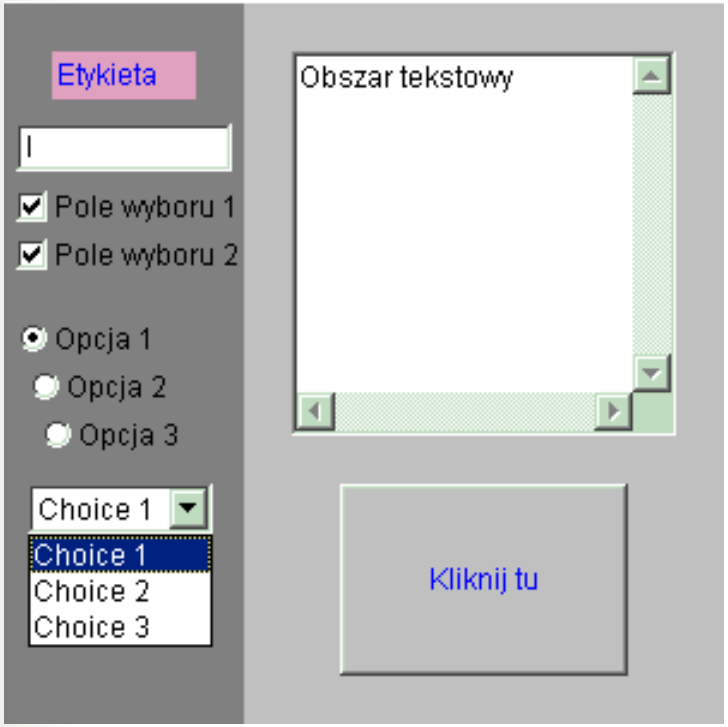
Projektowanie interfejsu

Pakiet AWT, komponenty i kontenery, klasa Component, metody klasy Component, klasa Label, komponenty tekstowe, klasa TextField, konstruktory i metody klasy TextField,

FUW

Pakiet AWT

- **Pakiet Abstract Windowing Toolkit zawiera zestaw klas, służących do tworzenia graficznego interfejsu użytkownika, nazywanego w skrócie GUI (Graphical User Interface).**



Poszczególne elementy GUI to komponenty, takie jak przyciski, listy, etykiety czy pola wyboru.
Są one reprezentowane przez klasy wywodzące się z java.awt.Component.

FUW

Komponenty i kontenery

Elementy GUI dzielą się na komponenty (kontrolki, sterowniki) oraz kontenery, które są używane do przechowywania innych elementów.

- przyciski (**Button**)
- płótna (**Canvas**)
- pola wyboru (**Checkbox**)
- etykiety (**Label**)
- paski przewijania (**ScrollBar**)
- listy (**List**)
- listy rozwjane (**Choice**)
- polecenia menu (**MenuItem**)
- komponenty tekstowe (**TextField** i **TextArea**)

- kontenery (**Container**)
- okna (**Window**)
- ramki (**Frame**)
- okna dialogowe (**Dialog**)
- dialogi plikowe (**FileDialog**)
- panele (**Panel**)
- okna przewijalne (**ScrollPane**)

Klasa Component

W klasie Component zdefiniowane są metody wspólne dla wszystkich komponentów. Metod tych używa się do pobierania (get), ustawiania (set) i sprawdzania (is) właściwości komponentów AWT.

