

Jacek Matulewski

<http://www.fizyka.umk.pl/~jacek/>

Java w NetBeans

część 1: Szybki start

Skrypt dla słuchaczy studium podyplomowego SPIN
(WFAiIS UMK)

Toruń, 10 listopada 2007

Najnowsza wersja skryptu i źródła jest dostępna pod adresem
http://www.fizyka.umk.pl/~jacek/dydaktyka/java_spin/

Spis treści

Spis treści	2
Java, JDK, NetBeans.....	3
Projekt „Hello World!”	3
Tworzenie i zarządzanie projektem.....	3
Przewodnik po oknach i zakładkach IDE.....	4
Aplikacja konsolowa Hello World!.....	5
Uruchamianie aplikacji spoza IDE.....	5
Aplikacja GUI (AWT)	6
Projektowanie interfejsu AWT	8
Tworzenie okna w metodzie hello.Main.main	10
Aplet AWT.....	10
Osadzanie apletu w dokumencie HTML.....	11
Zmiana nazwy pliku/klasz (refactoring)	13
Aplikacja GUI (Swing) i projektowanie interfejsu Swing	13
Projekt „Kolory”	14
Idea.....	14
Tworzenie projektu	14
Projektowanie interfejsu Swing	15
Zdarzenia.....	15
HSB.....	17
Debugowanie kodu w NetBeans.....	20

Java, JDK, NetBeans

NetBeans to środowisko programistyczne (IDE, ang. *Integrated Development Environment*) dołączane do JDK (ang. *Java Development Kit*). Zawiera edytor kodu z mechanizmem podpowiadania kodu, debugger (kontrolowane uruchamianie aplikacji i apletów) oraz narzędzia projektowania wizualnego (RAD). NetBeans można ściągnąć ze strony Sun wraz z JDK: <http://java.sun.com/javase/downloads/netbeans.html>.

JDK to wirtualna maszyna Javy (ang. JVM = *Java Virtual Machine*) oraz kompilator, cały pakiet narzędzi, bibliotek i dokumentacji dostarczanych przez twórców Javy — firmę Sun.

Kod pośredni

Dokumentacja do klas JDK: <http://java.sun.com/javase/6/docs/>, <http://java.sun.com/javase/6/docs/api/>

Alternatywne rozwiązania: Eclipse, JBuilder (Borland)

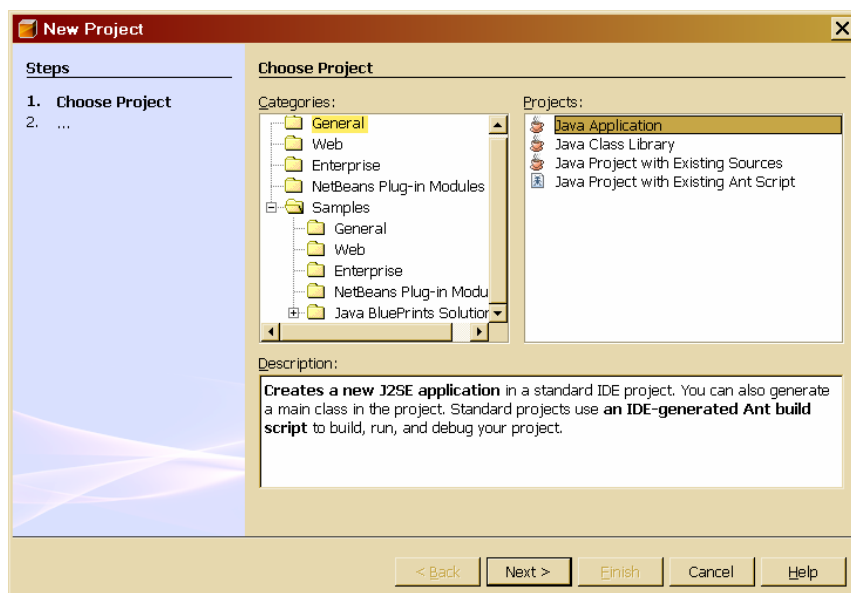
Projekt „Hello World!”

Tworzenie i zarządzanie projektem

Jak w większości programów pracę z NetBeans zaczynamy od menu *File*. Tu zgromadzona jest większość dostępnych kreatorów, w tym kreatory tworzące projekty. Pierwszą czynnością, jaką powinniśmy zrobić, aby rozpocząć projektowanie apletu lub aplikacji, jest stworzenie projektu. Projekt umożliwia sprawne zarządzanie plikami zapamiętując ich położenie, niektóre ustawienia dotyczące kompilacji, debugowania, dodatkowe elementy związane z tworzeniem dokumentacji lub archiwów JAR itp.

Aby stworzyć nowy projekt o nazwie *hello*:

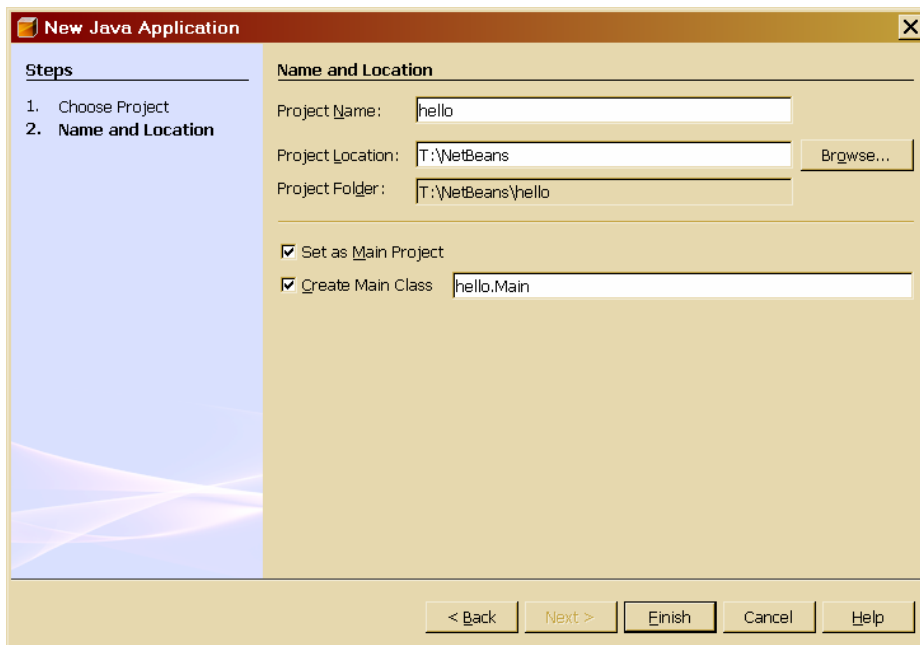
1. Z menu *File* wybieramy pozycję *New Project...* Otwieramy okno *New Project* (rysunek 1)
 - a) w nowym oknie wybieramy pozycję *General* w drzewie z lewej strony okna,
 - b) zaznaczamy ikonę *Java Application*,
 - c) klikamy przycisk *Next >*.



Rysunek 1. Po wybraniu pozycji *New Project...* z menu *File* zobaczymy galerię możliwości NetBeans

2. Pojawia się drugie okno kreatora projektu (rysunek 2), w którym określamy:

- a) nazwę projektu (pole *Project Name*) będącą jednocześnie nazwą podstawowego pakietu¹: **hello**,
- b) lokalizację projektu (katalog na lokalnym dysku). Powstanie podkatalog o nazwie identycznej, jak nazwa projektu np. *T:\NetBeans\hello*.
- c) należy zaznaczyć opcję *Set as Main Project* jeżeli NetBeans obsługuje aktualnie inne projekty,
- d) należy również pozostawić zaznaczoną opcję *Create Main Class*, aby kreator utworzył za nas klasę **Main**. Wywołanie jej metody **main** pozwala na uruchomienie aplikacji (ang. *entry point*). Nazwa klasy **Main** nie jest istotny ze względu na możliwość uruchamiania programu (można ją zmienić), ale nazwa metody **main** musi być właśnie taka.
- e) Klikamy przycisk *Finish*.



