Moduł 8: Interfejs użytkownika

Założenia biblioteki graficznej

Aplikacje Javy powinny być przenośne nie tylko na poziomie kodu źródłowego i binarnego, ale również w warstwie graficznej. Oprogramowanie z graficznym interfejsem użytkownika (GUI) powinno dostosowywać się do wyglądu i funkcjonowania interfejsu konkretnej platformy. W tym celu można zastosować biblioteki graficzne. Jedną z nich jest biblioteka AWT (ang. *Abstract Window Toolkit*), która pozwala na tworzenie podstawowych elementów interfejsu graficznego oraz definiowanie ich zachowania. Biblioteka AWT została przygotowana w oparciu o wybrane komponenty dostępne na platformach Windows, Motif, MacOS. Podczas uruchomienia pod abstrakcyjne komponenty graficzne "podkładane są" komponenty dostępne na danej platformie uruchomieniowej. Oznacza to, że ilość dostępnych kontrolek jest niewielka i ich wygląd pozostawia wiele do życzenia. Bogatszy zbiór komponentów jest jednak udostępniony w bibliotece Swing dostępnej w JFC (ang. *Java Foundation Classes*), gdzie komponenty są rysowane przez maszynę wirtualną.

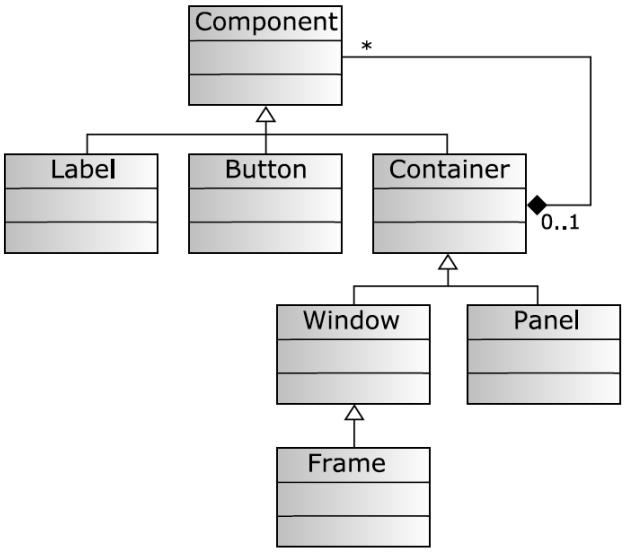
Zasady budowy interfejsu

Budowanie graficznego interfejsu użytkownika opiera się o trzy elementy:

- komponenty – wybór właściwych dla danego zastosowania zestawu komponentów (najczęściej kontroler graficznych);
- kontenery – osadzenia komponentów w założonym układzie w kontenerze;
- zdarzenia – przygotowania obsługi zdarzeń związanych z poleceniami wydawanymi przez użytkownika.

Podstawowe zależności między komponentami

Między opisanymi powyżej elementami zachodzą zależności. Dowolny obiekt typu `Component` (komponent) może być dodany do obiektu typu `Container` (kontener). Ponieważ komponent jest również kontenerem, to może on przyjmować również inne obiekty typu `Component` – istnieje więc możliwość zagnieżdżania komponentów.



Przykładowy schemat zależności pomiędzy komponentami

Komponent

Komponent jest to podstawowy typ, z którego są specjalizowane odpowiednie komponenty. Obiekt typu `Component` zawiera kilkadziesiąt metod związanych z:

- zarządzaniem położenia;
- obsługą zdarzeń;
- wyglądem graficznym.

Wymienione powyżej wskazują na to, że komponent jest odpowiedzialny za określenie właściwości elementu graficznego, jednak sam w sobie nie reprezentuje żadnego wyglądu i funkcjonowania. Przy budowie GUI korzysta się z klas potomnych odpowiadającym konkretnym składnikom graficznym, są to np. `JButton`, `JCheckbox`, `JChoice`, `JLabel`, `JList`, `JTextField`, `JTextArea`.

Kontener

Klasa `Container` służy do agregowania innych komponentów (kontenerów). Posiada szereg metod umożliwiających dodanie konkretnego komponentu. Dodatkowo, kontener może zawierać mechanizmy sterowania rozkładem komponentów. Podstawowe kontenery to:

- `JPanel` – kontener pomocniczy;
- `JFrame`, `JWindow`, `JDialog` – kontenery mające swoją reprezentację graficzną.