Size ***rozmiar komponentu***	**getSize()** **getSize(Dimension rozmiar)** **setSize(int width, int height)**
Location ***położenie***	**getLocation()** **getLocation(Point p)** **setLocation(Point p)**
Bounds ***rozmiar i położenie***	**getBounds() lub getBounds(Rectangle r)** **setBounds(x, y, width, height)** **setBounds(Rectangle r)**

Metody klasy Component

Font ***pismo***	**getFont()** **setFont(Font f)**
Background ***kolor tła***	**getBackground()** **setBackground(Color c)**
Foreground ***kolor pierwszego planu***	**getForeground()** **setForeground(Color c)**
Visible ***widzialność***	**isVisible()** **setVisible(boolean b)**
Enabled ***dostępność***	**isEnabled()** **setEnabled(boolean b)**

Wyrównywanie komponentów

Do wyrównywania komponentów używane są następujące stałe, zdefiniowane w klasie Component:

- **BOTTOM_ALIGNMENT**
- **CENTER_ALIGNMENT**
- **LEFT_ALIGNMENT**
- **RIGHT_ALIGNMENT**
- **TOP_ALIGNMENT**

Zdarzenia

- **Graficzny interfejs użytkownika oprócz samego wyświetlania komponentów powinien reagować na zdarzenia pochodzące od użytkownika. Źródłami i słuchaczami zdarzeń są obiekty**
  - **zdarzenie (*event*) - obiekt "niosący" informację o stanie źródła**
  - **źródło (*source*) - obiekt, który generuje zdarzenia**
  - **słuchacz (*listener*) -obiekt powiadamiany o wystąpieniu zdarzenia.**
- **Każdy obiekt-słuchacz, który ma reagować na zdarzenia musi spełniać dwa wymogi:**
  1. **musi być zarejestrowany na liście słuchaczy zdarzeń. Rejestracji dokonuje się za pomocą metody addxxxListener(), wywoływanej na rzecz komponentu inicjującego zdarzenie, gdzie "xxx" reprezentuje typ nasłuchiwanego zdarzenia;**
  2. **musi mieć zaimplementowany odpowiedni interfejs nasłuchu:**
     - **ActionListener (dla przycisku i pola tekstowego),**
     - **AdjustaĺŐe (dla paska przewijania) oraz**
     - **ItemListener (dla pól wyboru i list)**

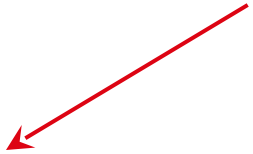
Klasa Button

- Przyciski polecenia są jednym z najczęściej używanych elementów graficznego interfejsu użytkownika.
- Do tworzenia przycisków wykorzystuje się następujące konstruktory klasy Button:
  - Button() - tworzy pusty przycisk (bez napisu)
  - Button(String) - tworzy przycisk z napisem
- Tekst na przycisku można ustawiać za pomocą metody
  - setLabel(String)
- Pobieranie tekstu wyświetlanego na przycisku realizuje metoda
  - getLabel()
- Rejestrowanie słuchacza zdarzeń akcji umożliwia metoda
  - addActionListener()
- Wyrejestrowanie słuchacza zdarzeń akcji - metoda
  - removeActionListener()

Przycisk i zdarzenia

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class Ok extends Applet implements ActionListener {
   Button button; String s;
   public void init() {
       s="";
       button = new Button("OK");
       button.addActionListener(this);
       add(button);
   }
   public void paint(Graphics g)  {
       g.drawString("" + s, 80, 70);
   }
   public void actionPerformed(ActionEvent e) {
       s = "" + Math.random();
       repaint();
   }
}
```



Gdy mamy kilka przycisków, możemy łatwo sprawdzić, który z nich wygenerował zdarzenie, np.

if (e.getSource() == button1) { ....

Klasa Label

- **Klasa Label służy do tworzenia etykiet umożliwiających wyświetlanie tekstu (tylko jeden wiersz).**
- **Etykiety są w zasadzie wykorzystywane do opisywania innych elementów apletu, więc tekst wyświetlany na nich nie może być modyfikowany przez użytkownika.**
- **Instrukcja tworząca etykietę ma postać:**

```
Label lab = new Label("Etykieta", Label.CENTER);
add(lab);
```

- **Tekst umieszczony w etykiecie może być wyrównany lewostronnie, prawostronnie lub wycentrowany za pomocą odpowiednich stałych:**
  - **Label.LEFT,**
  - **Label.RIGHT,**
  - **Label.CENTER.**

Własności etykiet

Jeśli chcemy utworzyć etykietę o zasięgu globalnym - by mieć do niej dostęp z dowolnej metody programu - musimy najpierw zadeklarować na poziomie klasy referencję do obiektu klasy Label, a następnie wewnątrz metody init() utworzyć obiekt klasy Label i ustawić jego początkowe własności.

