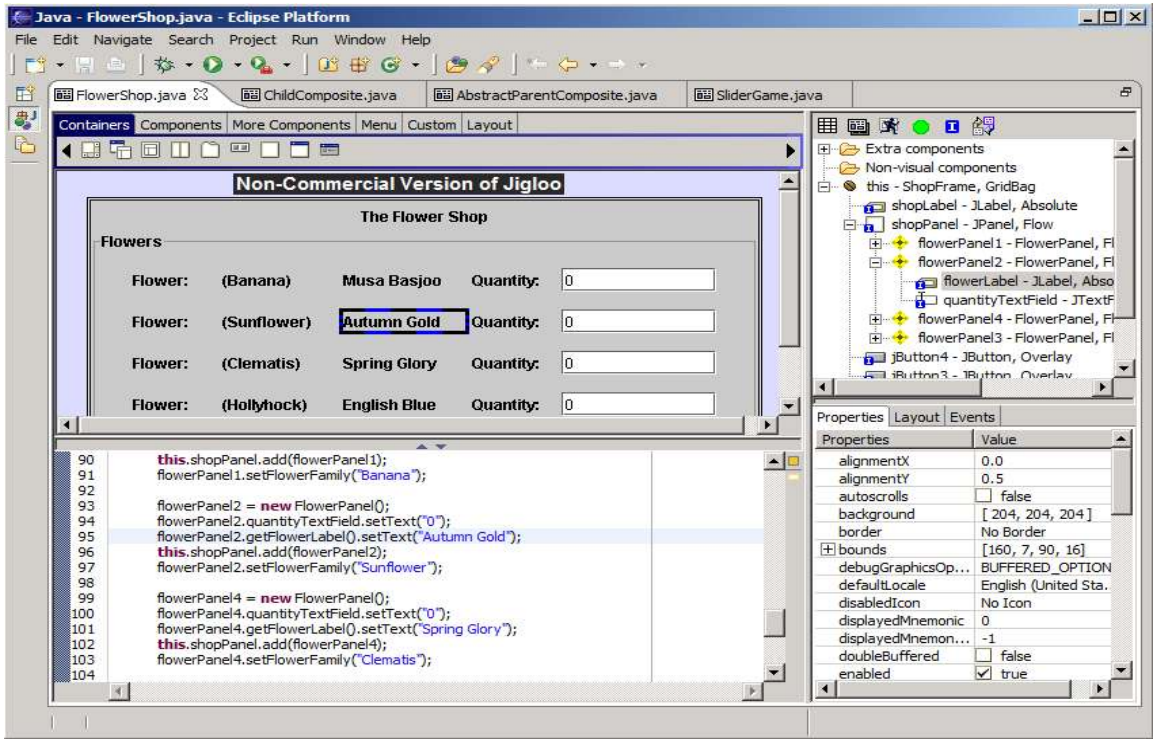
```
MainFrame = new JFrame("Roznice w wyglądzie są duże");
MainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JLabel label = new JLabel("JAKIS TEKST");

JButton but = new JButton();
but.setText("OK");

JPanel panel = new JPanel();
panel.setBorder(BorderFactory.createEmptyBorder());
panel.add(but);
panel.add(label);
MainFrame.getContentPane().add(panel);
MainFrame.pack();
MainFrame.setSize(400,100);
```

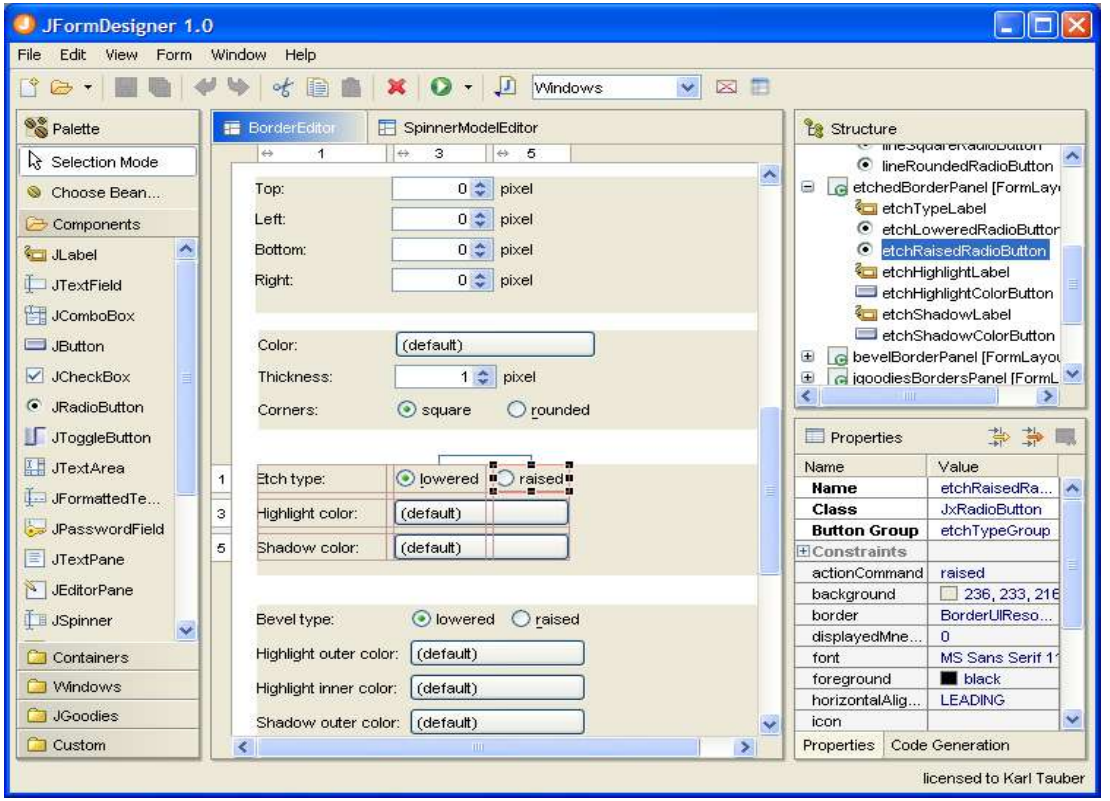
Generatory kodu

- Nie musimy wszystkiego pisać sami



Jigloo

Generatory kodu



JFormDesigner

Generatory kodu

Główny problem – generatory są płatne

- JFormDesigner kosztuje 129 $
- Jigloo kosztuje 75 $, ale jest darmowy do domowego użytku

Podsumowanie

- Swing rozszerza AWT i przerasta go w wielu względach
- Wygląd aplikacji napisanych przy użyciu Swinga jest taki, jakiego sobie zażyczymy na każdej platformie
- Swing został zaprojektowany z uwzględnieniem architektury Model-View-Controller
- Dzięki zastosowaniu anonimowych słuchaczy pisanie kodu obsługującego zdarzenia jest szybkie

Gdzie dalej szukać ?

Porównanie AWT i Swing:
http://bdn.borland.com/article/0,1410,26970,00.html
http://mindprod.com/jgloss/swing.html

Dostępne schematy wyglądu:
http://java.sun.com/docs/books/tutorial/uiswing/misc/plaf.html

Model-View-Controller:
http://ootips.org/mvc-pattern.html

Oczywiście polecam dokumentację ze strony Sun-a:
http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/package-summary.html

Gdzie dalej szukać ?

Jigloo
http://www.cloudgarden.com/jigloo/index.html

JFormDesigner
http://www.jformdesigner.com/