Wykład VIII

Java - środowisko GUI
Menedżer układu graficznego komponentów

Wszystkie komponenty, które zostały omówione były rozmieszczane przy wyłączonym menedżerze komponentów graficznych za pomocą instrukcji

```
this.setLayout(null);
```

W takim przypadku będziemy jednak musieli sami określać kształt i położenie każdego komponentu za pomocą definiowanej przez klasę Component metody setBounds().

Korzystanie z ręcznego pozycjonowania kontrolek nie jest najlepszym rozwiązaniem z następujących powodów:

- rozmieszczanie dużej liczby kontrolek jest dosyć kłopotliwe i czasochłonne
- informacje na temat szerokości i wysokości kontrolek nie zawsze są dostępne na etapie ich pozycjonowania, np. dlatego, że odpowiednie komponenty programu nie są jeszcze gotowe.

Każdy obiekt klasy Container jest związany z jakimś menedżerem układu graficznego. Menedżer układu jest egzemplarzem dowolnej klasy, która implementuje interfejs LayoutManager. Do ustawiania odpowiednich menedżerów układu służy metoda setLayout(). Jeśli w naszym programie nie wywołamy tej metody, zostanie użyty domyślny menedżer układu.
Ogólna postać metody setLayout() jest następująca:
void setLayout(LayoutManager *layoutObj*)
Parametr *layoutObj* jest w tym przypadku referencją do docelowego menedżera układu graficznego.

Każdy z dostępnych menedżerów układu graficznego utrzymuje listę nazw obsługiwanych komponentów. Menedżer układu jest informowany za każdym razem, gdy dodajemy komponent do odpowiedniego kontenera. Za każdym razem, gdy rozmiar kontenera wymaga zmiany, sprawdzane są minimalne i preferowane wymiary menedżera układu odpowiednio za pośrednictwem metod minimumLayoutSize() oraz preferredLayoutSize().
Każdy komponent, którego położenie jest kontrolowane przez menedżera układu, dodatkowo definiuje metody getPreferredSize() i getMinimumSize(). Metody te zwracają odpowiednio preferowany i minimalny rozmiar wymagany do wyświetlenia danego komponentu. Menedżer układu stara się uwzględniać te wartości przy jednoczesnym zachowaniu integralności realizowanej polityki pozycjonowania. Możemy przykryć te metody w kodzie kontrolek będących podklasami istniejących klas komponentów.

Język Java zawiera kilka predefiniowanych klas LayoutManager wybrane klasy to.

FlowLayout

FlowLayout jest domyślnym menedżerem układu graficznego.
Klasa FlowLayout implementuje bardzo prosty styl rozmieszczania kontrolek.
Komponenty są układane od lewego górnego narożnika, od lewej do prawej strony i od

góry do dołu. Kiedy okazuje się, że bieżący wiersz nie może pomieścić kolejnych komponentów, bieżący i następne komponenty są umieszczane w nowym wierszu.
Konstruktory definiowane przez klasę FlowLayout mają następującą postać:
FlowLayout()
FlowLayout(int *how*)
FlowLayout(int *how*, int *horz*, int *vert*)
Pierwsza wersja konstruktora tworzy domyślny układ graficzny, który umieszcza wszystkie komponenty jak najbliżej środka kontenera (w poziomie)
Druga wersja dodatkowo umożliwia określenie sposobu wyrównywania kolejnych wierszy komponentów. Za pomocą parametru *how* możemy przekazać jedną z następujących wartości stałych:
FlowLayout.LEFT - do lewej krawędzi
FlowLayout.CENTER - do środka
FlowLayout.RIGHT - prawej krawędzi
FlowLayout.LEADING - do górnej krawędzi
FlowLayout.TRAILING - do dolnej krawędzi

Trzecia wersja konstruktora umożliwia określanie ilości wolnej przestrzeni dzielącej komponenty w pionie i w poziomie (odstępy przekazujemy za pośrednictwem odpowiednio parametrów *vert* i *horz*).

