Wykład V

Java - środowisko GUI

Tworzone programy pobierały dane z klawiatury, przetwarzały je i wyświetlały wyniki w konsoli lub prostych obiektach typu InputBox lub MsgBox .
Aktualne wymagania koncentrują się na nowoczesnych interfejsach użytkownika bazujących na graficznych rozwiązaniach. Tworzenie aplikacji jest związane z nauką pisania programów z graficznym interfejsem użytkownika (GUI).
Budowa aplikacji graficznych wymaga poznania i umiejętności zastosowania bibliotek graficznych dostępnych w Javie, takich jak AWT, Swing czy Java FX. W trakcie wykładów przedstawione zostaną rozwiązania umożliwiające zastosowanie następujących elementów aplikacji graficznych bazujących na obsłudze zdarzeń.
Poznamy:

- Obiekty odpowiedzialne z wyświetlenie, ustawienie rozmiaru i położenie okien na ekranie,
- Wyświetlenie tekst pisany różnymi krojami czcionki,
- Wyświetlenie i manipulacja obrazów
- Podstawowe obiekty interfejsu GUI pola tekstowe, etykiety, przyciski, listy, menu itd.
- Przetwarzanie zdarzeń, takie jak wciśnięcie klawisza na klawiaturze lub kliknięcie przyciskiem myszy lub inne zdarzenia systemowe ,
- Tworzenie figur 2D
- Metody użycia palety kolorów
- Tworzenie kompozycji interfejsu z rozmieszczeniem obiektów

Podstawową biblioteką graficzną jest biblioteka klas AWT Abstract Window Toolkit (AWT), która dostarczała podstawowe narzędzia do programowania GUI.
Biblioteka AWT wykorzystuje narzędzia tworzenia GUI platformy systemowej (Windows Mac OS X, Linux, Solaris) do tworzenia elementów interfejsu oraz obsługi ich zachowań. Użycie oryginalnej biblioteki AWT do umieszczenia w oknie pola tekstowego spowoduje. że wszystkie działania związane ze zdarzeniami oraz danymi będą realizowane przez odpowiedni obiekt wykorzystywanego systemu operacyjnego.
Takie podejście umożliwiało tworzenie dowolnych aplikacji z obiektami GUI które wyglądają i są obsługiwane tak jak obiekty w tym systemie operacyjnym. Aplikacja napisana w Javie wyglądała w każdym systemie operacyjnym inaczej ale zgodnie z wyglądem aplikacji w tym systemie. Styl aplikacji był powiązany z systemem operacyjnym. Stąd slogan firmy Sun: „Napisz raz, uruchamiaj wszędzie”.
Zalety biblioteki AWT:

- Styl aplikacji powiązany z systemem operacyjnym
- Obsługa zdarzeń ściśle związana z systemem operacyjnym
- Środowisko odpowiednie dla prostych aplikacji

Metoda programowania wykorzystująca odpowiedniki obiektów GUI w systemach operacyjnych sprawdzała się w przypadku prostych aplikacji.
Wady biblioteki AWT:

- Nie nadaje się do tworzenia złożonych aplikacji graficznych gdyż obiekty a w

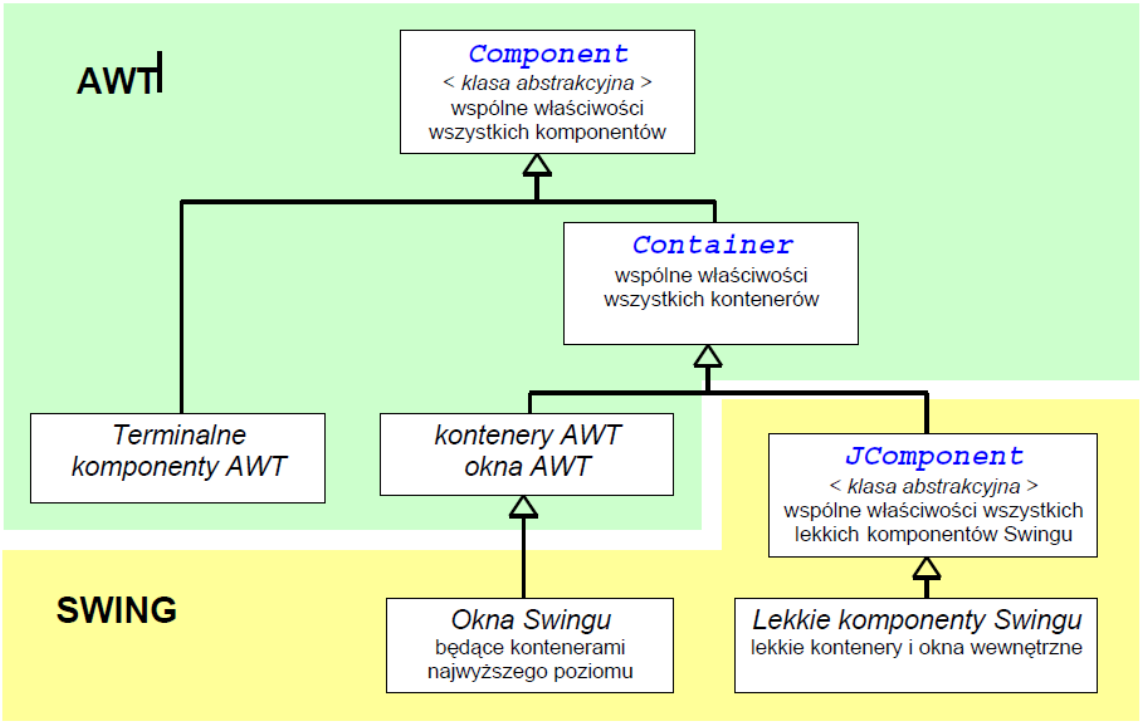
szczególności sposób ich działania różni się w stosowanych systemach operacyjnych
- Systemy operacyjne oferują różne biblioteki GUI co prowadzi do zawężenia wykorzystywanych obiektów
- Błędy biblioteki AWT dla różnych systemów operacyjnych (aplikacja mogła poprawnie działać w dwóch systemach a w trzecim mogły pojawiać się błędy)
- Zwiększenie nakładu pracy programistów, testy aplikacji musiały być prowadzone na wszystkich platformach, co złośliwie nazywano „napisz raz, testuj wszędzie”

Biblioteka Swing

Usunięcie wad biblioteki AWT wymagał stworzenia nowej biblioteki graficznej .
Swing stał się oficjalną nazwą zestawu narzędzi do tworzenia GUI, niewykorzystującego systemowych odpowiedników elementów. Swing wchodzi w skład Java Foundation Classes (JFC). Biblioteka JFC jest bardzo duża i zawiera wiele innych narzędzi poza Swingiem. Zaliczają się do nich interfejsy API dostępności, grafiki dwuwymiarowej oraz obsługi operacji przeciągania i upuszczania.

Biblioteka Swing nie jest osobnym komponentem języka Java jest ona nadbudową biblioteki AWT z wykorzystaniem jej architektury a w szczególności z wykorzystania obsługi zdarzeń. Biblioteka Swing oferuje ujednolicony interfejs użytkownika w ramach wszystkich systemów operacyjnych i architektury sprzętowej umożliwiających uruchomienie wirtualnej maszyny Java. Podstawowym zadaniem biblioteki Swing jest oferowanie klas umożliwiających utworzenie zaawansowanego interfejsu użytkownika.