```
public class Elementy extends Applet
{
   Label label;
   public void init()
   {
      setLayout(null);
      label = new Label("Etykieta", Label.CENTER);
      label.setBackground(new Color(96,148,148));
      label.setForeground(Color.orange);
      label.setFont(new Font("Arial", Font.BOLD, 16));
      label.setLocation(20,20);
      label.setSize(80,20);
      add(label); // dodanie etykiety do apletu
   }
```

Komponenty tekstowe

- **Pola tekstowe i obszary tekstowe to podklasy klasy komponentu tekstowego - TextComponent.**
- **Pola tekstowe - obiekty klasy TextField umożliwiają wprowadzanie i modyfikowanie tekstu, ale ich rozmiar jest ograniczony do jednego wiersza.**
- **Do wprowadzania większych, wielowierszowych tekstów stosuje się obszary tekstowe - obiekty klasy TextArea.**
- **Tworząc obszar tekstowy można dokładnie zaprojektować jego szerokość i wysokość, a także za pomocą odpowiednich stałych wyposażyć go w żądane paski przewijania:**
  - `TextArea.SCROLLBARS_BOTH`
  - `TextArea.SCROLLBARS_HORIZONTAL_ONLY`
  - `TextArea.SCROLLBARS_NONE`
  - `TextArea.SCROLLBARS_VERTICAL_ONLY`

Konstruktory klasy TextField

Konstruktor	Opis
`TextField()`	Tworzy nowe puste pole tekstowe
`TextField(int)`	Tworzy nowe puste pole tekstowe o podanej liczbie kolumn
`TextField(String)`	Tworzy nowe pole tekstowe zawierające podany tekst
`TextField(String, int)`	Tworzy nowe pole tekstowe zawierające podany tekst o podanej liczbie kolumn

Metody klasy TextField

Metoda	Opis
`echoCharIsSet()`	Sprawdza, czy pole tekstowe wyświetla zaszyfrowane informacje.
`getColumns()`	Zraca liczbę widocznych kolumn
`getEchoChar()`	Zwraca znak zastępujący wprowadzane litery (szyfrowanie tekstu)
`setColumns(int)`	Ustawia liczbę widocznych kolumn
`setEchoChar(char)`	Ustawia znak do szyfrowania
`setText(String)`	Ustawia domyślny tekst w polu tekstowym

Konstruktory klasy TextArea

Konstruktor	Opis
`TextArea()`	Tworzy nowy pusty obszar tekstowy.
`TextArea(int w, int k)`	Tworzy nowy pusty obszar tekstowy o określonej liczbie wierszy i kolumn.
`TextArea(String tekst)`	Tworzy nowy obszar tekstowy zawierający podany tekst.
`TextArea(String tekst, int w, int k)`	Tworzy nowy obszar tekstowy zawierający podany tekst, o określonej liczbie wierszy i kolumn.
`TextArea(String tekst, int w, int k, int paski)`	Tworzy nowy obszar tekstowy zawierający podany tekst, o określonej liczbie wierszy i kolumn, z podanymi paskami przewijania.

Metody klasy TextArea

Metoda	Opis
append(String s)	Dołącza podany tekst na koniec obszaru tekstowego.
getColumns()	Zwraca liczbę kolumn obszaru tekstowego
getRows()	Zwraca liczbę wierszy obszaru tekstowego
getScrollbarVisibility()	Zwraca informację o paskach przewijania
insert(String s, int n)	Wstawia podany tekst na podanej pozycji.
replaceRange(String s, int start, int end)	Zastępuje podanym tekstem informację w określonym zakresie
setColumns(int c)	Ustawia liczbę wyświetlanych kolumn
setRows(int r)	Ustawia liczbę wyświetlanych wierszy

Klasa ScrollBar

- **Klasa Scrollbar służy do tworzenia pasków przewijania.**
- **Komponenty te pozwalają na ustawianie wartości liczbowych za pomocą myszki.**
- **Wyróżniamy dwa rodzaje pasków przewijania: pasek poziomy i pionowy. Ich orientację określa się za pomocą stałych:**
  - **Scrollbar.HORIZONTAL**
  - **Scrollbar.VERTICAL**
- **Każda zmiana ustawienia paska przewijania generuje zdarzenie typu Adjustment.**
- **Jeśli klasa ma przechwytywać zdarzenia tego typu powinna implementować interfejs AdjustmentListener. Włącza on tylko jedną metodę (należy ją przedefiniować):**