Rysunek 2. Tworzenie projektu aplikacji

<< koniec ćwiczenia >>

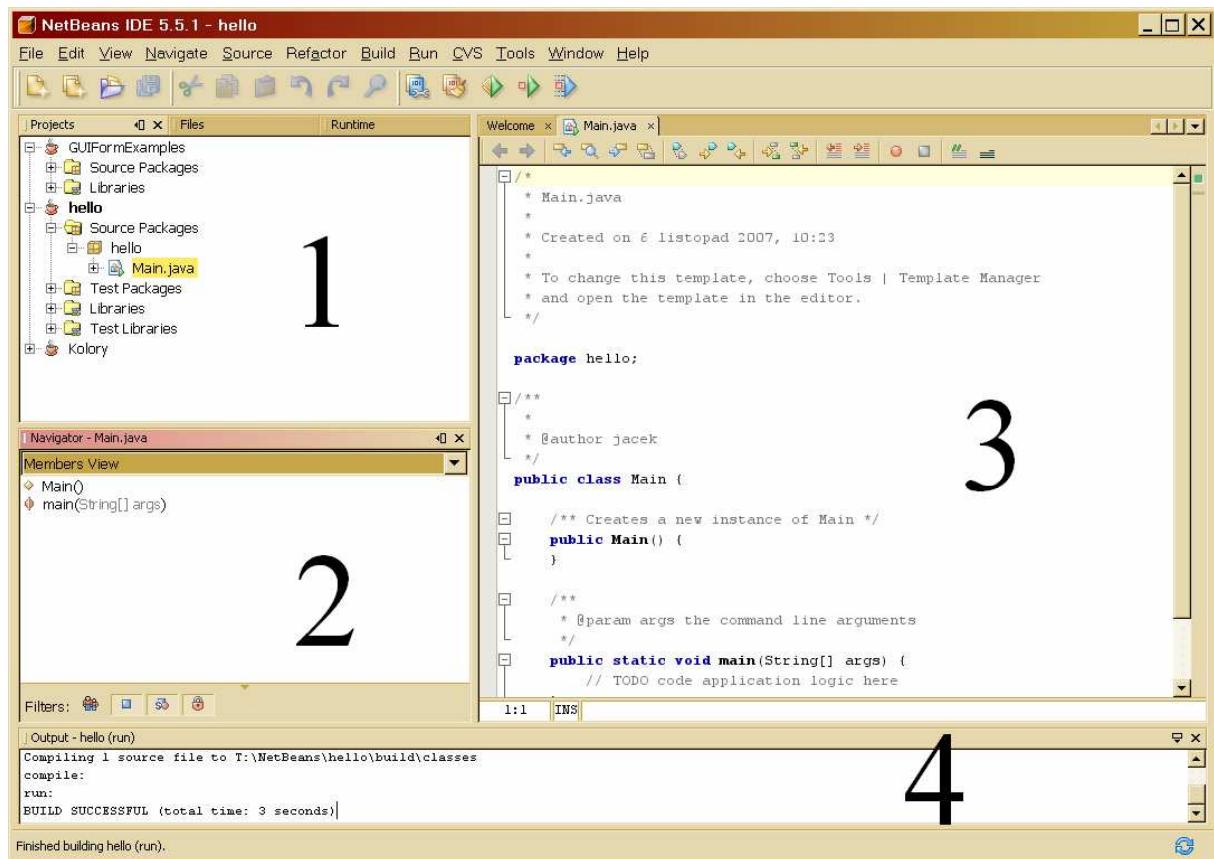
Po naciśnięciu *Finish* kreator zakończy działanie i zostawi nas sam na sam z nowym projektem. W liście plików projektu znajduje się plik *Main.java* (o ile pozostawiliśmy zaznaczoną odpowiednią opcję kreatora).

Możemy natychmiast skompilować ów „pusty” projekt naciskając klawisz *F6* lub klikając ikonę *Run Main Project* na pasku narzędzi.

Przewodnik po oknach i zakładkach IDE

Zintegrowane środowisko programistyczne NetBeans składa się z czterech podstawowych okien (rysunek 3). W domyślnym ustawieniu z lewej strony od góry znajdziemy okno projektu (numer 1 na rysunku) zawierające listę projektów i ich plików (okno *Projects*). Poniżej znajduje się navigator (numer 2) ułatwiającego orientację w bieżącym pliku oglądanym w edytorze (numer 3). Navigator umożliwia wyświetlenie listy metod i pól zdefiniowanych w klasie (a tym samym w pliku) lub ścieżkę dziedziczenia. Na samym dole widać okno wyświetlające komunikaty kompilacji oraz zawartość standardowego strumienia wyjścia i w czerwonym kolorze — standardowego strumienia błędów uruchamianych apletów i aplikacji (*Output*). Główną część zajmuje wspomniane już okno edytora.

¹ Pakiet grupuje klasy pozwalając na kontrolę dostępu i unikanie konfliktów nazw.



Rysunek 3. Okna NetBeans podczas pracy z kodem

Aplikacja konsolowa Hello World!

W pliku *Main.java* znajduje się klasa *Main*, która posiada metodę *main*. To ta metoda zostanie uruchomiona w momencie uruchamiania aplikacji. Dopiszmy do niej polecenie wysyłające do standardowym strumienia wyjścia łańcuch „Hello World!”.

Listing 1. Plik *Main.java* z usuniętymi komentarzami

```
package hello; //pakiet

public class Main { //klasa
    public Main() { //konstruktor
    }

    public static void main(String[] args) { //entry point
        System.out.println("Hello World!");
    }
}
```

Ponownie naciśnijmy *F6* – wśród komunikatów wyświetlanych w podoknie *Output* zobaczymy także łańcuch „Hello World!”.

Uruchamianie aplikacji spoza IDE

Aby uruchomić aplikację *hello* z konsoli/wiersza poleceń należy:

1. W systemie Windows uruchamiamy wiersz poleceń, w systemach Linux i Solaris konsolę.

2. Musimy przejść do podkatalogu *classes* katalogu projektu tak, żeby skompilowana klasa aplikacji *Main.class* znajdowała się względem bieżącego katalogu w podkatalogu *hello*. W moim przypadku przeszedłem do katalogu *T:\NetBeans\hello\build\classes*.
3. Następnie wydajemy polecenie (ścieżka do *java* jest oczywiście przykładowa tylko dla systemu Windows, w Linuksie i Solaris można ją w ogóle pominąć):

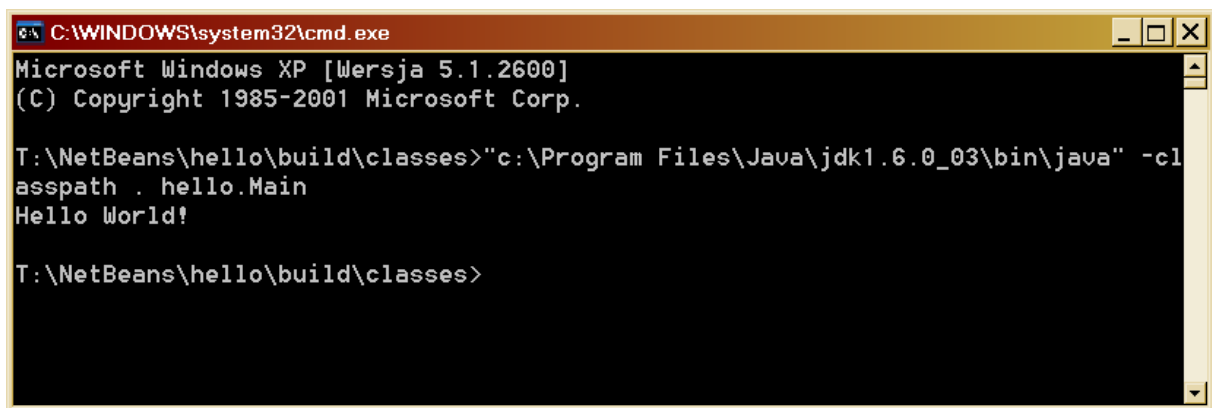
```
c:\Program Files\Java\jdk1.6.0_03\bin\java -classpath . hello.Main
```

<<koniec ćwiczenia>>

Uwaga! Wielkość liter przy pisaniu nazwy klasy jest istotna.