Przykład kodu źródłowego z zastosowaniem menedżera układu graficznego:

```
package kontrolki;
// Demonstruje użycie układu z wyrównywaniem do lewej krawędzi.
import java.awt.*;
public class FLayout extends Frame{
public static void main(String argv[]){
      FLayout fa=new FLayout();
            fa.setLayout(new FlowLayout(FlowLayout.LEFT));
            fa.setSize(400,300);
            fa.setVisible(true);
    }
FLayout(){
                add(new Button("Pierwszy"));
                add(new Button("Drugi"));
                add(new Button("Trzeci"));
                add(new Button("Czwarty"));
                add(new Button("Piąty"));
                add(new Button("Szósty"));
                add(new Button("Siódmy"));
                add(new Button("Ósmy"));
                add(new Button("Dziewiąty"));
                add(new Button("Dziesiąty"));
        }//Koniec konstruktora
}//Koniec aplikacji
```

BorderLayout

W klasie BorderLayout okno jest podzielone na cztery wąskie komponenty o stałej szerokości, które przylegają do krawędzi, i jeden duży obszar w środku okna (otoczony ze wszystkich stron tymi czterema komponentami). Cztery strony tego układu graficznego zwyczajowo nazywa się północą, południem, wschodem i zachodem, natomiast obszar środkowy jest nazywany centrum.
Klasa ma dwa konstruktory klasy BorderLayout:
BorderLayout()
BorderLayout(int *horz*, int *vert*)
Pierwsza wersja konstruktora tworzy domyślny układ typu BorderLayout. Druga umożliwia dodatkowe określenie odstępu dzielącego komponenty w poziomie i pionie za pomocą odpowiednio parametrów *horz* i *vert*.
Klasa BorderLayout definiuje następujące stałe reprezentujące pięć regionów okna:

- BorderLayout.CENTER
- BorderLayout.SOUTH
- BorderLayout.EAST
- BorderLayout.WEST
- BorderLayout.NORTH

Stałe są używane w metodzie add() klasy Container.
Składnia jest następująca:
void add(Component *compRef*, Object *region*)
Parametr *compRef* w tym przypadku reprezentuje dodawany komponent, natomiast parametr *region* określa miejsce (obszar), w którym ten komponent jest umieszczany.
Przykład kodu źródłowego z zastosowaniem menedżera układu graficznego:

```
package kontrolki;
// Demonstruje użycie układu z dziesięcioma elementami.
import java.awt.*;
public class BLayout extends Frame{
public static void main(String argv[]){
      BLayout fa=new BLayout();
            fa.setSize(400,400);
            fa.setVisible(true);
    }
BLayout(){
       //Ustawiony menedżer okna BorderLayout jako menedżer domyślny
                  setLayout(new BorderLayout());
                add(new Button("Pierwszy"),BorderLayout.NORTH);
                add(new Button("Drugi"),BorderLayout.NORTH);
                add(new Button("Trzeci"),BorderLayout.SOUTH);
               add(new Button("Czwarty"),BorderLayout.SOUTH);
               add(new Button("Piąty"),BorderLayout.EAST);
               add(new Button("Szósty"),BorderLayout.EAST);
               add(new Button("Siódmy"),BorderLayout.WEST);
               add(new Button("Ósmy"),BorderLayout.WEST);
                add(new Button("Dziewiąty"),BorderLayout.CENTER);
               add(new Button("Dziesiąty"), BorderLayout.CENTER);
        }//Koniec konstruktora
}//Koniec aplikacji
```

GridLayout

Menedżer układu graficznego GridLayout ustawia komponenty w postaci tabeli. Definicja instancji klasy GridLayout wymaga podania liczby wierszy i kolumn tabeli.
Definiowane są trzy typy konstruktorów Składnię konstruktorów definiowanych przez tę klasę przedstawiono poniżej:
GridLayout()
GridLayout(int *numRows*, int *numColumns*)
GridLayout(int *numRows*, int *numColumns*, int *horz*, int *vert*)
Pierwsza wersja konstruktora tworzy układ siatki złożony z pojedynczej kolumny.
Druga wersja tworzy układ siatki z określoną liczbą wierszy i kolumn.
Trzecia postać konstruktora umożliwia określanie poziomych i pionowych odstępów dzielących komponenty (odpowiednio za pośrednictwem parametrów *horz* i *vert*).
Zarówno parametr *numRows*, jak i parametr *numColumns* może mieć wartość zerową. Przekazanie tej wartości w argumencie *numRows* spowoduje utworzenie kolumn nieograniczonej szerokości, natomiast przekazanie zera za pośrednictwem parametru *numColumns* spowoduje utworzenie wierszy nieograniczonej szerokości.
Przedstawiony poniżej przykładowy program tworzy siatkę 4×3 i wypełnia ją dziesięcioma przyciskami.