```
public void adjustmentValueChanged(AdjustmentEvent  e)
{ }
```

Konstruktory klasy Scrollbar

Konstruktor	Opis
Scrollbar()	Tworzy nowy pionowy pasek przewijania
Scrollbar(int)	Tworzy nowy pionowy pasek przewijania o określonej orientacji
Scrollbar (int orientacja, int wartość, int rozmiar, intÏmin, int max)	Tworzy nowy pionowy pasek przewijania o podanej orientacji, wartości początkowej, rozmiarze ruchomego elementu, wartości minimalnej i wartości maksymalnej

Metody klasy Scrollbar

Metoda	Opis
getMaximum()	Zwraca maksymalną możliwą wartość
getMinimum()	Zwraca minimalną możliwą wartość
getOrientation()	Zwraca orientację paska
getValue()	Zwraca wartość bieżącą paska
getBlockIncrement()	Zwraca wartość blokowego skoku
getUnitIncrement()	Zwraca wartość jednostkowego skoku
setOrientation(int orientacja)	Ustawia orientację paska
setValue(int wartość)	Ustawia wartość bieżącą na pasku

Klasa Checkbox

- **Abstract Windowing Toolkit zawiera klasę Checkbox, która umożliwia tworzenie pól wyboru.**
- **Komponenty te mogą znajdować się w jednym z dwóch stanów - "wybrany" (true) albo "niewybrany" (false).**
- **Klikając na polu wyboru użytkownik może łatwo zmienić jego stan na przeciwny.**
- **Pola te są niezależne od siebie i użytkownik może dokonać wyboru wielokrotnego, tzn. wiele z tych pól może być ustawionych w pozycji "wybrany".**
- **Możliwe też jest tworzenie grup opcji za pomocą klasy CheckboxGroup, która wymusza pojedynczy wybór.**
- **Kiedy pola są zebrane w grupę, traktowane są jak przyciski radiowe (wzajemnie się wykluczające) i tylko jeden z nich może być wybrany.**

Konstruktory klasy Checkbox

Konstruktor	Opis
Checkbox()	Tworzy puste pole wyboru
Checkbox(String opis)	Tworzy pole wyboru z określoną etykietą opisu
Checkbox(String opis, boolean stan)	Tworzy pole wyboru z etykietą opisu oraz ustala jego stan
Checkbox(String opis, boolean stan, CheckboxGroup grupa)	Tworzy pole wyboru z z etykietą opisu, ustala jego stan oraz przypisuje je do danej grupy
Checkbox(String opis, CheckboxGroup grupa, boolean stan)	Tworzy pole wyboru z etykietą opisu, przydziela je do danej grupy oraz ustala jego stan.

Metody klasy Checkbox

Metoda	Opis
getCheckboxGroup()	Pobiera informację, do jakiej grupy należy dane pole wyboru
getLabel()	Pobiera etykietę pola wyboru
getSelectedObjects()	Zwraca jednoelementową tablicę zawierającą etykietę pola lub null jeśli pole nie jest zaznaczone
getState()	Sprawdza, czy pole jest zaznaczone
setCheckboxGroup(CheckboxGroup g)	Dodaje pole wyboru do podanej grupy
setLabel(String etykieta)	Ustawia etykietę pola wyboru
setState(boolean stan)	Ustawia stan pola wyboru

Listy i listy rozwijane

- **Komponenty te są wykorzystywane podczas tworzenia programów, w których użytkownik może dokonywać wyboru spośród większej liczby możliwych opcji.**
- **Klasa List służy do tworzenia list wyboru, które wyświetlają wiele elementów naraz.**
- **Jeśli mamy niewiele miejsca na aplecie, możemy utworzyć listę rozwijaną - komponent klasy Choice.**