Efekt wykonania ćwiczenia można zobaczyć na rysunku 4. Jeżeli chcemy wskazywania ścieżki do bieżącego katalogu w parametrze *classpath* możemy ustalić zmienną środowiskową *CLASSPATH* tak, aby zawierała ścieżkę do katalogu *T:\NetBeans\hello\build\classes* (wtedy polecenie można będzie wydawać z dowolnego katalogu roboczego) lub umieścić w niej po prostu odniesienie do katalogu bieżącego:

```
set CLASSPATH=%CLASSPATH%;.
```

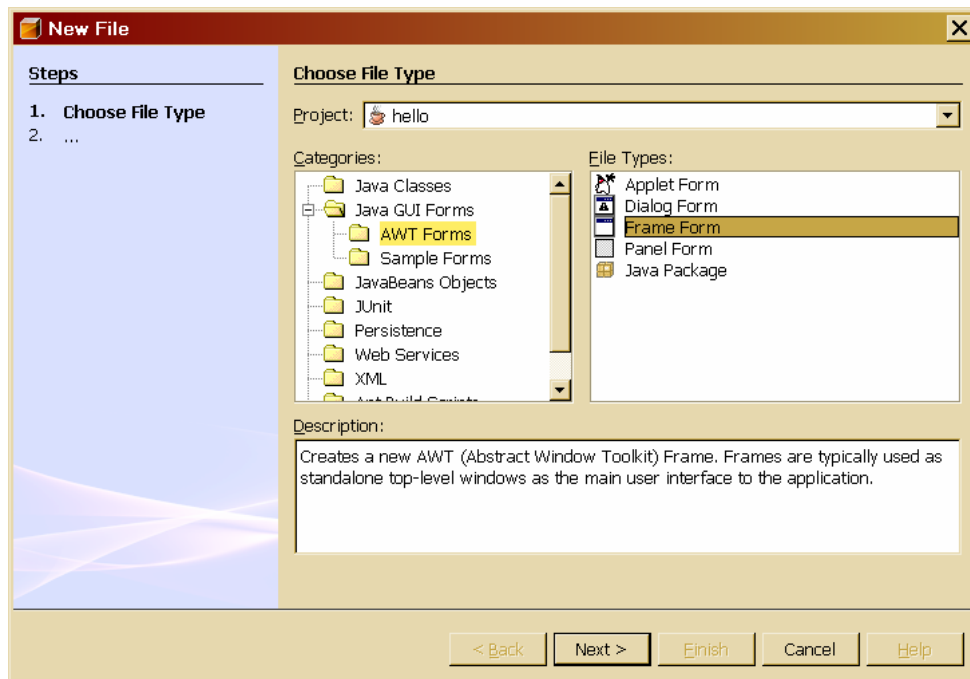


Rysunek 4. Efekt wykonania aplikacji „Hello world!” w konsoli

Chciałbym teraz w kilku bardzo prostych ćwiczeniach przedstawić podstawowe zagadnienia projektowania w NetBeans. Nie będzie to systematyczny przegląd możliwości, a raczej przybliżenie najczęściej spotykanych w praktyce zagadnień. Przegląd ten będziemy kontynuować, już w bardziej uporządkowany sposób, w części 3.

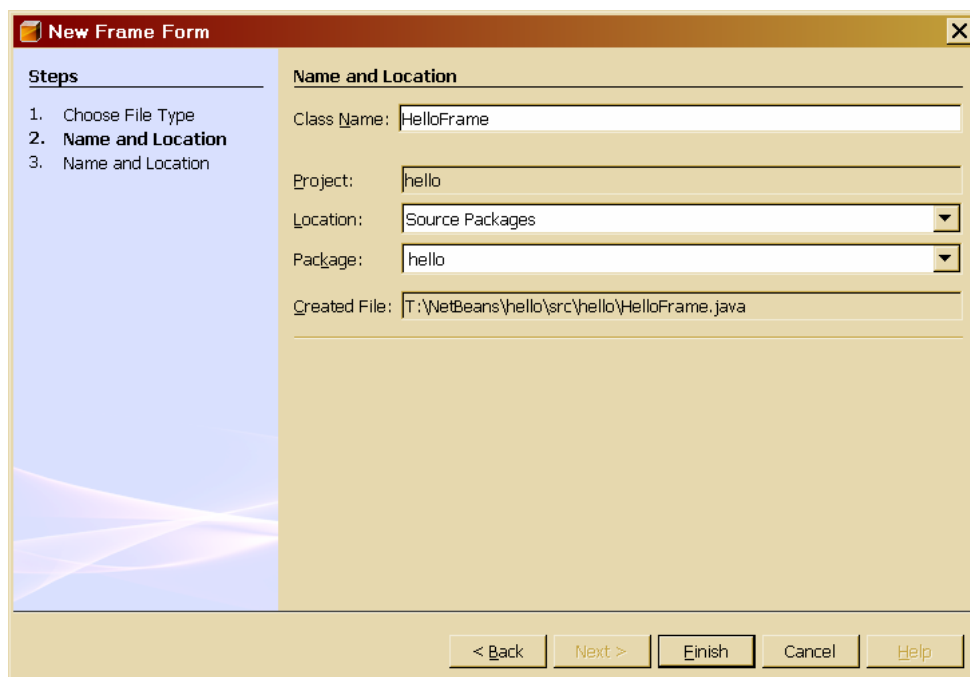
Aplikacja GUI (AWT)

1. Z menu *File*, wybieramy *New File...*
2. W oknie *New File* (rysunek 5) w drzewie *Categories* rozwijamy drzewo *Java GUI Forms, AWT Forms*.
3. W liście *File Types* zaznaczamy pozycję *Frame Form*.
4. Klikamy *Next >*.



Rysunek 5. Kreator formy

5. W oknie *New Frame Form* (rysunek 6) wskazujemy nazwę klasy (pole *Class Name*). Nazwę pakietu pozostawiamy równą *hello*.



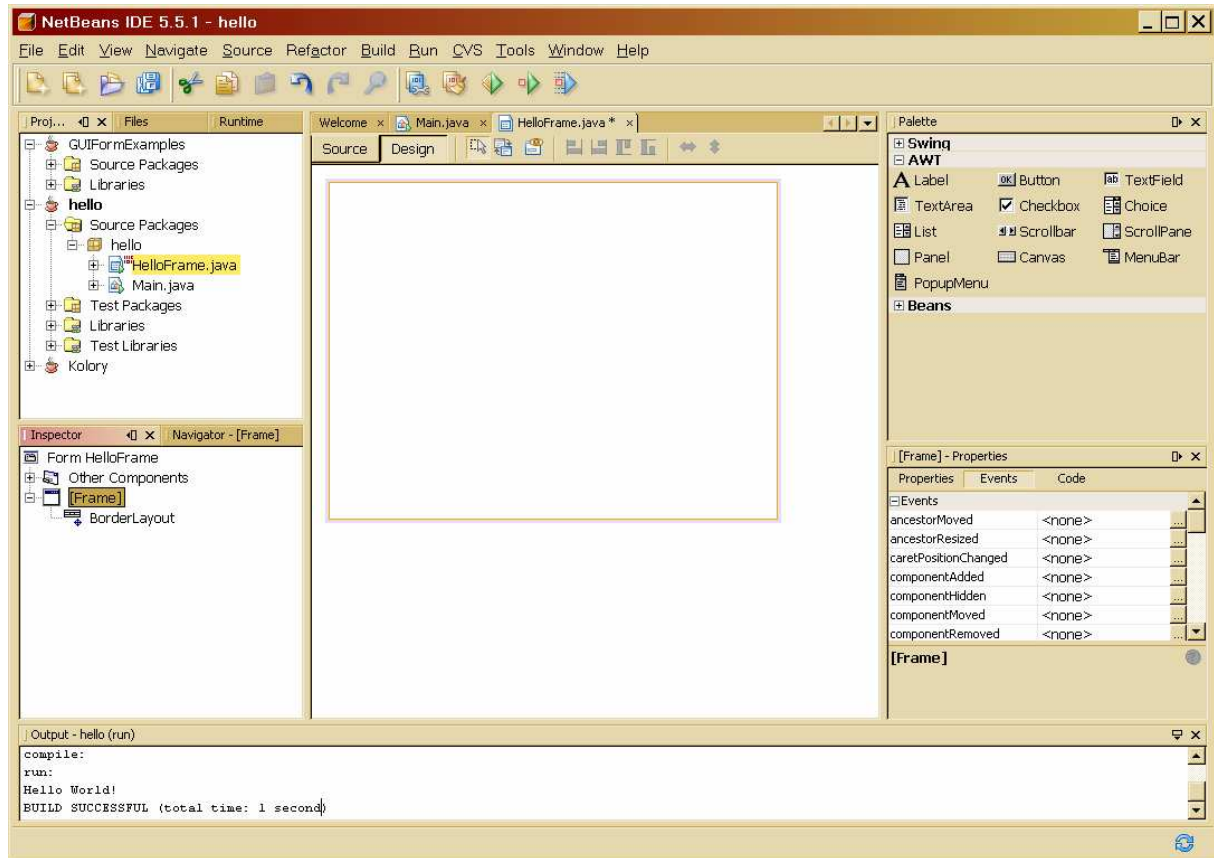
6. Klikamy *Finish*.

