Wykład VII

Java - środowisko GUI
Stosowanie kontrolek biblioteki AWT

Biblioteka AWT (Abstract Window Toolkit) zawiera szereg dodatkowych narzędzi do tworzenia GUI są to standardowe kontrolki oraz menedżer układów graficznych. Elementem niezbędnym przy tworzeniu interfejsu użytkownika jest poznanie obsługi zdarzeń związanych z kontrolkami.

Kontrolki są komponentami, które umożliwiają użytkownikowi interakcję z aplikacją na wiele różnych sposobów — jedną z najczęściej stosowanych kontrolek jest przycisk, pole tekstowe, lista, pola przycisków wyboru i opcji.
Menedżer układu graficznego automatycznie pozycjonuje komponenty w ramach kontenera. Oznacza to, że wygląd okna zależy od kombinacji umieszczonych na nim kontrolek oraz użytego menedżera układu, który odpowiada za ich właściwe pozycjonowanie.
Poznamy następujące typy kontrolek:
etykiety,

- przyciski,
- pola wyboru,
- listy rozwijane,
- listy,
- paski przewijania,
- pola tekstowe.

Wszystkie wymienione powyżej kontrolki są podklasami klasy Component. Choć powyższy zbiór kontrolek nie jest szczególnie bogaty, to jednak z powodzeniem wystarcza do tworzenia prostych aplikacji. Znacznie bogatszy zestaw kontrolek jest dostępny w pakietach Swing, jak i JavaFX

Dodawanie i usuwanie kontrolek.

Dodanie kontrolki do istniejącego okna wymaga następujących czynności:

- utworzyć egzemplarz docelowej kontrolki
- dodać ją do okna za pomocą definiowanej przez klasę Container metody add().

Metoda add() występuje w wielu wersjach. W pierwszej części tego rozdziału będziemy wykorzystywać następującą jej postać:
Component add(Component *compRef*)
Parametr *compRef* jest w tym przypadku egzemplarzem kontrolki, którą chcemy dodać do okna.
Metoda zwraca referencję do tego obiektu. Kiedy już kontrolka zostanie dodana do okna, automatycznie będzie widoczna w momencie wyświetlenia okna macierzystego. Jej położenie będzie zależne od przyjętego

Usuwanie kontrolek

W niektórych sytuacjach konieczne jest usunięcie kontrolki z okna, kiedy uznamy, że nie jest ona potrzebna. W tym celu wystarczy wywołać definiowaną przez klasę Container metodę remove(). Ogólna postać metody remove() jest następująca:
void remove(Component *compRef*)
Tym razem parametr *compRef* jest kontrolką, którą chcemy usunąć. Wywołując metodę removeAll(), możemy w jednym kroku usunąć wszystkie kontrolki.

Listy rozwijalne - kontrolki

Do tworzenia **list rozwijalnych** używana jest klasa Choice, użytkownik dokonuje wyboru elementu listy za pomocą kursora myszki lub klawiatury. Klikniecie na element pokazuje pełna listę elementów.
Klasa Choice definiuje jeden domyślny konstruktor, za pomocą którego możemy tworzyć puste listy. Aby dodać nową opcję do listy rozwijanej, należy wywołać metodę add(). Ogólna postać tej metody jest następująca:
void add(String *name*)
Parametr *name* reprezentuje w tym przypadku nazwę dodawanego elementu listy. Elementy są dodawane do listy w kolejności odpowiednich wywołań metody add().
Aby określić, który element listy jest aktualnie zaznaczony, możemy wywołać albo metodę getSelectedItem(), albo metodę getSelectedIndex(). Składnia obu metod przedstawiono poniżej:
String getSelectedItem()
int getSelectedIndex()
Metoda getSelectedItem() zwraca ciąg znaków zawierający nazwę danego elementu, natomiast metoda getSelectedIndex() zwraca liczbę całkowitą będącą indeksem tego elementu. Pierwszy element jest składowany pod indeksem równym 0. Domyślnie zaznaczonym elementem listy jest ten, który został do niej dodany jako pierwszy. Elementy listy są prezentowane w taki sposób jak były dodawane.
Aby uzyskać liczbę elementów listy, należy wywołać metodę getItemCount(). Możemy ustawić aktualnie zaznaczony (wybrany) element za pomocą metody select(), która pobiera w formie argumentu albo numer indeksu elementu (liczony od zera), albo tekst reprezentujący nazwę odpowiedniego elementu na liście.
Składnia metody getItemCount() i obu wersji metody select() przedstawiono poniżej:
int getItemCount()
void select(int *index*)
void select(String *name*)
Znając indeks elementu listy, możemy w prosty sposób uzyskać jego nazwę — wystarczy wywołać metodę getItem(), której ogólną postać przedstawiono poniżej:
String getItem(int *index*)
Parametr *index* reprezentuje w tym przypadku indeks interesującego nas elementu listy.