Ramka

Ramka (`Frame`) jest podstawowym kontenerem, który może samodzielnie zaistnieć na ekranie. Standardowa ramka jest wyposażona w:

- pasek tytułu;
- menu systemowe;

- przyciski sterujące rozmiarem okna;
- obwódkę, która umożliwiającą płynną zmianę rozmiaru.

Tworząc interfejs oparty na `JFrame` można korzystać z dwóch podejść. Pierwszym z nich jest dziedziczenie bezpośrednio po klasie `JFrame` i uzupełnienie interfejsu komponentami.

```
public class MyFrame extends JFrame {
	public MyFrame() {
		super("Tytuł");
		setSize(100, 100);
		setVisible(true);
	}
}
```

Drugim sposobem jest typowe wywołanie obiektu `JFrame` i zasilanie komponentami poprzez referencję.

```
public class MyFrame {
	public static void main(String a[]) {
		JFrame f = new JFrame("Tytuł");
		f.setSize(100, 100);
		f.setVisible(true);
	}
}
```

Zarządcy rozkładu

Rozkład komponentów umieszczanych w kontenerze jest domyślnie zarządzany przez odpowiedni obiekt – zarządcę rozkładu. Rolą zarządcy rozkładu jest rozmieszczenie komponentów danego kontenera zgodnie z odpowiednimi regułami. Zarządca rozkładu bada preferowane rozmiary poszczególnych komponentów i na tej podstawie dobiera rozmiar kontenera. Z pomocą przychodzi tutaj metoda `pack()`, która dynamicznie dobiera rozmiar kontenera w zależności od rozmiaru komponentów na danej platformie.

Zarządca rozkładu jest również odpowiedzialny za przebudowę rozkładu przy zmianie rozmiaru kontenera. W każdym kontenerze jest domyślny zarządca rozkładu, który może być zmieniony lub wyłączony.

Zarządca BorderLayout

Obszar każdego z kontenerów jest podzielony na pięć regionów (`North`, `South`, `East`, `West`, `Center`). Z kolei do każdego z tych regionów może zostać dodany jeden komponent (kontener). Za bezpośrednie zarządzanie pięcioma komponentami odpowiada `BorderLayout`, który jest domyślnym zarządcą dla obiektów `JFrame` oraz `JDialog`.

Komponenty domyślnie przyjmują rozmiar dopasowany do komórki, którą zajmują. Przy zmianie kontenera:

- obszary `North` i `South` zachowują swoją wysokość;
- obszary `West` i `East` zachowują swoją szerokość.

```
public BLFrame() {
        add(new JButton("Center"), BorderLayout.CENTER);
        add(new JButton("South"), BorderLayout.SOUTH);
        add(new JButton("North"), BorderLayout.NORTH);
        add(new JButton("West"), BorderLayout.WEST);
        add(new JButton("East"), BorderLayout.EAST);
        pack();
        setVisible(true);
}
```

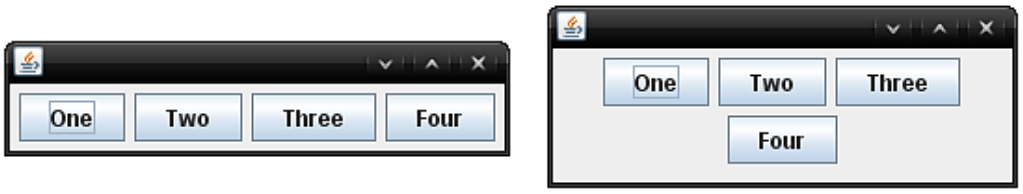
Efektem wykonania powyższego kodu będzie ramka zawierająca pięć przycisków odpowiadającym obszarom `BorderLayout`:

Zarządca FlowLayout

`FlowLayout` jest domyślnym zarządcą dla obiektu `JPanel`. Kontener może przyjmować dowolną liczbę komponentów. Komponenty te umieszczane są jeden po drugim, a gdy rozmiar okna nie pozawala na umieszczenie kolejnego w danym wierszu, umieszczany jest on w wierszu poniżej. Komponenty umieszczane są w preferowanym przez nie rozmiarze.

```
public FLFrame() {
        setLayout(new FlowLayout());
        add(new Button("One"));
        add(new Button("Two"));
        add(new Button("Three"));
        add(new Button("Four"));
        pack();
        setVisible(true);
}
```