Przykład kodu źródłowego:

```
package kontrolki;
// Demonstruje użycie układu z pięcioma elementami.
import java.awt.*;
public class GLayout extends Frame{

GLayout(){
        //Ustawiony menedżer okna BorderLayout
             setLayout(new GridLayout(4,3,10,10));

               add(new Button("Pierwszy"));
                add(new Button("Drugi"));
                add(new Button("Trzeci"));
               add(new Button("Czwarty"));
               add(new Button("Piąty"));
               add(new Button("Szósty"));
               add(new Button("Siódmy"));
               add(new Button("Ósmy"));
                add(new Button("Dziewiąty"));
               add(new Button("Dziesiąty"));
        }//Koniec konstruktora
public static void main(String argv[]){
      GLayout fa=new GLayout();

            fa.setSize(400,500);
            fa.setVisible(true);
    }
}//Koniec aplikacji
```

Menu i paski menu

Okno zwykle zawiera własny pasek menu. Taki pasek wyświetla listę opcji dostępnych na najwyższym poziomie hierarchii. Każda z tych opcji jest powiązana z odpowiednim menu rozwijanym. Koncepcja menu i paska menu jest implementowana przez następujące klasy: MenuBar, Menu oraz MenuItem.
Pasek menu (obiekt klasy MenuBar) zawiera jeden lub więcej obiektów klasy Menu. Każdy taki obiekt zawiera listę obiektów klasy MenuItem, z kolei każdy obiekt klasy MenuItem reprezentuje coś, co może być wybierane przez użytkownika.
Menu jest podklasą klasy MenuItem, możliwe jest tworzenie hierarchii zagnieżdżonych podmenu.
Istnieje także możliwość umieszczania w menu elementów z opcją zaznaczania. Tego typu elementy mają postać obiektów klasy CheckboxMenuItem i jeśli zostaną wybrane przez użytkownika, wyświetlają obok swoich etykiet znaki zaznaczenia.
Aby utworzyć nowy pasek menu, musimy w pierwszej kolejności utworzyć egzemplarz klasy MenuBar. Klasa ta definiuje tylko jeden konstruktor domyślny.
Kolejnym krokiem jest utworzenie egzemplarzy klasy Menu, które będą definiowały opcje wyświetlane na pasku. Klasa Menu definiuje trzy konstruktory, przedstawione poniżej:
Menu() throws HeadlessException
Menu(String *optionName*) throws HeadlessException
Menu(String *optionName*, boolean *removable*) throws HeadlessException

Za pośrednictwem parametru *optionName* możemy określić nazwę tworzonego menu. Jeśli w argumencie *removable* przekażemy wartość true, użytkownik będzie miał możliwość usuwania i przekształcania menu w swobodne okno. W przeciwnym przypadku menu będzie trwale związane z odpowiednim paskiem. (Funkcjonowanie menu z możliwością usuwania i przekształcania w swobodne okno jest uzależnione od stosowanej implementacji).
Pierwsza wersja konstruktora tworzy puste menu.
Pojedyncze elementy menu mają postać obiektów klasy MenuItem. Klasa ta definiuje następujące konstruktory:
MenuItem() throws HeadlessException
MenuItem(String itemName) throws HeadlessException
MenuItem(String itemName, MenuShortcut keyAccel) throws HeadlessException
Parametr itemName reprezentuje nazwę tworzonego elementu menu, natomiast parametr keyAccel definiuje skrót klawiszowy dla tego elementu.
Za pomocą metody setEnabled() możemy wyłączać lub włączać elementy menu. Ogólna postać tej metody jest następująca:
void setEnabled(boolean enabledFlag)
Jeśli za pośrednictwem parametru enabledFlag przekażemy wartość true, element menu zostanie włączony.
Możemy w prosty sposób określać status elementów — wystarczy wywołać metodę isEnabled(). Składnia tej metody przedstawiono poniżej:
boolean isEnabled()
Metoda isEnabled() zwraca wartość true, jeśli dany element menu jest włączony. W przeciwnym przypadku metoda zwraca wartość false.
Za pomocą metody setLabel() możemy w dowolnym momencie zmieniać nazwy elementów menu. Do uzyskiwania bieżącej nazwy elementu służy metoda getLabel(). Składnia obu metod są następujące:
void setLabel(String newName)
String getLabel()
Parametr newName reprezentuje docelową nazwę wywołującego elementu menu. Metoda getLabel() zwraca bieżącą nazwę tego elementu.