Obsługa zdarzeń list rozwijanych

Użytkownik wybierając dowolny element z listy, automatycznie generuje zdarzenie typu ItemEvent. Obiekt tego zdarzenia jest przekazywany wszystkim obiektom nasłuchującym, które zarejestrowały swoje zainteresowanie sygnałami o tego typu zdarzeniach generowanych przez dany komponent. Każdy z tych obiektów nasłuchujących musi implementować interfejs ItemListener. Interfejs ten definiuje metodę itemStateChanged(), która otrzymuje reprezentujący zdarzenie obiekt klasy ItemEvent w formie swojego argumentu.

Przykład kodu źródłowego z zastosowaniem obiektów listy

```
package kontrolki;
//Demonstruje użycie list rozwijanych.
import java.awt.*;
import java.awt.event.*;
public class Listy extends Frame implements ItemListener {
Choice system_op, przegladarka;
String msg = "";
public Listy() {
system_op = new Choice();
przegladarka = new Choice();
// dodaje elementy do listy systemów operacyjnych
system_op.add("Windows");
system_op.add("Android");
system_op.add("Solaris");
system_op.add("MacOS");
// dodaje elementy do listy przeglądarek
przegladarka.add("Internet Explorer");
przegladarka.add("Firefox");
przegladarka.add("Chrome");

//Wyłączenie menedżera rozłożenia elementów w oknie programu
this.setLayout(null);
// dodaje listy rozwijane do okna
add(system_op);
add(przegladarka);
//położenie list w oknie
             system_op.setBounds(50,40,150,20);
             przegladarka.setBounds(300,40,150,20);

// rejestruje obiekt otrzymujący informacje o zdarzeniach typu ItemEvent
system_op.addItemListener(this);
przegladarka.addItemListener(this);
}
//Tworzy okno.
public static void main(String args[])
{
Listy okienko = new Listy();
okienko.setSize(500, 300);
//okienko.setSize(new Dimension(300, 200));
okienko.setTitle("Listy biblioteki  AWT");

okienko.setVisible(true);
}
public void itemStateChanged(ItemEvent ie) {
repaint();
}
// Wyświetla aktualnie wybrane opcje.
public void paint(Graphics g) {
msg = "Wybrany system operacyjny: ";
msg += system_op.getSelectedItem();
g.drawString(msg, 60, 120);
msg = "Wybrana przeglądarka: ";
msg += przegladarka.getSelectedItem();
g.drawString(msg, 60, 180);
}
}
```

Listy - kontrolki

Obiekt List jest listą umożliwiającą obsługę przewijanych list z możliwościa wielokrotnego zaznaczenia elementów listy. Klasa Choice, wyświetlają tylko jeden wybrany element listy , obiekty klasy List mogą być konstruowane w taki sposób, aby wyświetlać w widocznym oknie dowolną liczbę dostępnych opcji.
Klasa List definiuje następujące konstruktory:
List() throws HeadlessException
List(int *numRows*) throws HeadlessException
List(int *numRows*, boolean *multipleSelect*) throws HeadlessException
Pierwsza wersja konstruktora tworzy kontrolkę typu List, która umożliwia użytkownikowi jednoczesne zaznaczanie tylko pojedynczych elementów. W drugiej wersji parametr *numRows* określa liczbę elementów listy, które zawsze będą widoczne (jeśli łączna liczba elementów listy będzie większa, możliwe będzie jej przewijanie). Jeśli pobierany przez trzecią wersję konstruktora parametr *multipleSelect* zawiera wartość true, użytkownik będzie mógł jednocześnie zaznaczać po dwa i więcej elementów. Jeśli przekażemy w tym argumencie wartość false, możliwe będzie zaznaczanie tylko jednego elementu.
Aby dodać do listy nowy element, należy wywołać jedną z dwóch dostępnych wersji metody add().
Sygnatury obu wersji przedstawiono poniżej:
void add(String *name*)
void add(String *name*, int *index*)