Efektem powyższej metody będzie:



Zarządca GridLayout

W przypadku zarządcy `GridLayout` kontener dzieli obszar na komórki ułożone w wiersze i kolumny. Dodawane komponenty umieszczane są w kolejnych komórkach, a rozmiar tych

komórek dopasowany jest do największego komponentu. Komponenty wypełniają natomiast całą przestrzeń komórki jaką zajmują.

```
public GLFrame() {
        setLayout(new GridLayout(3,2));
        add(new JButton("One"));
        add(new JButton("Two"));
        add(new JButton("Three"));
        add(new JButton("Four"));
        add(new JButton("Five"));
        pack();
        setVisible(true);
}
```

Powyższa metoda daje następujący efekt:

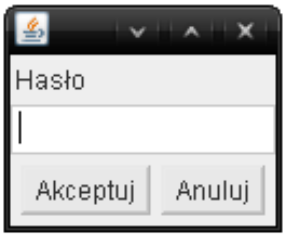
Budowanie rozkładów złożonych

Podczas tworzenia graficznych interfejsów użytkownika możliwe jest budowanie rozkładów złożonych. W tym celu należy wykorzystać własności dostępnych zarządców rozkładu oraz możliwość hierarchicznego układania kontenerów.

`JPanel` jest kontenerem pomocniczym przy budowaniu złożonych rozkładów. Wykorzystuje się tutaj metodę `pack()`, której wywołanie na rzecz najwyższego w hierarchii kontenera jest propagowane do każdego kontenera w nim osadzonego. Metoda `pack()` powoduje autodopasowanie rozmiaru okna, aby miało ono możliwość wyświetlenia wszystkich użytych komponentów.

```
public ComplexLayout() {
        add(new Label("Hasło"), BorderLayout.NORTH);
        add(new TextField(), BorderLayout.CENTER);
        Panel bottom = new Panel();
        bottom.add(new Button("Akceptuj"));
        bottom.add(new Button("Anuluj"));
        add(bottom, BorderLayout.SOUTH);
        pack();
        setVisible(true);
}
```

Efektem powyższej metody będzie:



Pozycjonowanie komponentów

Komponenty mogą być pozycjonowane niezależnie od zarządcy rozkładu. W przypadku ręcznego pozycjonowania należy wyłączyć zarządcę rozkładu oraz unikać stosowania metody `pack()`. Ręczne pozycjonowanie sprawdza się w przypadku, gdy aplikacja jest przewidziana dla jednej, konkretnej platformy (systemu operacyjnego).

```
	public ManualPositioning() {
		// wyłączenie zarządcy
		setLayout(null);
		JTextField tf = new JTextField();
		tf.setBounds(10, 30, 90, 30);
		add(tf);
		JButton b = new JButton("Akceptuj");
		b.setBounds(10, 60, 90, 30);
		add(b);
		setSize(100, 100);
		setVisible(true);
	}
```

Podstawowe komponenty

Do podstawowych komponentów wykorzystywanych przy budowaniu graficznych interfejsów graficznych zaliczamy:

- `JButton` – przycisk, z którym może być związana akcja;
- `JCheckbox` – pole wyboru umożliwiające zaznaczenie wybranej opcji. Może być organizowany w grupy tworząc tzw. przyciski radiowe, dające możliwość zaznaczenia tylko jednego przycisku z grupy;
- `JChoice` – opuszczane menu, umożliwiające dokonanie wyboru jednego z umieszczonych tam elementów;
- `JLabel` – element umożliwiający na umieszczenie opisu innych komponentów;
- `JList` – lista elementów do wyboru;
- `JTextField` – pole tekstowe umożliwiające wpisanie jednej linii tekstu;
- `JTextArea` – pole tekstowe pozwalające na wpisywanie tekstu w wielu wierszach.

Obsługa zdarzeń

Zdarzenia związane z interakcją z użytkownikiem są opisywane z postaci obiektów. Każdy komponent może generować różne rodzaje zdarzeń. Obiekt, który chce odbierać zdarzenia z wybranego komponentu spełniać dwa warunki. Pierwszym z nich jest zarejestrowanie się w wybranym komponencie na konkretny typ zdarzenia. Drugim warunkiem jest spełnienie

odpowiednich wymagań tj. zaimplementowanie interfejsu związanego z danym rodzajem zdarzenia.