```
List l =new List();
l.add("zielony");
l.add("czerwony");
l.add("niebieski");
add(l);
```

```
Choice c =new Choice();
c.addItem("zielony");
c.addItem("czerwony");
c.addItem("niebieski");
add(c);
```

Metody klasy Choice

Metoda	Opis
add(String element)	dodaje składnik do listy
addItem(String element)	dodaje składnik do listy
getItem(int numer)	zwraca napis odpowiadający elementowi o podanym numerze
getItemCount()	zwraca liczbę elementów listy
getSelectedIndex()	zwraca numer zaznaczonego elementu
getSelectedItem()	zwraca napis odpowiadający zaznaczonemu elementowi
getSelectedObjects()	zwraca jednoelementową tablicę zawierającą zaznaczony element

Metody klasy Choice

Metoda	Opis
insert(String element, int numer)	wstawia nowy element na podanej pozycji
remove(int pozycja)	usuwa element o podanej pozycji
remove(String elem)	usuwa pierwszy element wyznaczony przez podany napis
removeAll()	■ usuwa ws
select(int nr)	wybiera element o podanym numerze
select(String s)	wybiera element którego opis jest identyczny jak podany

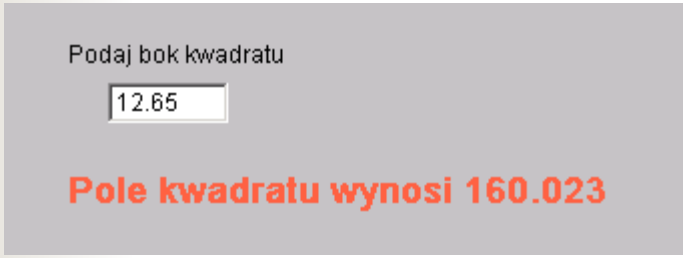
FUW

Menedżery układu

- **Klasy Javy umożliwiające automatyczne rozmieszczanie komponentów nazywamy menedżerami układu (*Layout managers*).**
- **Sposób ułożenia komponentów w oknie aplikacji zależy od użytego do tego celu menedżera.**
- **Pakiet AWT udostępnia pięć takich klas:**
  - **FlowLayout - układ ciągły**
  - **GridLayout - układ siatkowy**
  - **BorderLayout - układ brzegowy**
  - **CardLayout - układ kartowy**
  - **GridBagLayout - układ torebkowy**
- **Jeśli programista nie chce skorzystać z żadnego menedżera układu, może go pominąć. W tym celu musi on wywołać metodę setLayout(null), a następnie własnoręcznie określić współrzędne i rozmiary poszczególnych komponentów.**

Ćwiczenie 1

Utwórz aplet obliczający kwadraty podanych liczb rzeczywistych.



Umieść na aplecie dwa komponenty:

- **Pole tekstowe do wpisywania liczby**
- **Etykietkę z opisem tego pola**

W rozwiązaniu posłużymy się metodą action(). Metoda ta była używana w Javie 1.0 i obecnie jest już niepożądana. Mimo to występuje jeszcze w wielu apletach.

Proszę zwrócić uwagę, że po skompilowaniu kodu źródłowego otrzymamy komunikat:

Note: Kwadrat.java uses or overrides a deprecated API.

Rozwiązanie

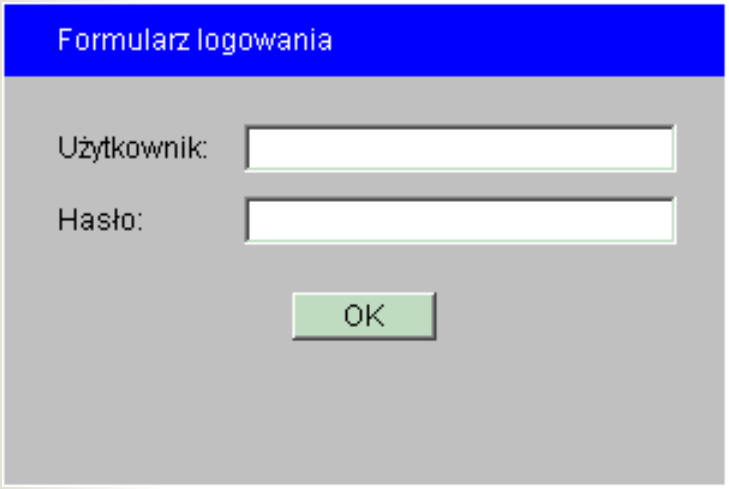
```
import java.awt.*;
import java.applet.*;
public class Kwadrat extends Applet {
   TextField t;
   public void init() {
      setLayout(null);
      t = new TextField();
      t.setBounds(60,50,60,20);
      t.setText("0.0");
      add(t);