Parametr *name* reprezentuje w tym przypadku nazwę nowego elementu listy. Pierwsza wersja metody add() dodaje elementy na koniec danej listy; druga wersja wstawia elementy na pozycjach określonych za pomocą parametru *index*. Indeksowanie listy rozpoczyna się od zera. Aby dodać nowy element na koniec listy, wystarczy w argumencie *index* użyć wartości –1.
W przypadku list umożliwiających zaznaczanie tylko pojedynczych elementów możemy określić aktualnie wybraną opcję albo za pomocą metody getSelectedItem(), albo metody getSelectedIndex().
Składnia obu metod przedstawiono poniżej:
String getSelectedItem()
int getSelectedIndex()
Metoda getSelectedItem() zwraca obiekt String zawierający nazwę danego elementu. Jeśli zaznaczony jest więcej niż jeden element lub jeśli użytkownik nie zaznaczył jeszcze żadnego elementu, metoda zwraca wartość null. Metoda getSelectedIndex() zwraca indeks tego elementu. Pierwszy element jest składowany pod indeksem równym 0. Jeśli zaznaczony jest więcej niż jeden element lub jeśli użytkownik nie zaznaczył jeszcze żadnego elementu, metoda zwraca wartość –1.
W przypadku list dopuszczających możliwość zaznaczania wielu elementów do określania aktualnie wybranych opcji powinniśmy używać albo metody getSelectedItems(), albo metody getSelectedIndexes();
Składnie tych metod są następujące:
String[] getSelectedItems()
int[] getSelectedIndexes()
Metoda getSelectedItems() zwraca tablicę zawierającą nazwy aktualnie zaznaczonych elementów listy, natomiast metoda getSelectedIndexes() zwraca tablicę z indeksami aktualnie zaznaczonych elementów.
Aby uzyskać łączną liczbę elementów listy, należy wywołać metodę getItemCount().
Możemy zaznaczać elementy listy za pomocą metody select(), której argumentem jest liczba całkowita (liczona od zera) indeks elementu przeznaczonego do zaznaczenia.

Składnia obu metod przedstawiono poniżej:
int getItemCount()
void select(int *index*)
Znając indeks elementu listy, możemy w prosty sposób uzyskać jego nazwę — wystarczy wywołać metodę getItem(), której ogólną postać przedstawiono poniżej:
String getItem(int *index*)
Parametr *index* reprezentuje w tym przypadku indeks interesującego nas elementu listy.

Zdarzenia generowane przez obiekt listy

W odróżnieniu od obiektu Choice zdarzenia generowane przez kontrolki typu List wymagają zaimplementowania interfejsu ActionListener. Za każdym razem, gdy użytkownik dwukrotnie klika element takiej listy, generowany jest obiekt klasy ActionEvent. Do odczytywania nazwy właśnie zaznaczonego elementu możemy używać definiowanej przez tę klasę metody getActionCommand(). Dodatkowo za każdym razem, gdy użytkownik zaznacza lub usuwa zaznaczenie elementu, stosując pojedyncze kliknięcie, generowany jest obiekt klasy ItemEvent. Definiowana przez tę klasę metoda getStateChange() zapewnia możliwość sprawdzenia, czy powodem wygenerowania danego zdarzenia było zaznaczenie, czy usunięcie zaznaczenia elementu. W takim przypadku metoda getItemSelectable() zwraca referencję do obiektu, który wywołał to zdarzenie.