Przygotowanie klasy obsługującej zdarzenia

Klasa obsługująca zdarzenie musi implementować odpowiedni interfejs. Dla zdarzeń akcji jest to `ActionListener`. Zdarzenia typu `Action` występują m.in. dla obiektów `JButton`, `JList` oraz `JTextField`. Obiekt będący źródłem zdarzeń rozsyła zaistniałe zdarzenie do wszystkich zarejestrowanych słuchaczy.

```
ActionHandler handler = new ActionHandler();
// ...
JButton b = new JButton("OK");
b.addActionListener(handler);
// ...
public class ActionHandler implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button pressed");
    }
}
```

Pozyskanie informacji o źródle zdarzenia

Obiekt opisujący zdarzenie posiada informacje o obiekcie źródłowym jako referencję. Zdarzenia typu `Action` można rozróżniać po skojarzonej z nimi `ActionCommand`, która domyślnie może być etykietą przycisku lub zawartością `JTextField`. Domyślną wartość można zmieniać poprzez metodę `setActionCommand()`.

```
public void actionPerformed(ActionEvent e) {
      if (e.getSource() == button) {
            // ...
      } else if (e.getActionCommand().equals("OK")) {
            // ...
      }
}
```

Rodzaje zdarzeń

Poniżej przedstawione zostały podstawowe rodzaje zdarzeń:

- `Action` – `ActionListener` – obsługa akcji wywołanych myszką lub klawiaturą;
- `Item` – `ItemListener` – oznacza zmianę wartości elementu (np. `Checkbox`);
- `Mouse` – `MouseListener`, `MouseMotionListener` – obsługa zdarzeń powstałych na skutek operowania myszką (wszystkie komponenty);
- `Window` – `WindowListener` – obsługa operacji dotyczących okna (`Frame`, `Dialog`);
- `Key` – `KeyListener` – obsługa klawiatury; dotyczy komponentów umożliwiających wprowadzanie tekstu;
- `Text` – `TextListener` – obsługa zmian wprowadzanego tekstu (`TextComponent`);

- `Focus` – `FocusListener` – przechwytywanie informacji o otrzymaniu lub utracie sterowania przez komponent;
- `Component` – `ComponentListener` – przechwytywanie informacji o zmianach komponentu;
- `Container` – `ContainerListener` – przechwytywanie informacji o zmianach w kontenerze.

Zdarzenia złożone

Wybrane zdarzenia mają więcej niż jedną metodę do obsługi, która jest zaimplementowana w odpowiednim interfejsie. Obsługa zdarzeń złożonym wiąże się z implementacją wszystkich niezbędnych metod. Klasa obsługująca zdarzenia może implementować dowolnie dużo interfejsów związanych z różnymi rodzajami zdarzeń.

```
public class ComplexEventHandling implements
                              ActionListener, MouseMotionListener {
      public void actionPerformed(ActionEvent e) {
            System.out.println(e.getActionCommand());
      }

      public void mouseDragged(MouseEvent e) {
            System.out.println(e.getPoint());
      }

      public void mouseMoved(MouseEvent e) {
            System.out.println(e.getPoint());
      }

      public static void main(String[] args) {
            Frame f = new Frame();
            ComplexEventHandling ceh = new ComplexEventHandling();
            TextField tf = new TextField(30);
            tf.addActionListener(ceh);
            tf.addMouseMotionListener(ceh);
            f.add(tf);
            f.pack();
            f.setVisible(true);
      }
}
```

Budowanie menu

Budując menu osadzamy je na pasku menu – `MenuBar` – który należy przywiązać do ramki. Menu stanowi uchwyt (kontener) do osadzania elementów menu. Każdy poszczególny element menu jest typu `MenuItem`. Do takiego elementu można zarejestrować odbiorcę zdarzeń typu `Action`. Dodatkowo, w menu można osadzać też inne elementy Menu, co daje możliwość budowania menu o strukturze hierarchicznej. W menu można osadzać również elementy typu `CheckboxMenuItem` – dające możliwość zaznaczenia konkretnego wyboru.

Przykładowe menu:

```
MenuBar mb = new MenuBar();
frame.setMenuBar(mb);

Menu file = new Menu("File");
mb.add(file);

MenuItem mi = new MenuItem("Open");
file.add(mi);

Menu submenu = new Menu("Sub Menu");
file.add(submenu);

MenuItem opt = new MenuItem("Option");
submenu.add(opt);
```