Komponenty AWT a komponenty Swingu



JavaFX

JavaFX jest rozwiązaniem RIA (Rich Internet Application)
Umożliwia programistom integrację grafiki wektorowej, animacji, sieciowych zasobów dźwiękowych oraz wideo w procesie tworzenia bogatych, interaktywnych i złożonych aplikacji. Rozszerza technologię Java, umożliwiając korzystanie z dowolnej biblioteki

Java w aplikacjach JavaFX.
Aplikację można uruchomić :

- na pulpicie,
- w przeglądarce - jako aplet.
- na urządzeniu mobilnym,

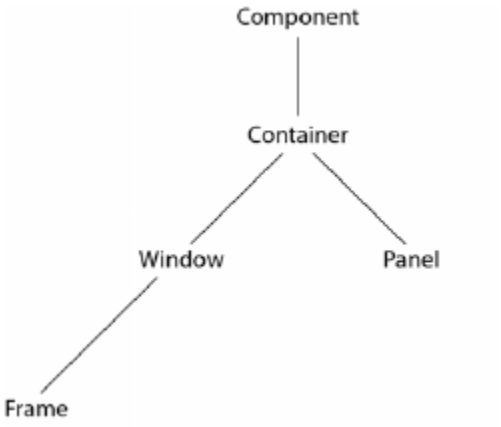
Pełna obsługa multimediów

- maj 2007 - pierwsze informacje.
- 4 grudzień 2008 – upublicznienie wersji 1.0.
- 12 lutego 2009 – JavaFX 1.1 dla urządzeń mobilnych
- 2 czerwca 2009 – JavaFX 1.2, dodanie CSS, wykresów, obsługa I/O, poprawki związane z wydajnością, wsparcie dla systemów Linux i Solaris
- 22 kwietnia 2010 – JavaFX 1.3
- 10 października 2011 – JavaFX 2.0, usunięcie JavaFX Script, usunięcie JavaFX Mobile, wprowadzenie FXML, tylko Windows
- 27 kwietnia 2012 – JavaFX 2.1, wsparcie dla Macintosh
- 14 sierpnia 2012 – JavaFX 2.2, wsparcie dla Linuxa, element Java SE7u6
- lipiec 2014 - JavaFX 2.2.7 element Java SE8

Tworzenie aplikacji okienkowych z wykorzystaniem biblioteki AWT

Tworzenie Okna

Klasy AWT znajdują się w pakiecie java.awt. Jest to jeden z największych pakietów Javy. AWT definiuje okna zgodnie z hierarchią klas, w której każdy kolejny poziom oznacza poszerzoną funkcjonalność i większą specjalizację. Do najczęściej stosowanych okien należą klasy dziedziczące bezpośrednio po klasie Panel (czyli te wykorzystywane w przypadku apletów) oraz dziedziczące po klasie Frame (które tworzą standardowe okna).



Klasa Component

Na szczycie hierarchii AWT znajduje się klasa Component. Jest to klasa abstrakcyjna definiująca wszystkie atrybuty wizualnych komponentów AWT. Wszystkie elementy graficznego interfejsu użytkownika (poza menu), które są widoczne na ekranie i które umożliwiają interakcję z użytkownikiem, tak naprawdę są podklasami klasy Component. Klasa ta definiuje ponad sto publicznych metod, które odpowiadają za zarządzanie zdarzeniami (także tymi generowanymi przez mysz i klawiaturę użytkownika), pozycjonowanie i dostosowywanie rozmiaru okna oraz ponowne wyświetlanie komponentów.
Do zadań obiektów klasy Component należy zapamiętywanie bieżących kolorów tła i pierwszego planu oraz aktualnie wybranej czcionki.

Klasa Window

Klasa Window tworzy okno najwyższego poziomu. **Okno najwyższego poziomu** nie może się zawierać w żadnym innym obiekcie; działa bezpośrednio na pulpicie. Zasadniczo obiektów klasy Window nie tworzy się bezpośrednio; zamiast tego stosuje się podklasę klasy Window nazwaną Frame.

Klasa Frame

Klasa Frame reprezentuje to, co powszechnie określa się mianem „okna”. Frame jest podklasą klasy Window i definiuje pasek tytułu, pasek menu, obramowania i narożniki zmiany rozmiaru. Jej konkretny wygląd jest różny i zależy od używanego środowiska.