<< koniec ćwiczenia >>

GUI = *Graphics User Interface*

AWT = *Abstract Window Toolkit*, biblioteka komponentów bazujących na API systemu operacyjnego (wygląd interfejsu może zależeć od systemu operacyjnego). AWT nie jest już rozwijane, ale ze względu na zgodność z wersją Javy dodawaną przez Microsoft do Windows warto rozważyć jej użycie. Alternatywa to biblioteka Swing.

Po wykonaniu powyższych czynności zmieni się wygląd IDE (rysunek 7). Zamiast edytora widzimy teraz podgląd okna. Z jego prawej strony widoczna jest paleta komponentów, z których możemy zbudować interfejs aplikacji, a pod nim okno własności. W tej chwili widoczne są własności okna. Natomiast w podoknie, które zajmował nawigatory widoczny jest także inspektor. Prezentuje on hierarchię zawierania komponentów w komponentach-pojemnikach (ang. *container*). Czytelnikom znającym VCL powinna się narzucać analogiczna różnica między właścicielem (*Owner*) i rodzicem (*Parent*) komponentów z tej biblioteki.



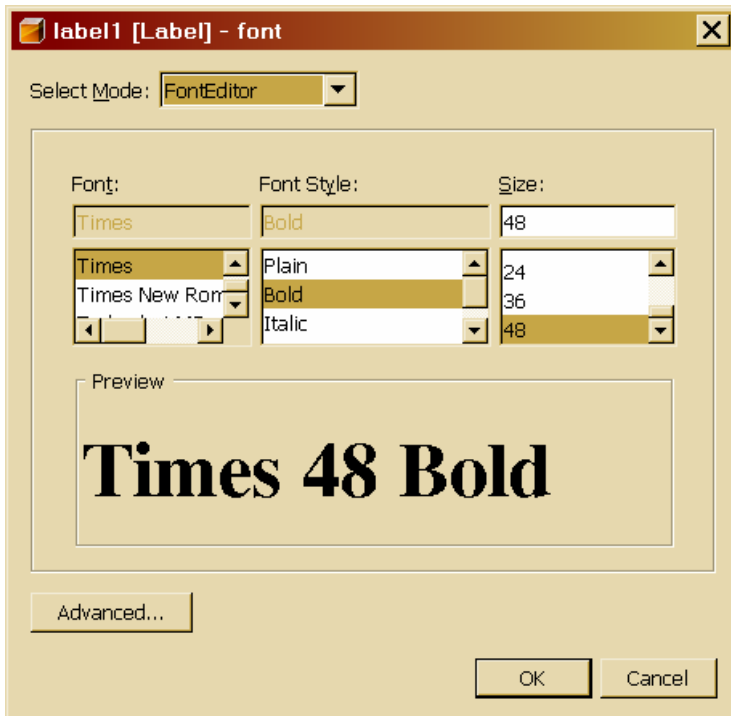
Rysunek 7. Widok projektowania okna AWT

Między edytorem klasy okna, a jego podglądem można przełączać się za pomocą przycisków *Source* i *Design* widocznych na pasku narzędzi zakładki *HelloFrame.java*. Każda zmiana w widoku projektowania ma swoje odzwierciedlenie w kodzie – nie ma osobnych plików przechowujących ustawienia interfejsu (por. *.dfw/.afm* w VCL, *.xaml* w WPF). Zepsucie kodu klasy *HelloFrame* uniemożliwia wyświetlenie podglądu okna.

Plik XML *HelloFrame.form*.

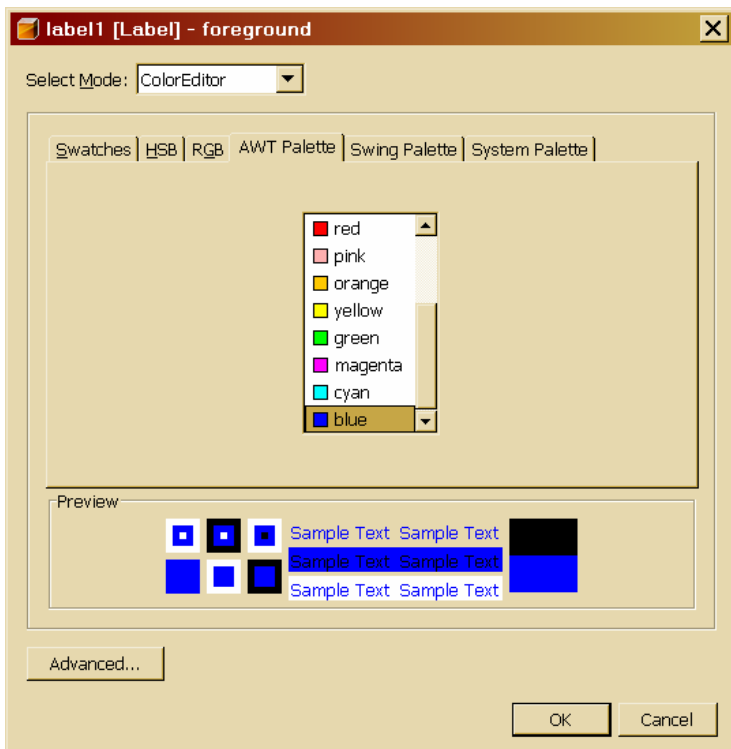
Projektowanie interfejsu AWT

1. W palecie komponentów zwińmy grupę *Swing*, a rozwińmy grupę *AWT*.
2. Zaznaczmy komponent *Label* i umieśmy go na połowie wysokości okna.
3. Ponieważ po umieszczeniu go na formie jest on zaznaczony, jego własności będą widoczne w oknie *Properties*
 - a. Zmieńmy jego własność *Alignment* na *CENTER*. W ten sposób znajdzie się w centrum całego okna.
 - b. Zmieńmy czcionkę (własność *Font*) korzystając z edytora własności (przycisk z trzema przyciskami) – por. rysunek 8.



Rysunek 8. Edytor wyboru czcionki

c. Zmieńmy kolor napisu (własność `foreground`) – rysunek 9.



Rysunek 9. Wybór koloru spośród standardowego zestawu kolorów

d. Na koniec zmieniamy własność `text` odpowiadającą za treść napisu na „Hello World!”.

<< koniec ćwiczenia >>

Po wszystkich opisanych wyżej zmianach możemy nacisnąć klawisz `Shift+F6`, który uruchamia metodę `main` bieżącej klasy. Kreator utworzył automatycznie taką metodę w klasie `HelloFrame` opisującej okno. Okno powinno wyglądać mniej więcej jak to widoczne na rysunku 10.



Rysunek 10. Okno aplikacji z etykietą Label

Tworzenie okna w metodzie `hello.Main.main`

Aby okno opisane klasą `hello.HelloFrame` tworzone było w metodzie `hello.Main.main` należy w niej umieścić poniższe polecenie:

Listing 2. Tworzenie okna w metodzie `main`

```
public static void main(String[] args) {  
    System.out.println("Hello World!");  
    new HelloFrame().setVisible(true);  
}
```

[Mechanizm uzupełniania kodu]

Tworzy ono obiekt – instancję klasy `HelloFrame` i wywołuje jej metodę `setVisible` z argumentem `true`, która pokazuje okno na ekranie. Po wzór można zajrzeć do metody `hello.HelloFrame.main`.