Przykład kodu źródłowego z zastosowaniem obiektów listy

```
//Demonstruje użycie list rozwijanych.
import java.awt.*;
import java.awt.event.*;
public class Listy extends Frame implements ActionListener //, ItemListener
{
List system_op, przegladarka;
String msg = "";
public Listy() {
        system_op = new List(4, true);
przegladarka = new List(4,false);
// dodaje elementy do listy systemów operacyjnych
system_op.add("Windows");
system_op.add("Android");
system_op.add("Solaris");
system_op.add("MacOS");
// dodaje elementy do listy przeglądarek
przegladarka.add("Internet Explorer");
przegladarka.add("Firefox");
przegladarka.add("Chrome");
//dodaje elementy do listy przeglądarek
przegladarka.select(1);

//Wyłączenie menedżera rozłożenia elementów w oknie programu
this.setLayout(null);
// dodaje listy rozwijane do okna
add(system_op);
add(przegladarka);
//położenie list w oknie
             system_op.setBounds(50,40,150,100);
             przegladarka.setBounds(300,40,150,100);
// rejestruje obiekt otrzymujący informacje o zdarzeniach typu ActionEvent
system_op.addActionListener(this);
przegladarka.addActionListener(this);

// rejestruje obiekt otrzymujący informacje o zdarzeniach typu ItemEvent
//system_op.addItemListener(this);
// przegladarka.addItemListener(this);
}
//Tworzy okno.
```

```
public static void main(String args[])
{
Listy okienko = new Listy();
okienko.setSize(500, 300);
//okienko.setTitle(new Dimension(300, 200));
okienko.setTitle("Listy biblioteki  AWT");
okienko.setVisible(true);
}
public void itemStateChanged(ItemEvent ie) {
// repaint();
}
// Wyświetla aktualnie wybrane opcje.
public void paint(Graphics g) {
	int idx[];
	msg = "Wybrany system operacyjny: ";
	idx = system_op.getSelectedIndexes();
	for(int i=0; i<idx.length; i++)
	msg += system_op.getItem(idx[i]) + " ";
	g.drawString(msg, 60, 220);
	msg = "Wybrana przeglądarka: ";
	msg += przegladarka.getSelectedItem();
	g.drawString(msg, 60, 240);
}
@Override
public void actionPerformed(ActionEvent e) {
	repaint();

}
}
```

Pole tekstowe kontrolka typu Stosowanie kontrolek typu TextArea

Rozszerzoną wersją pola TextField jest pole tekstowe o wielu liniach można go porównać do prostego wielowierszowego edytor tekstu jest to TextArea. Poniżej przedstawiono ogólną postać pięciu konstruktorów tej klasy:
TextArea() throws HeadlessException
TextArea(int *numLines*, int *numChars*) throws HeadlessException
TextArea(String *str*) throws HeadlessException
TextArea(String *str*, int *numLines*, int *numChars*) throws HeadlessException
TextArea(String *str*, int *numLines*, int *numChars*, int *sBars*) throws HeadlessException
Parametr *numLines* reprezentuje wyrażoną w wierszach wysokość pola tekstowego, natomiast parametr *numChars* reprezentuje jego szerokość (w znakach). Początkową zawartość pola tekstowego można zdefiniować za pomocą parametru *str*. W piątej wersji konstruktora możemy dodatkowo określić, które paski przewijania chcemy umieścić w tworzonej kontrolce. Parametr *sBars* musi mieć jedną z wymienionych poniżej wartości:

- SCROLLBARS_BOTH
- SCROLLBARS_NONE
- SCROLLBARS_HORIZONTAL_ONLY
- SCROLLBARS_VERTICAL_ONLY

TextArea jest podklasą klasy TextComponent. Oznacza to, że kontrolka TextArea obsługuje opisane w poprzednim podrozdziale metody getText(), setText(), getSelectedText(), select(), isEditable() oraz setEditable().
Klasa TextArea dodaje do tego zbioru następujące trzy metody związane z edycją tekstowej zawartości kontrolki:
void append(String *str*)
void insert(String *str*, int *index*)
void replaceRange(String *str*, int *startIndex*, int *endIndex*)

Metoda append() dołącza tekst reprezentowany przez parametr *str* na koniec bieżącej zawartości wielowierszowego pola tekstowego. Metoda insert() wstawia tekst reprezentowany przez parametr *str* w miejscu wskazywanym przez określony indeks. Aby zamienić fragment istniejącego tekstu, należy wywołać metodę replaceRange(), która wymienia znaki od punktu *startIndex* do punktu *endIndex*–1 na tekst przekazany w argumencie *str*.
Wielowierszowe pola tekstowe są niemal samowystarczalne. Nasz program nie ma prawie żadnego wpływu na zarządzanie kontrolkami tego typu. Wielowierszowe pola tekstowe generują wyłącznie zdarzenia związane z uzyskaniem lub utratą aktywności klawiatury. Najczęściej nasz program jedynie odczytuje aktualną zawartość pola tekstowego w momencie, gdy dane te są potrzebne do dalszego przetwarzania.
Przykład kodu źródłowego z polem tekstowym

```
package kontrolki;
import java.awt.*;
//Demonstruje użycie komponentu TextArea.

	public class Pole_tekstowe extends Frame {

		public Pole_tekstowe() {
String val =
"Java 8 jest najnowszą wersją najczęściej używanego\n" +
"języka programowania na świecie. Bazując na bogatej\n" +
"spuściźnie innych języków, Java rozwinęła zarówno sztukę,\n" +
"jak i naukę o projektowaniu języków programowania.\n\n" +
"Jednym z powodów ciągłej popularności Javy jest jej\n" +
"stała, powolna ewolucja i rozwój. Java nigdy nie stała\n" +
"w miejscu. Zamiast tego bezustannie adaptowała się\n" +
"do sieciowego świata, podlegającego ciągłym, szybkim \n" +
"zmianom. Co więcej, Java często przewodziła rozwojowi,\n" +
"wytyczając szlaki dla innych języków programowania.";
TextArea tekst = new TextArea(val, 10, 30);
//Wyłączenie menedżera rozłożenia elementów w oknie programu
this.setLayout(null);
// położenie etykiet w oknie
tekst.setBounds(100,40,200,100);
// dodaje pole tekstowe  do okna
add(tekst);
}
public static void main(String[] args) {
		Pole_tekstowe okienko = new Pole_tekstowe();
		okienko.setSize(new Dimension(600, 800));
		//okienko.setSize(300, 200);
		okienko.setTitle("Pole tekstowe w  AWT");
		okienko.setVisible(true);
}
}
```