   public void paint(Graphics g) {
      g.drawString("Podaj bok kwadratu", 40, 40);
      String s = t.getText();
      double a = Double.valueOf(s).doubleValue();
      double pole = Math.round((a*a*1000))/1000.0;
      g.setColor(new Color(255,99,71));
      g.setFont(new Font("TimesRoman",Font.BOLD,18));
      g.drawString("Pole = " + pole, 40, 110);
   }
   public boolean action(Event e, Object arg){
      repaint();
      return true;
```

FUW

Ćwiczenie 2

Zaprojektuj prosty formularz logowania.



Umieść na aplecie komponenty:

- **Pola tekstowe -do wpisywania nazwy użytkownika i hasła;**
- **Etykietki z opisem tych pól;**
- **Przycisk do zatwierdzania wprowadzonych danych.**

Działanie formularza jest następujące: po wpisaniu i zatwierdzeniu prawidłowej nazwy użytkownika i hasła powinien wyświetlić się napis "Hasło przyjęte". Wpisanie i zatwierdzenie niepoprawnych danych powinno spowodować wyczyszczenie pól tekstowych.

FUW

Rozwiązanie

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class Login extends Applet
implements ActionListener
{
   String s;
   TextField t1, t2;
   Button button;
   public void init()
   {
     s = "";
     setLayout(null);
     Panel p = new Panel();
     p.setBounds(0, 0, 300, 30);
     p.setBackground(Color.blue);
     add(p);
     p.setLayout(null);
     Label label = new Label("Formularz logowania");
     label.setForeground(Color.white);
     label.setBounds(20, 5, 120, 20);
     p.add(label);
```

Rozwiązanie

```
    Label l1 = new Label("Użytkownik:");
    ¤1.setBounds(20, 50, 70, 20);
    add(l1);

    Label l2 = n¤w Label("Hasło:");
    l2.setBounds(20, 80, 70, 20);
    add(l2);

    t1 = new TextField();
    t1.setBounds(100, 50, 180, 20);
    add(t1);

    t2 = new TextField();
    t2.setEchoChar('*');
    t2.setBounds(100, 80, 180, 20);
    add(t2);

    button = new Button("OK");
    button.setBounds(120, 120, 60, 20);
    add(button);
    button.addActionListener(this);
}
```