Teraz można uruchomić aplikację naciskając po prostu klawisz `F6`. Można też uruchomić go poleceniami: `java hello.Main` lub `java hello.HelloFrame`.

Jeżeli zależy nam na przechowaniu referencji do obiektu okna, możemy wyróżnioną linię w powyższym listingu zamienić na:

```
HelloFrame hf=new HelloFrame();  
hf.setVisible(true);
```

Aplet AWT

Mając doświadczenie z projektowaniem interfejsu AWT możemy niemal automatycznie powtórzyć czynności z poprzedniego ćwiczenia do sytuacji, w której projektować będziemy interfejs apletu.

1. Postępujemy podobnie, jak przy tworzeniu okna AWT tzn. za pomocą polecenia menu *File, New File...* otwieramy okno widoczne na rysunku 5.
2. Jednak tym razem zaznaczamy pozycję *Applet Form*.
3. Klikamy *Next >*.
4. Nadajemy nowej klasie nazwę `HelloApplet`. W konsekwencji plik, w którym ta klasa będzie zapisana nazywać się będzie `HelloApplet.java`.
5. Klikamy *Finish*.

<< koniec ćwiczenia >>

Do projektu dodany zostanie jeszcze jeden plik o nazwie `HelloApplet.java`. Jak widać jeden projekt NetBeans może zawierać klasy różnego typu. Widok projektowania apletu jest niemal identyczny, jak okna aplikacji. Powtarzamy zatem czynności, które doprowadziły do utworzenia okna z napisem „Hello World!”:

1. Na palecie AWT zaznaczmy komponent `Label` i umieścmy go na połowie wysokości okna.

2. Zmieniamy jego własność `Alignment` na `CENTER`. Zmieniamy czcionkę (własność `Font`). Zmieniamy kolor napisu (własność `foreground`). Oraz zmieniamy własność `text` odpowiadającą za treść napisu na „Hello World!”.

<< koniec ćwiczenia >>

Aplet możemy uruchomić tylko w kontekście innej aplikacji – zazwyczaj przeglądarki WWW. Jednak NetBeans po naciśnięciu `Shift+F6` uruchamia aplet korzystając z dołączonej do JDK aplikacji `appletviewer`.

Osadzanie apletu w dokumencie HTML

Kreator apletu tworzy i dodaje do projektu stowarzyszoną z apletem stronę HTML, która pozwala na testowanie działania apletu (`T:\NetBeans\hello\build\HelloApplet.html`). Przykładowa zawartość dokumentu przedstawiona jest na listingu 3. Pozwala ona zorientować się w sposobie, w jaki aplety osadzone są w kodzie HTML.

Listing 3. Kod HTML stworzony przez kreator apletu po usunięciu komentarzy

```
<HTML>
<HEAD>
  <TITLE>Applet HTML Page</TITLE>
</HEAD>
<BODY>
<H3><HR WIDTH="100%">Applet HTML Page<HR WIDTH="100%"></H3>
<P>
<APPLET codebase="classes" code="hello/HelloApplet.class" width=350 height=200></APPLET>
</P>
<HR WIDTH="100%"><FONT SIZE=-1><I>Generated by NetBeans IDE</I></FONT>
</BODY>
</HTML>
```

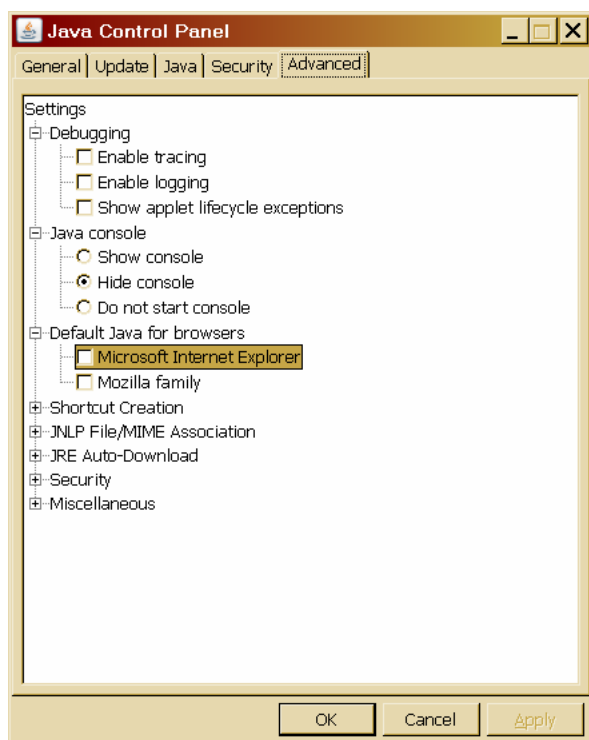
Część wytłuszczona odpowiada za osadzenie pakietu w dokumencie. Należy pamiętać o uwzględnieniu nazwy pakietu/katalogu w atrybucie `code`. Atrybut `codebase` jest odpowiednikiem `classpath` z polecenia `java`. Aby aplet został załadowany, musi znajdować się w podkatalogu `hello` względem katalogu `classes`, a plik i klasa główna muszą mieć nazwę `HelloApplet.class`.

Uwaga! Podczas uruchamiania strony z apletem przeglądarka może blokować jego załadowanie.



Rysunek 11. Aplet w przeglądarce.

Przy obecnych ustawieniach projektu i JDK (domyślnych po instalacji) uruchomienie apletu w przeglądarce (IE lub Mozilla) zaangażuje JRE w wersji 1.6. Wsparcie dla przeglądarki można wyłączyć w *Panelu Sterowania*, ikona *Java* (rysunek 12). Wówczas będzie ona starała się korzystać z Javy 1.1 wbudowanej w Windows (wersja Microsoft, a nie Sun, dodawana do Windows do wersji XP). Takiej wersji Java nie wspiera jednak NetBeans, który pozwala na wsteczną zgodność jedynie z Java 1.2 (ustawiane we własnościach projektu dostępnych w menu *File*, pozycja *Source level*).



Rysunek 12. Kontrola Javy w przeglądarce

Zmiana nazwy pliku/klasz (refactoring)

Ze względu na zależność między nazwą klasy i nazwą pliku oraz wieloma miejscami, gdzie znajdują się odwołania do nazwy klasy do zmiany nazwy najlepiej użyć narzędzia refactoringu.

1. W podoknie projektu klikamy prawym klawiszem myszy na pliku `HelloApplet.java` i z menu kontekstowego wybieramy `Refactor, Rename...`
2. W oknie `Rename Class HelloApplet` wpisujemy nową nazwę `HelloAppletAWT`.
3. Klikamy `Next >`.
4. Jeżeli zaznaczona jest opcja `Preview All Changes`, w podoknie `Output` pojawią się informacje związane ze zmianą nazwy. Obok jest przycisk `Do Refactoring`, który należy nacisnąć. Jeżeli nie jesteśmy pewni tego, co robimy należy kliknąć `Cancel`.

<< koniec ćwiczenia >>

Poprawność zmiany możemy sprawdzić naciskając `Shift+F6` – jeżeli aplet uruchomi się, wszystko jest w porządku. Warto oczywiście zajrzeć również do kodu pliku.

Ćwiczenie: W analogiczny sposób zmienić klasę okna `HelloFrame` na `HelloFrameAWT`. Zmianie ulegnie także kod dodany do `hello.Main.main`.

Aplikacja GUI (Swing) i projektowanie interfejsu Swing

Forma (ang. *form*) = okno

Kontynuujemy rozwój projektu `hello`. Dodamy do niego aplikację i aplet Swing (część JFC = ang. *Java Foundation Classes*, analogicznie do *MFC*). Jest to nowa biblioteka kontrolki niezależna od systemu-gospodarza.

1. Z kreatora uruchamianego poleceniem `File, New File...` wybieramy `Java GUI Forms` w `Categories`, a `JFrame Form` w `File Type`.
2. Klikamy `Next >`.
3. Nową klasę nazywamy `JHelloFrame` („J” jest typowym prefiksem kontrolki *Swing*).
4. Klikamy `Finish`.
5. Po chwili pojawi się podgląd formy, na której umieszczamy komponent `JLabel`. Zwróćmy uwagę, że inny jest sposób umieszczania kontrolki na formie. Możemy kontrolować ich dowolne położenie i wielkość. W *AWT* decydował o tym „layout” formy.
6. Korzystając z edytorów własności zmieniamy własności napisu (`font`, `foreground`, `text`, `toolTipText`).
7. Możemy zmienić tytuł formy korzystając z jej własności `title` i (należy ją kliknąć na podglądzie, aby jej własności widoczne były w podoknie *Properties*).