Klasa Canvas

Chociaż ta klasa nie jest częścią hierarchii związanej z oknami apletów, istnieje jeszcze inny typ okien, który w wielu przypadkach może się okazać bardzo przydatny: typ Canvas. Klasa Canvas, dziedzicząca po Component, reprezentuje puste okno, w którym możemy rysować potrzebne komponenty.

Tworzenie programu okienkowego

Okno najwyższego poziomu, tzn. takie, które nie jest zawarte w żadnym innym oknie, nazywa się w Javie ramką (ang. frame). W bibliotece AWT dla tego typu okien utworzono klasę o nazwie Frame.
Utworzenie okienka to utworzenie obiektu w metodzie main(). Działanie metody main() kończy się wraz z wywołaniem metody appwin.setVisible(true).
Program działa jednak dalej aż do momentu zamknięcia tego okna. W praktyce, kiedy budujemy aplikację wykorzystującą okna, metoda main() służy do tworzenia i wyświetlania jej głównego okna (ze szczytu hierarchii okien). Taki program funkcjonuje ostatecznie jako aplikacja oparta na graficznym interfejsie użytkownika, a więc zupełnie inaczej niż programy konsoli
Przykład kodu źródłowego

```
// Tworzy aplikację okienkową GUI.
import java.awt.*;
import java.awt.event.*;

public class Okno_AWT extends Frame
{

// Tworzy okno.
public static void main(String args[])
{
Okno_AWT okienko = new Okno_AWT();
okienko.setSize(new Dimension(300, 200));
// okienko.setSize(300, 200);
okienko.setTitle("Aplikacja oparta na AWT");
okienko.setVisible(true);
}
}
```

Ustawianie wymiarów okna

Do ustawiania wymiarów okna służy metoda setSize(). Składnię metody przedstawiono poniżej:
void setSize(int *newWidth*, int *newHeight*)
void setSize(Dimension *newSize*)

Rozmiar okienka

Docelowy rozmiar okna definiuje się albo za pośrednictwem parametrów *newWidth* i *newHeight*,
albo za pośrednictwem pól width i height obiektu klasy Dimension reprezentowanego przez parametr *newSize*. Wszystkie wymiary są wyrażane w pikselach.
Do odczytywania bieżącego rozmiaru okna służy metoda getSize(). Składnia tej metody jest następująca: Dimension getSize()
Metoda getSize() zwraca bieżący rozmiar okna w postaci odpowiednich wartości pól width i height obiektu klasy Dimension.

Ukrywanie i wyświetlanie okna

Po utworzeniu okna typu Frame nie będzie ono widoczne na ekranie aż do momentu wywołania metody setVisible(), której składnię przedstawiono poniżej:
void setVisible(boolean *visibleFlag*)
Komponent będzie widoczny, jeśli przekażemy do tej metody wartość true za pośrednictwem jej jedynego parametru. W przeciwnym przypadku okno zostanie ukryte.

Ustawianie tytułu okna

Za pomocą metody setTitle() możemy zmienić tytuł okna typu Frame; składnia tej metody jest następująca: void setTitle(String *newTitle*)
Parametr *newTitle* reprezentuje w tym przypadku nowy tytuł danego okna.

Zamykanie okna typu Frame

Jeśli zdecydujemy się na stosowanie okien typu Frame, musimy zapewnić mechanizm ich usuwania z ekranu w momencie zamykania programu w tym celu należy używać wspomnianej już metody setVisible(false). Aby przechwycić zdarzenie zamykania okna, musimy zaimplementować metodę windowClosing() interfejsu WindowListener. Wewnątrz metody windowClosing() musimy usunąć dane okno z ekranu.
Przykład przedstawiony w następnym podrozdziale ilustruje zastosowanie tej techniki w praktyce.
Aplikacja działa ale nie obsługuje żadnych zdarzeń również zdarzenia zamknięcia okna