Do tworzenia elementów menu z możliwością ich zaznaczania służy podklasa klasy MenuItem nazwana CheckboxMenuItem.
Trzy konstruktory definiowane przez tę klasę przedstawiono poniżej:
CheckboxMenuItem() throws HeadlessException
CheckboxMenuItem(String itemName) throws HeadlessException
CheckboxMenuItem(String itemName, boolean on) throws HeadlessException
Parametr itemName reprezentuje w tym przypadku wyświetlaną w menu nazwę elementu.
Elementy oferujące możliwość zaznaczania działają jak przełączniki. Za każdym razem, gdy użytkownik zaznacza lub usuwa zaznaczenie takiego elementu, zmienia się jego stan.
Pierwsze dwie wersje konstruktora tworzą element z początkowym zaznaczeniem. Także w sytuacji, gdy w wywołaniu trzeciej wersji przekażemy wartość true za pośrednictwem parametru on, nowy element będzie zaznaczony.
W przeciwnym przypadku nowo utworzony element początkowo nie będzie zaznaczony.
Do odczytywania stanu takiego elementu służy metoda getState(). Możemy także ustawiać stan tego typu elementów z poziomu naszego programu — wystarczy w tym celu wywołać metodę setState(). Składnia jest następująca:
boolean getState()
void setState(boolean checked)
Jeśli element wywołujący jest zaznaczony, metoda getState() zwróci wartość true. W przeciwnym przypadku ta sama metoda zwróci wartość false. Aby zaznaczyć element, należy przekazać wartość true w jedynym argumencie metody setState(). Aby usunąć zaznaczenie elementu, musimy przekazać wartość false.
Kiedy już będziemy dysponowali gotowym elementem menu, musimy go dodać do obiektu klasy Menu za pomocą metody add(), której ogólną postać przedstawiono poniżej:
MenuItem add(MenuItem item)
Parametr *item* reprezentuje dodawany element menu. Elementy są dodawane do menu w kolejności wywołań metody add(). Metoda ta dodatkowo zwraca obiekt *item*.
Po dodaniu do obiektu klasy Menu wszystkich elementów możemy ten obiekt dodać do paska menu, wywołując następującą wersję definiowanej przez klasę MenuBar metody add():
Menu add(Menu *menu*)
Parametr *menu* reprezentuje w tym przypadku menu dodawane do paska. Metoda add() dodatkowo zwraca obiekt *menu*.

Menu generują zdarzenia tylko wtedy, gdy element typu MenuItem lub CheckboxMenuItem jest wybierany (zaznaczany) przez użytkownika. Warto pamiętać, że nie są generowane zdarzenia np. w momencie kliknięcia i rozwinięcia menu przez użytkownika. Za każdym razem, gdy użytkownik wybiera element menu, generowany jest obiekt klasy ActionEvent. Za każdym razem, gdy użytkownik zaznacza lub usuwa zaznaczenie pola wyboru przy elemencie menu, generowany jest obiekt klasy ItemEvent. Oznacza to, że obsługa zdarzeń związanych z kontrolkami menu wymaga od nas zaimplementowania zarówno interfejsu ActionListener, jak i interfejsu ItemListener.
Definiowana przez klasę ItemEvent metoda getItem() zwraca referencję do elementu menu, który wygenerował dane zdarzenie. Ogólna postać tej metody jest następująca:
Object getItem()