<< koniec ćwiczenia >>

Następnie naciskamy `Shift+F6`, aby skompilować i uruchomić nową klasę (rysunek 13).

Okno możemy otworzyć poleceniem `java hello.JHelloFrame` (oczywiście z odpowiednimi ścieżkami i katalogiem roboczym) lub `java hello.Main` jeżeli do metody `hello.Main.main` dodamy polecenie `new JHelloFrame().setVisible(true);`.



Rysunek 13. „Hello World!” w wersji Swing

Ćwiczenie: przygotować aplet Swing z napisem „Hello World!” (typ pliku *JApplet Form*, nazwa klasy *JHelloApplet* itd.)

Projekt „Kolory”

Projektowanie interfejsu to tylko wstępny etap tworzenia aplikacji lub apletu. Powinna ona jeszcze reagować na działania użytkownika. Do tego służą zdarzenia.

[Model zdarzeniowy] vs. main

Idea

Aby ściśle określić kolor, niezbędne jest podanie trzech parametrów. Niezbędny jest wobec tego trójwymiarowy układ współrzędnych. Jest bardzo wiele układów współrzędnych określających kolory. Jak to możliwe?

Wyjaśnię to na prostszym przykładzie: jeżeli chcemy określić, jaka woda ma lecieć z kranu, musimy wykorzystać dwa parametry. Jednak i tu możemy użyć różnych układów współrzędnych. Możemy powiedzieć, jak szybko woda ma lecieć i jaka ma być jej temperatura (nowoczesne kranu jednouchwytowe) lub ile ma lecieć ciepłej i ile zimnej wody (tradycyjne kranu dwugałkowe). Podobnie w przypadku kolorów możemy określać kolor podając natężenie wybranych składowych (np. czerwonego, zielonego i niebieskiego — układ RGB), czyli sposób, w jaki kolor generują monitory i drukarki, lub podając barwę, nasycenie i jasność (rodzina układów HSB), co jest bliższe temu, w jaki sposób o kolorach mówimy na co dzień.

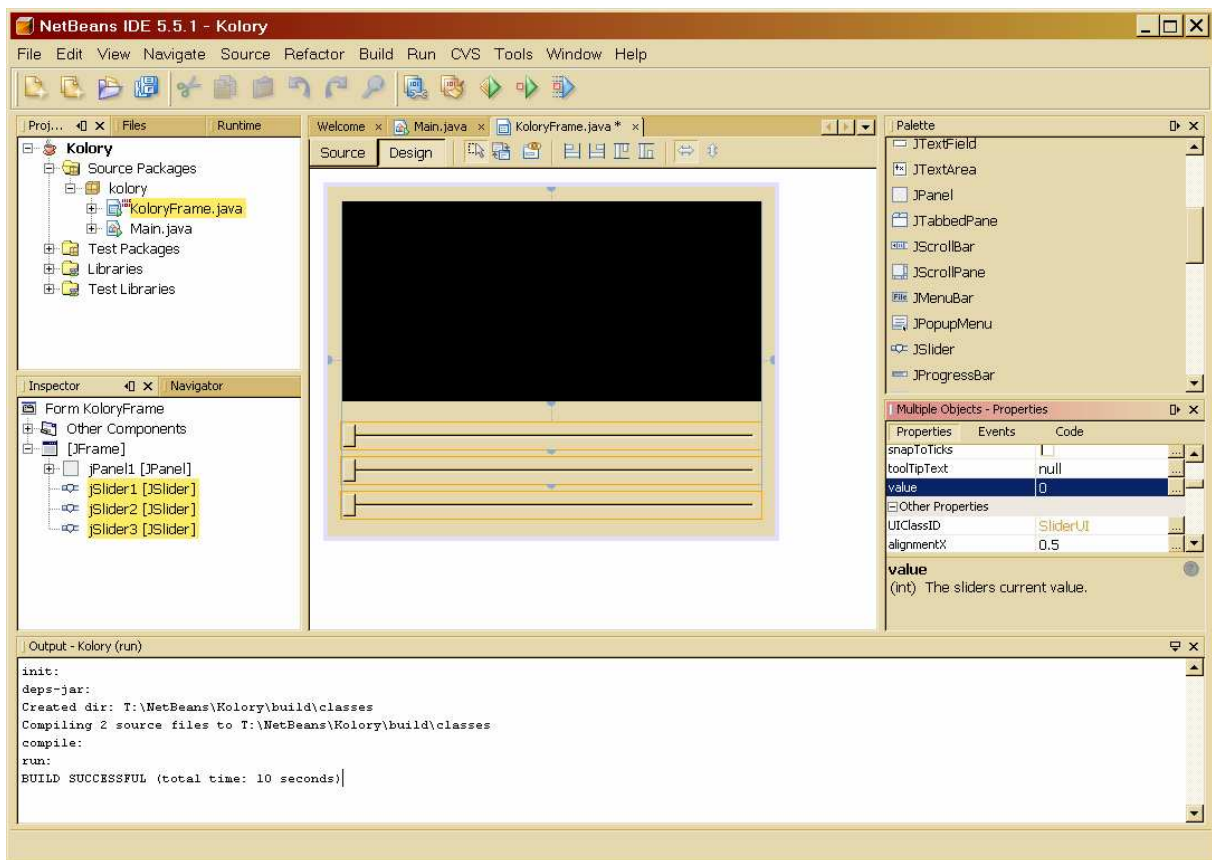
Tworzenie projektu

1. Zamknijmy projekt „hello” wybierając z menu *File* polecenie *Close „hello”*.
2. Stwórzmy nowy projekt typu *Java Application* o nazwie *Kolory*.
3. Dodajmy do niego okno Swing (aplikacja), jego klasa niech nazywa się *KoloryFrame*.
4. Do metody *kolory.Main.main* z pliku *Main.java* dodajmy polecenie

```
new KoloryFrame().setVisible(true);
```

Projektowanie interfejsu Swing

1. Zmieniamy zakładkę na *KoloryFrame.java, Design*.
2. W górnej części formy umieszczamy komponent `JPanel`. Zmieniamy jego kolor (własność `background`) na czarny.
3. W dolnej trzy suwaki (`JSlider` lub alternatywnie `JScrollBar`).
4. Rozmiar formy możemy zmienić na podglądzie.
5. Przytrzymując klawisz *Shift* zaznaczamy wszystkie trzy suwaki i zmieniamy ich własności:
 - a) ustawiamy `maximum` na `255`,
 - b) wartość `minimum` pozostaje równa `0`.
 - c) wartość `value` zmieniamy na `0` (suwaki zmieniają pozycje na podglądzie).



Rysunek 3.2. Podgląd interfejsu

<< koniec ćwiczenia >>

Paski przewijania posłużą do ustalenia składowych RGB koloru panelu. Dlatego powinny przyjmować liczby z zakresu od 0 do 255 (a więc 256 możliwych kombinacji, które można zakodować za pomocą jednego bajtu).

Zakończyliśmy przygotowywanie interfejsu apletu. Teraz będziemy musieli włożyć silnik pod tę maskę. Zadaniem apletu będzie umożliwienie użytkownikowi określenie koloru panelu `pane11` za pomocą pasków przewijania określających składowe `R`, `G` i `B` koloru. Konieczne jest zatem, aby aplet reagował na zdarzenia `adjustValueChanged` każdego z pasków. Z tym zdarzeniem każdego z trzech pasków przewijania zwiążemy jedną metodę, która wykona odpowiednie obliczenia i zmieni kolor panelu.