Paski przewijania - kontrolki

Paski przewijania służą do wyboru spośród wielu sąsiadujących wartości pomiędzy wyznaczonym minimum i maksimum. Mamy do dyspozycji poziome i pionowe paski przewijania. Pasek przewijania w praktyce składa się z wielu pojedynczych fragmentów. Na obu jego końcach wyświetlane są przyciski strzałek, które klikane przez użytkownika przesuwają bieżącą wartość paska przewijania o jedną jednostkę w kierunku zgodnym z orientacją strzałki. Bieżąca wartość paska przewijania jest wartością wyznaczaną względem jego wartości minimalnej i maksymalnej i jest graficznie reprezentowana przez **suwak** wyświetlany w ramach paska. Użytkownik może w dowolnym momencie przeciągnąć suwak paska przewijania na pożądaną pozycję. Odpowiedzią na takie działanie będzie odpowiednie dostosowanie wartości paska przewijania. Użytkownik może dodatkowo kliknąć przestrzeń tła paska przewijania po obu stronach suwaka, aby spowodować jego przesunięcie w odpowiednim kierunku o wartość większą od 1 jednostki. Takie działanie użytkownika jest zwykle traktowane jak specyficzna postać przewijania strony w górę i w dół. Paski przewijania są reprezentowane przez klasę Scrollbar.
Klasa Scrollbar definiuje następujące konstruktory:
Scrollbar() throws HeadlessException
Scrollbar(int *style*) throws HeadlessException
Scrollbar(int *style*, int *initialValue*, int *thumbSize*, int *min*, int *max*) throws HeadlessException
Pierwsza wersja konstruktora tworzy pionowy pasek przewijania. Druga i trzecia dodatkowo umożliwiają określanie orientacji nowo tworzonego paska.
Jeśli parametr *style* ma wartość Scrollbar.VERTICAL, tworzony jest pionowy pasek przewijania; jeśli natomiast parametr *style* ma wartość Scrollbar.HORIZONTAL, zostanie utworzony pasek poziomy. Trzecia wersja konstruktora umożliwia określenie początkowej wartości paska przewijania (przekazywanej za pośrednictwem parametru *initialValue*). Liczbę jednostek reprezentowanych przez suwak przekazujemy w argumencie *thumbSize*.
Parametry *min* i *max* reprezentują odpowiednio wartość minimalną i maksymalną tworzonego paska przewijania.
Jeśli do konstruowania paska przewijania użyjemy jednego z dwóch pierwszych konstruktorów, przed jego użyciem będziemy musieli dodatkowo ustawić jego parametry za pomocą metody setValues(), której ogólną postać przedstawiono poniżej:
void setValues(int *initialValue*, int *thumbSize*, int *min*, int *max*)
Parametry tej metody mają takie same znaczenia jak opisane przed momentem odpowiednie parametry trzeciej wersji konstruktora klasy Scrollbar.
Aby uzyskać bieżącą wartość paska przewijania, należy wywołać metodę getValue(), która zwraca aktualne ustawienie tego paska. Aby ustawić nową wartość paska przewijania, należy użyć metody setValue().
Składnia tych metod są następujące:
int getValue()
void setValue(int *newValue*)
Parametr *newValue* określa w tym przypadku nową wartość paska przewijania. Kiedy używamy metody setValue() do ustawienia tej wartości, suwak paska przewijania jest automatycznie przenoszony na pozycję odpowiadającą nowym ustawieniom.
Za pomocą przedstawionych poniżej metod getMinimum() i getMaximum() możemy odczytywać odpowiednio wartość minimalną i maksymalną paska przewijania.
int getMinimum()
int getMaximum()
Obie metody zwracają żądane wielkości.
Za każdym razem, gdy użytkownik przewija pasek w górę lub w dół o jeden wiersz, domyślnie wartość paska jest odpowiednio zwiększana lub zmniejszana o 1. Możemy to jednak zmienić, wywołując metodę setUnitIncrement(). Zgodnie z ustawieniami