Przykładowy program:

```
package kontrolki;

//Demonstruje użycie menu.
import java.awt.*;
import java.awt.event.*;

//Tworzy obiekt klasy Frame.
```

```
class Menu_Okno extends Frame {

String msg = "";
CheckboxMenuItem debug, test;
Menu_Okno() {

//tworzy pasek menu i dodaje go do okna typu Frame
MenuBar Menum_pasek = new MenuBar();
setMenuBar(Menum_pasek);
//tworzy elementy menu
Menu file = new Menu("Plik");
MenuItem item1, item2, item3, item4, item5;
file.add(item1 = new MenuItem("Nowy..."));
file.add(item2 = new MenuItem("Otwórz..."));
file.add(item3 = new MenuItem("Zamknij"));
file.add(item4 = new MenuItem("-"));
file.add(item5 = new MenuItem("Zakończ..."));
Menum_pasek.add(file);
Menu edit = new Menu("Edycja");
MenuItem item6, item7, item8, item9;
edit.add(item6 = new MenuItem("Wytnij"));
edit.add(item7 = new MenuItem("Kopiuj"));
edit.add(item8 = new MenuItem("Wklej"));
edit.add(item9 = new MenuItem("-"));
Menu sub = new Menu("Specjalne");
MenuItem item10, item11, item12;
sub.add(item10 = new MenuItem("Pierwszy"));
sub.add(item11 = new MenuItem("Drugi"));
sub.add(item12 = new MenuItem("Trzeci"));
edit.add(sub);
// tworzy elementy menu z możliwością zaznaczania
debug = new CheckboxMenuItem("Tryb usuwania błędów");
edit.add(debug);
test = new CheckboxMenuItem("Tryb testowania");
edit.add(test);
Menum_pasek.add(edit);
// tworzy obiekt obsługujący zdarzenia typu ActionEvent i ItemEvent
MyMenuHandler handler = new MyMenuHandler(this);
// rejestruje obiekt otrzymujący informacje o tego typu zdarzeniach
item1.addActionListener(handler);
item2.addActionListener(handler);
item3.addActionListener(handler);
item4.addActionListener(handler);
item5.addActionListener(handler);
item6.addActionListener(handler);
item7.addActionListener(handler);
item8.addActionListener(handler);
item9.addActionListener(handler);
item10.addActionListener(handler);
item11.addActionListener(handler);
item12.addActionListener(handler);
debug.addItemListener(handler);
test.addItemListener(handler);
// tworzy obiekt obsługujący zdarzenia typu WindowEvent
MyWindowAdapter adapter = new MyWindowAdapter(this);
// rejestruje ten obiekt otrzymujący informacje o tego typu zdarzeniach
addWindowListener(adapter);
}
public static void main(String[] args) {
      Menu_Okno okienko = new Menu_Okno();
      okienko.setSize(new Dimension(600, 800));
      //okienko.setSize(300, 200);
      okienko.setTitle("Menu  w   AWT");
      okienko.setVisible(true);
```

```
}
public void paint(Graphics g) {
g.drawString(msg, 10, 200);
if(debug.getState())
g.drawString("Usuwanie błędów jest włączone.", 10, 220);
else
g.drawString("Usuwanie błędów jest wyłączone.", 10, 220);
if(test.getState())
g.drawString("Testowanie jest włączone.", 10, 240);
else
g.drawString("Testowanie jest wyłączone.", 10, 240);
}
}
class MyWindowAdapter extends WindowAdapter {
Menu_Okno menuFrame;
public MyWindowAdapter(Menu_Okno menuFrame) {
this.menuFrame = menuFrame;
}
public void windowClosing(WindowEvent we) {
menuFrame.setVisible(false);
}
}
class MyMenuHandler implements ActionListener, ItemListener {
Menu_Okno menuFrame;
public MyMenuHandler(Menu_Okno menuFrame) {
this.menuFrame = menuFrame;
}
// obsługuje zdarzenia typu ActionEvent
public void actionPerformed(ActionEvent ae) {
String msg = "Wybrałeś opcję ";
String arg = (String)ae.getActionCommand();
if(arg.equals("Nowy..."))
msg += "Nowy.";
else if(arg.equals("Otwórz..."))
msg += "Otwórz.";
else if(arg.equals("Zamknij"))
msg += "Zamknij.";
else if(arg.equals("Zakończ..."))
msg += "Zakończ.";
else if(arg.equals("Edycja"))
msg += "Edycja.";
else if(arg.equals("Wytnij"))
msg += "Wytnij.";
else if(arg.equals("Kopiuj"))
msg += "Kopiuj.";
else if(arg.equals("Wklej"))
msg += "Wklej.";
else if(arg.equals("Pierwszy"))
msg += "Pierwszy.";
else if(arg.equals("Drugi"))
msg += "Drugi.";
else if(arg.equals("Trzeci"))
msg += "Trzeci.";
else if(arg.equals("Tryb usuwania błędów"))
msg += "Tryb usuwania błędów.";
else if(arg.equals("Tryb testowania"))
msg += "Tryb testowania.";
menuFrame.msg = msg;
menuFrame.repaint();
}
// obsługuje zdarzenia typu ItemEvent
public void itemStateChanged(ItemEvent ie) {
menuFrame.repaint();
}
```

```
}
```