Zdarzenia

Aby stworzyć metodę zdarzeniową wywoływaną po zmianie pozycji każdego z pasków przewijania, która będzie modyfikować kolor panelu w zależności od położenia suwaków:

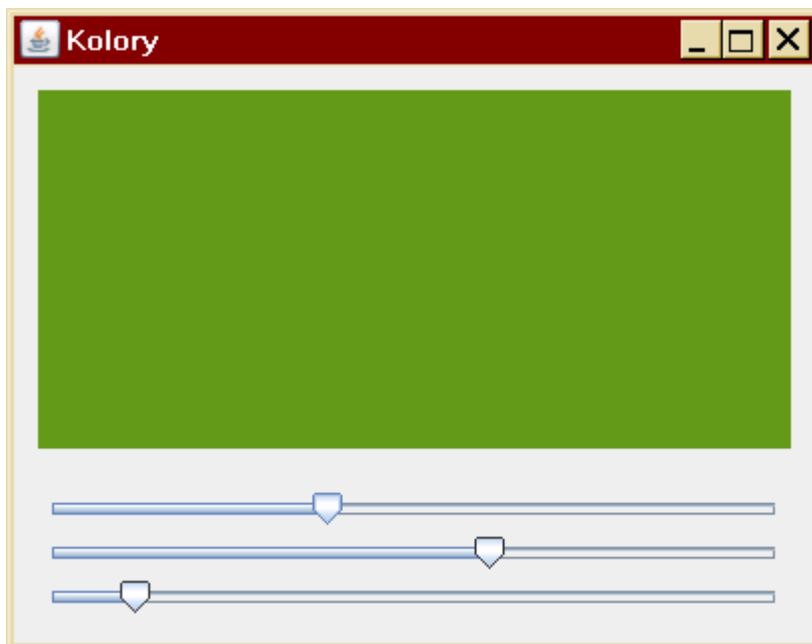
1. W widoku projektowania (zakładka *Design*) zaznaczamy pierwszy pasek przewijania `jSlider1`.
2. Przechodzimy na zakładkę *Events* w oknie *jSlider1 [JSlider] - Properties*.
3. Klikamy pole edycyjne przy zdarzeniu `stateChanged` i naciskamy *Enter*.
4. Zostaniemy przeniesieni do edytora kodu (do zakładki *Source*), który będzie pokazywał pustą na razie metodę o nazwie `jSlider1stateChanged`.
5. Do „nagłówka” pliku *KoloryFrame.java* dodajemy instrukcję

```
import java.awt.Color;
```

6. Następnie uzupełniamy stworzoną wcześniej metodę zdarzeniową następującym kodem (wyróżniona część):

```
private void jSlider1StateChanged(javax.swing.event.ChangeEvent evt) {
    Color kolor=new Color(scrollbar1.getValue(),
        scrollbar2.getValue(),
        scrollbar3.getValue());
    panell.setBackground(kolor);
}
```

<<koniec ćwiczenia>>



Rysunek 15. Działająca aplikacja Kolory

W pierwszej linii ciała metody deklarowany jest obiekt o nazwie `kolor` typu `java.awt.Color`, który tworzymy korzystając z konstruktora przyjmującego jako argumenty trzy wartości składowych RGB typu `int` z zakresu od 0 do 255². Jako wartości składowe koloru podajemy pozycje naszych pasków przewijania.

Mając obiekt `kolor` opisujący określony przez pozycję suwaków kolor, możemy łatwo zmienić kolor panelu. Służy do tego znana nam już metoda `setBackground` paneli, która jako jedyny argument przyjmuje obiekt klasy `Color`. Zdefiniowany przed chwilą obiekt `kolor` nadaje się do tego doskonale.

Uwaga! Usuwanie zbędnych metod zdarzeniowych możliwe jest w następujący sposób: 1) usuwamy zawartość metody na zakładce *Source*, 2) usuwamy jej nazwę w podoknie *Properties* na zakładce *Design*.

Aby z kodu metody wyeliminować referencję `kolor` możemy napisać tę metodę następująco:

Listing 4. Taki sposób tworzenia obiektów jest bardzo charakterystyczny dla kodu pisanego w Javie

```
private void jSlider1StateChanged(javax.swing.event.ChangeEvent evt) {
```

² Klasa `Color` posiada także konstruktory przyjmujące liczby rzeczywiste z zakresu od 0.0 do 1.0.


```

jPanel1.setBackground(new java.awt.Color(jSlider1.getValue(),
                                         jSlider2.getValue(),
                                         jSlider3.getValue()));
}

```

Ponieważ obiekt koloru zdefiniowanego na podstawie pozycji suwaków był wykorzystywany tylko w jednym miejscu, można pominąć deklarowanie referencji `kolor` dla tego obiektu tworząc obiekt lokalnie przy wywołaniu metody `jPanel1.setBackground`. Taki sposób tworzenia obiektów jest bardzo charakterystyczny dla Javy. W C++ prowadziłby do wycieku pamięci, bo nie ma przechowanego wskaźnika pozwalającego na zwolnienie pamięci zajmowanej przez obiekt.

Aby związać napisaną przez nas metodę ze zdarzeniami `jSlider1StateChanged` dwóch pozostałych pasków przewijania:

1. Przejdźmy do widoku projektowania.
2. Zaznaczmy obiekt `jSlider1` w nawigatorze lub na podglądzie apletu.
3. W inspektorze zmienimy zakładkę na *Events*.
4. Klikamy raz pole edycyjne przy zdarzeniu `adjustmentValueChanged` i wpisujemy w nim nazwę metody `jSlider1StateChanged`³.
5. Należy pamiętać o potwierdzeniu (naciskając *Enter*).
6. Czynność trzeba powtórzyć osobno dla obiektu `jSlider3`.
7. Skompilujmy i uruchommy projekt naciskając klawisz *F6*.

<<koniec ćwiczenia>>

HSB

Aby wybór koloru stał się bardziej intuicyjny, możemy dodać do aplikacji możliwość określenia koloru za pomocą układu współrzędnych HSB. Jest on bardziej intuicyjny niż RGB. W istocie to za pomocą tego układu określamy kolory w potocznym języku mówiąc o barwie, jasności i nasyceniu. Nie należy się obawiać trudności związanych z nowym układem współrzędnych — układ HSB jest również zaimplementowany w bibliotekach Javy. Używana przez nas klasa `Color` posiada już odpowiedni zestaw metod.

Aby dodać do interfejsu apletu pole wyboru pozwalające na wybór układu *RGB* lub *HSB*:

1. Zmieniamy zakładkę na *Design*.
2. Umieszczamy na oknie dwa komponenty `JRadioButton` z palety *Swing*.
3. Za pomocą inspektora zmieniamy etykietę pierwszego (właściwość `text`) na *RGB*, a drugiego na *HSB*. Właśność `text` można też zmienić klikając dwukrotnie komponent na podglądzie formy.
4. Właśność `selected` pierwszego z nowych komponentów zmieniamy na `true` (zaznaczamy pole opcji przy tej własności w podoknie *Properties*).
5. Do klasy okna dodajemy komponent `ButtonGroup`, również z palety *Swing*. Powstaje niewidoczny na podglądzie obiekt `buttonGroup1`.
6. Następnie zaznaczamy oba dodane wcześniej pola opcji (`jRadioButton1` i `jRadioButton2`) i w podoknie *Properties* ustawiamy ich właściwość `buttonGroup` wybierając z rozwijanej listy obiekt `buttonGroup1`.

³ Po kliknięciu pola w inspektorze przy zdarzeniu `stateChanged`, w którym można wpisać nazwę metody, pojawi się domyślna nazwa funkcji. Wystarczy tylko zmienić cyfrę przy „jSlider” w nazwie metody.

<<koniec ćwiczenia>>

Komponent `ButtonGroup` zmienia niezwiązaną grupę komponentów `JRadioButton` w grupę pól wyboru, spośród których tylko jeden może być zaznaczony. Klikając pole `HSB` zaznaczamy je, a jednocześnie usuwamy zaznaczenie `RGB`.

Aby umożliwić definiowanie koloru we współrzędnych HSB, musimy zmodyfikować metodę zdarzeniową związaną ze zmianą pozycji pasków przewijania. Jeżeli zaznaczony jest `JRadioButton2`, należy zamiast standardowego konstruktora klasy `Color` wykorzystać do określenia koloru metodę `Color.getHSBColor`. Jest to metoda statyczna, tzn. można ją wywołać na rzecz klasy, a nie jej instancji. Metoda ta zwraca referencję do nowego obiektu klasy `Color`. Można ją więc traktować jak alternatywny konstruktor dla współrzędnych HSB. Metoda ta przyjmuje trzy argumenty typu `float` z wartościami z zakresu od 0.0 do 1.0. Potrzebne będzie wobec tego odpowiednie przeskalowanie pozycji odczytanych z suwaków.