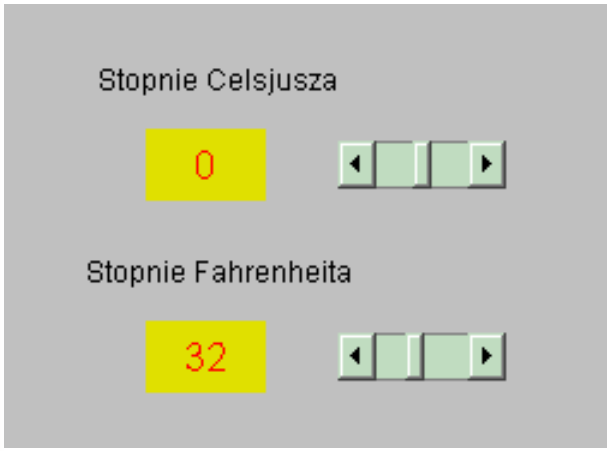
Rozwiązanie

```
    public void paint(Graphics g)
    {
        g.drawString(s, 20, 170);
    }

    public void actionPerformed(ActionEvent e)
    {
        String s1=t1.getText();
        String s2 = t2.getText();
        if (s1.equals("Student") && s2.equals("a"))
        {
            s="Hasło poprawne";
        }
        else
        {
            t1.setText("");
            t2.setText("");
            s="";
        }
        repaint();
    }
}
```

Zadania

1. **Napisz aplet służący do zamiany stopni Celsjusza na stopnie Fahrenheita za pomocą paska przewijania ($t_C=1.8t_F+32$).**
2. **Do poprzedniego apletu dodaj jeszcze jeden pasek przewijania, który będzie odzwierciedlał zmiany wartości temperatury w skali Fahrenheita.**



FUW

Rozwiązanie

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class Suwak extends Applet
implements AdjustmentListener
{
  Scrollbar cels, fahren;
  Label l1,l2,lC,lF;
  int min = -100, max = 101;

public void init()
{
  setLayout(null);
  l1=new Label("Stopnie Celsjusza",Label.CENTER);
  l1.setBounds(40,30, 110, 20);
  add(l1);
  l2=new Label("Stopnie Fahrenheita",Label.CENTER);
  l2.setBounds(40,110, 110, 20);
  add(l2);
  Font font = new Font("Arial",Font.PLAIN,16);
  Color color = new Color(230,220,20);
```

Rozwiązanie

```
lC = new Label("0",Label.CENTER);
lC.setFont(font);
lC.setBounds(65, 60, 50, 30);
lC.setBackground(color);
lC.setForeground(Color.red);
add(lC);
lF = new Label("32",Label.CENTER);
lF.setFont(font);
lF.setBounds(65, 140, 50, 30);
lF.setBackground(color);
lF.setForeground(Color.red);
add(lF);
cels = new Scrollbar(Scrollbar.HORIZONTAL,0,1,min,
 max);
cels.setBounds(145, 65, 70, 20);
add (cels);
cels.addAdjustmentListener(this);
fahren = new Scrollbar(Scrollbar.HORIZONTAL, 0, 1,
 -148, 213);
fahren.setBounds(145, 145, 70, 20);
add(fahren);
fahren.addAdjustmentListener(this);
}
```

Rozwiązanie

```
public void adjustmentValueChanged(AdjustmentEvent e)
{
   Object o = e.getSource();
   if (o.equals(cels)) {
      int x = cels.getValue();
      int y = (int) (1.8*x+32);
      lC.setText(String.valueOf(x));
      lF.setText(String.valueOf(y));
      fahren.setValue(y);
   }
   else if (o.equals(fahren)) {
      int x = fahren.getValue();
      double y = (x-32)/1.8;
      int z = (int) y;
      lC.setText(String.valueOf(z));
      lF.setText(String.valueOf(x));
      cels.setValue(z);
   }
}
}
```