FileDialog

Język programowania Java udostępnia wbudowane okno dialogowe, za pośrednictwem którego użytkownik może wskazywać wybrane przez siebie pliki. Aby przygotować takie okno, należy utworzyć egzemplarz klasy FileDialog. W efekcie odpowiednie okno dialogowe od razu zostanie wyświetlone na ekranie. Jest to zwykle standardowe okno z plikami obsługiwane przez system operacyjny. Poniżej przedstawiono trzy konstruktory definiowane przez klasę FileDialog:
FileDialog(Frame *parent*, String *boxName*)
FileDialog(Frame *parent*, String *boxName*, int *how*)
FileDialog(Frame *parent*)
Parametr *parent* reprezentuje w tym przypadku okno macierzyste tworzonego okna dialogowego, natomiast parametr *boxName* jest nazwą wyświetlaną w pasku tytułu nowego okna. Jeśli za pośrednictwem parametru *how* przekażemy stałą FileDialog.LOAD, wybrany w oknie dialogowym plik zostanie otwarty do odczytu. Jeśli przekażemy w tym argumencie stałą FileDialog.SAVE, użytkownik będzie mógł wskazać plik do zapisania. Trzeci konstruktor tworzy okno dialogowe umożliwiające wyłącznie wybór pliku do odczytu.
Klasa FileDialog definiuje metody, za pomocą których możemy określać nazwę i ścieżkę do pliku wybranego przez użytkownika. Dwa przykłady takich metod przedstawiono poniżej:
String getDirectory()
String getFile()
Wymienione metody zwracają odpowiednio katalog i nazwę wybranego pliku.
Przykładowy program:

```
package kontrolki;
/* Demonstruje użycie okna dialogowego plików.

*/
import java.awt.*;
import java.awt.event.*;
//Tworzy podklasę klasy Frame.
class Okno_Dialogowe extends Frame {
      Okno_Dialogowe(String title) {
super(title);

//Usuwa okno po zamknięciu.
addWindowListener(new WindowAdapter() {
public void windowClosing(WindowEvent we) { System.exit(0);}
});
}
}
//Demonstruje działanie klasy FileDialog.
class Okno_Dialogowe_Test {
public static void main(String args[]) {
//Tworzy ramkę zawierającą okno dialogowe.
Frame f = new Okno_Dialogowe("Przykład okna dialogowego plików");
f.setVisible(true);
f.setSize(400, 400);
FileDialog fd = new FileDialog(f, "Okno dialogowe plików");
fd.setVisible(true);
}
}
```

Opracowano na podstawie:
HERBERT SCHILDT - „Java. Kompendium programisty. Wydanie IX” Wydawnictwo Helion.