Aby w metodzie zdarzeniowej `scrollbar1_adjustmentValueChanged` uwzględnić możliwość definiowania kolorów za pomocą współrzędnych `HSB`:

Modyfikujemy metodę `scrollbar1_adjustmentValueChanged` zgodnie z wyróżnionymi fragmentami listingu 5.

Listing 5. Dodane fragmenty kodu ustalają kolor na podstawie współrzędnych HSB

```
private void jSlider1StateChanged(javax.swing.event.ChangeEvent evt) {
    //RGB
    if(jRadioButton1.isSelected())
        jPanel1.setBackground(new Color(jSlider1.getValue(),
                                        jSlider2.getValue(),
                                        jSlider3.getValue()));

    //HSB
    if(jRadioButton2.isSelected())
        jPanel1.setBackground(Color.getHSBColor(
            jSlider1.getValue()/(float)jSlider1.getMaximum(),
            jSlider2.getValue()/(float)jSlider2.getMaximum(),
            jSlider3.getValue()/(float)jSlider3.getMaximum()));
}
```

<<koniec ćwiczenia>>

W momencie przełączania między współrzędnymi kolorów zmienia się sens wartości określanych przez paski przewijania, ale w naszym aplecie pozycje pasków pozostają niezmienione. Powoduje to skokową zmianę kolorów przy poruszeniu któregoś z pasków. Aby pozbyć się tego efektu, należy równocześnie ze zmianą współrzędnych zmienić pozycję suwaków na paskach tak, żeby odpowiadały współrzędnym bieżącego układu.

Aby przy zmianie układu współrzędnych za pomocą pól wyboru aplet uaktualniał pozycje suwaków:

1. W widoku projektowania zaznaczamy obiekt `JRadioButton1`.
2. Za pomocą inspektora tworzymy metodę zdarzeniową dla zdarzenia `itemStateChanged` tego obiektu. Będzie ona wywoływana w momencie zmiany stanu komponentu (zaznaczony-niezaznaczony).
3. Metodę uzupełniamy zgodnie z listingiem 6.

Listing 6. Reakcja na wybór współrzędnych RGB

```
private void jRadioButton1ItemStateChanged(java.awt.event.ItemEvent evt) {
    Color kolor=jPanel1.getBackground();
    jSlider1.setValue(kolor.getRed());
    jSlider2.setValue(kolor.getGreen());
    jSlider3.setValue(kolor.getBlue());
}
```

```
}
```

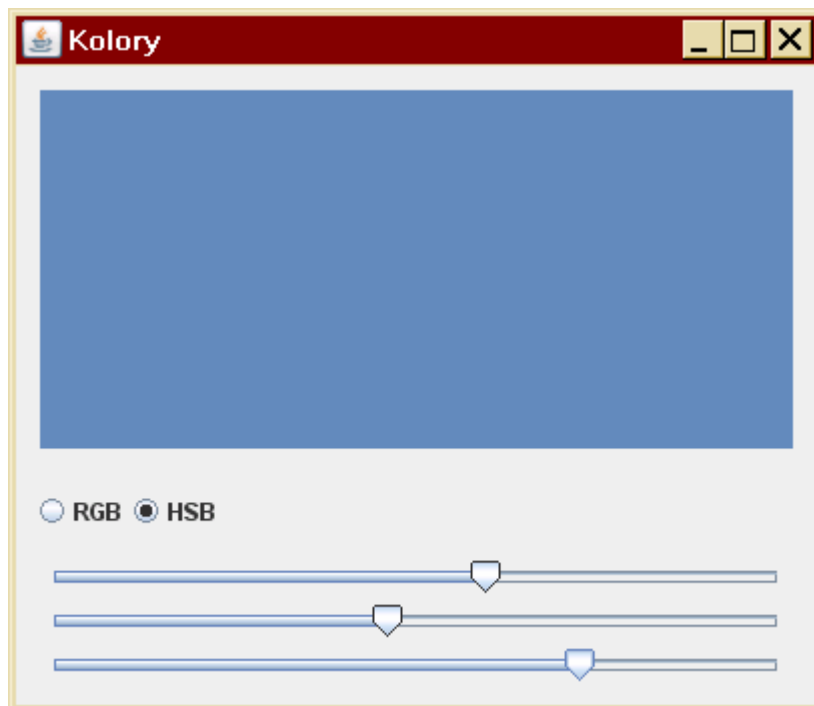
4. Następnie zaznaczamy obiekt `jRadioButton2` i w oknie inspektora również do jego zdarzenia `itemStateChanged` przypisujemy nową metodę zdarzeniową (listing 7).

Listing 7. Metoda aktualizująca położenie suwaków przy zmianie układu współrzędnych kolorów na HSB

```
private void jRadioButton2ItemStateChanged(java.awt.event.ItemEvent evt) {  
    Color kolor=jPanel1.getBackground();  
    float[] hsbkolory=Color.RGBtoHSB(kolor.getRed(),  
                                       kolor.getGreen(),  
                                       kolor.getBlue(),  
                                       null);  
    jSlider1.setValue((int)(255*hsbkolory[0]));  
    jSlider2.setValue((int)(255*hsbkolory[1]) );  
    jSlider3.setValue((int)(255*hsbkolory[2])) ;  
}
```

<<koniec ćwiczenia>>

W przypadku przełączenia na układ RGB odpowiednie wartości składowych koloru pobieramy po prostu z panelu (np. metoda `jPanel1.getBackground().getRed()` zwraca składową `R`). Natomiast po przełączeniu na HSB należy do obliczenia nowych pozycji suwaków wykorzystać statyczną metodę konwertującą `RGBtoHSB` klasy `Color`. Jej argumenty to składowe `R`, `G` i `B`. Czwartym może być referencja do istniejącej tablicy trójelementowej liczb zmiennoprzecinkowych typu `float`. My ustaliśmy czwarty argument jako pusty (ang. `null`), a wówczas metoda tworzy nową tablicę i zwraca referencję do niej przez wartość. Trzy elementy tej tablicy (o indeksach od 0 do 2) to oczywiście składowe `H`, `S` i `B`, które, przed zastosowaniem do zmiany pozycji pasków przewijania, musimy przeskalować na zakres 0 – 255 i rzutować na typ całkowity `int`.



Rysunek 16. Wybór koloru za pomocą układu współrzędnych HSB

W domu: powtórzyć projekt „Kolory” jako applet Swing lub aplikację/aplet AWT (w przypadku appletu sprawdzić działanie metody `showStatus`)

Debugowanie kodu w NetBeans

Debugowanie = kontrolowane uruchamianie aplikacji/apletu

Informacje o kodzie źródłowym (numery linii) dołączane do skompilowanego kodu

F5 – uruchomienie aplikacji z debugowaniem (główny projekt), **Shift+F5** – przerwanie

[Watches, Call stack]

Nowy projekt Debugowanie













```
package debugowanie;

public class Main {

    private static int kwadrat(int argument)
    {
        return argument*argument;
    }

    public Main() {
    }

    public static void main(String[] args) {
        int a=123456;
        int b=kwadrat(a);
        int c=kwadrat(b);
        System.out.println("a="+a+" b="+b+", c="+c);
    }
}
```

Run	CVS	Tools	Window	Help
	Run Main Project	F6		
	Debug Main Project	F5		
	Test "Debugowanie"	Alt+F6		
	Run File			▶
	Attach Debugger...			
	Finish Debugger Session	Shift+F5		
	Pause			
	Continue	Ctrl+F5		
	Step Over	F8		
	Step Into	F7		
	Step Out	Ctrl+F7		
	Run to Cursor	F4		
	Run Into Method	Shift+F7		
	Apply Code Changes			
	Stack			▶
	Toggle Breakpoint	Ctrl+F8		
	New Breakpoint...	Ctrl+Shift+F8		
	New Watch...	Ctrl+Shift+F7		
	Evaluate Expression...	Ctrl+F9		

Rysunek 17. Klawisze skrótu debugowania