domyślnymi operacje przewijania strony w górę i w dół odpowiednio zwiększają i zmniejszają wartość paska o 10. Także tę wartość możemy zmienić za pomocą metody setBlockIncrement().
Składnie tych metod są następujące:
void setUnitIncrement(int *newIncr*)
void setBlockIncrement(int *newIncr*)

Obsługa zdarzeń generowanych przez paski przewijania

Aby możliwe było przetwarzanie zdarzeń paska przewijania, musimy zaimplementować interfejs AdjustmentListener. Za każdym razem, gdy użytkownik wykonuje jakieś działanie na pasku przewijania, automatycznie generowany jest obiekt klasy AdjustmentEvent. Definiowana przez tę klasę metoda getAdjustmentType() umożliwia sprawdzenie, z jakiego rodzaju zdarzeniem mamy do czynienia.

Stałe reprezentujące poszczególne zdarzenia typu AdjustmentEvent

- BLOCK_DECREMENT Wygenerowano zdarzenie przewinięcia w dół całej strony
- BLOCK_INCREMENT Wygenerowano zdarzenie przewinięcia w górę całej strony
- TRACK Wygenerowano zdarzenie bezwzględnego ustawienia wartości paska przewijania
- UNIT_DECREMENT Użytkownik nacisnął przycisk przewijania wiersza w dół
- UNIT_INCREMENT Użytkownik nacisnął przycisk przewijania wiersza w górę

Przykład kodu źródłowego:

```
package kontrolki;
import java.awt.*;
import java.awt.event.*;

public class Paski_przewijania extends Frame implements AdjustmentListener,
MouseMotionListener {
	String msg = "";
	Scrollbar pionowyPP, poziomyPP;
	public Paski_przewijania() {
		int szerokosc = 600;
		int wysokosc =800;
	pionowyPP = new Scrollbar(Scrollbar.VERTICAL,0, 1, 0, wysokosc);
	pionowyPP.setPreferredSize(new Dimension(20, 100));
	poziomyPP = new Scrollbar(Scrollbar.HORIZONTAL, 0, 1, 0, szerokosc);
	poziomyPP.setPreferredSize(new Dimension(100, 20));

		add(pionowyPP);
		add(poziomyPP);
//Wyłączenie menedżera rozłożenia elementów w oknie programu
		this.setLayout(null);
		// położenie pasków przewijania  w oknie
		pionowyPP.setBounds(100,40,20,100);
		poziomyPP.setBounds(100,280,100,20);
// rejestruje obiekt otrzymujący informacje o zdarzeniach przewijania
		pionowyPP.addAdjustmentListener(this);
		poziomyPP.addAdjustmentListener(this);

		addMouseMotionListener(this);
		}
public static void main(String[] args) {
		Paski_przewijania okienko = new Paski_przewijania();
		okienko.setSize(new Dimension(600, 800));
		//okienko.setSize(300, 200);
		okienko.setTitle("Aplikacja wykorzystująca pakiet  AWT");
		okienko.setVisible(true);
	}
	// Wyświetla bieżące wartości pasków przewijania.
	public void paint(Graphics g) {
	msg = "Pionowy: " + pionowyPP.getValue();
	msg += ", Poziomy: " + poziomyPP.getValue();
	g.drawString(msg, 60, 360);
	// wyświetla bieżącą pozycję przeciąganego kursora myszy
	g.drawString("*", poziomyPP.getValue(),      pionowyPP.getValue());
	}
	@Override
	public void mouseDragged(MouseEvent e) {
		int x = e.getX();
		int y = e.getY();
		pionowyPP.setValue(y);
		poziomyPP.setValue(x);
		repaint();
	}
	@Override
	public void mouseMoved(MouseEvent e) {
	}
	@Override
	public void adjustmentValueChanged(AdjustmentEvent e) {
		repaint();

	}
}
```