Przykład aplikacji z obsługa zdarzenia zamknięcia okna:

```
// Tworzy aplikację okienkową GUI.
import java.awt.*;
import java.awt.event.*;

public class Okno_AWT extends Frame implements  WindowListener
{
    public Okno_AWT() {

       this.addWindowListener(this);
       }
// Tworzy okno.
public static void main(String args[])
{
 Okno_AWT okienko = new Okno_AWT();
okienko.setSize(500, 300);
// okienko.setSize(new Dimension(300, 200));
okienko.setTitle("Aplikacja oparta na AWT");
okienko.setVisible(true);
}

public void windowOpened(WindowEvent e) {
}
public void windowClosing(WindowEvent e) {
      System.exit(0);
}
public void windowClosed(WindowEvent e) {
}
public void windowIconified(WindowEvent e) {
}
public void windowDeiconified(WindowEvent e) {
}
public void windowActivated(WindowEvent e) {
}
public void windowDeactivated(WindowEvent e) {
}
}
```

Ten sam program z użyciem abstrakcyjnej klasy WindowAdapter

```
// Tworzy aplikację okienkową GUI.
import java.awt.*;
import java.awt.event.*;

public class Okno_AWT extends Frame
{
    public Okno_AWT() {
       addWindowListener(new Moje_zdarzenia_okno());
       }
// Tworzy okno.
public static void main(String args[])
{
 Okno_AWT okienko = new Okno_AWT();
okienko.setSize(500, 300);
// okienko.setSize(new Dimension(300, 200));
okienko.setTitle("Aplikacja oparta na AWT");
okienko.setVisible(true);
}
class Moje_zdarzenia_okno extends WindowAdapter {
public void windowClosing(WindowEvent we) {
System.exit(0);
}
}
}
```

Obsługa zdarzeń związanych z myszką

```
// Tworzy aplikację okienkową GUI.
import java.awt.*;
import java.awt.event.*;

public class Okno_AWT extends Frame implements  WindowListener, MouseListener
{
       String keymsg = "Łańcuch testowy.";
       String mousemsg = "";
       int mouseX=30, mouseY=30;
     public Okno_AWT() {
       this.addMouseListener(this);
       this.addWindowListener(this);
       }
     public void paint(Graphics g) {
       g.drawString(keymsg, 10, 40);
       g.drawString(mousemsg, mouseX, mouseY);
       }
// Tworzy okno.
public static void main(String args[])
{
 Okno_AWT okienko = new Okno_AWT();
okienko.setSize(500, 300);
// okienko.setSize(new Dimension(300, 200));
okienko.setTitle("Aplikacja oparta na AWT");
okienko.setVisible(true);
}

public void windowOpened(WindowEvent e) {
}
public void windowClosing(WindowEvent e) {
       System.exit(0);
}
public void windowClosed(WindowEvent e) {
}
public void windowIconified(WindowEvent e) {
}
public void windowDeiconified(WindowEvent e) {
}
public void windowActivated(WindowEvent e) {
}
public void windowDeactivated(WindowEvent e) {
}
public void mouseClicked(MouseEvent e) {
}
public void mousePressed(MouseEvent e) {
       //Okno_AWT okienko_m = new Okno_AWT();
       //this.okienko_m = okienko_m;
       mouseX = e.getX();
       mouseY = e.getY();
       mousemsg = "Naciśnięto przycisk myszy w punkcie (" + mouseX +
       ", " + mouseY + ")";
       repaint();
}
public void mouseReleased(MouseEvent e) {
}
public void mouseEntered(MouseEvent e) {
}
public void mouseExited(MouseEvent e) {
}
}
```

Obsługa zdarzeń związanych z myszką przy użyciu abstrakcyjnej klasy `MouseAdapter`

```
// Tworzy aplikację okienkową GUI.
import java.awt.*;
import java.awt.event.*;

public class Okno_AWT extends Frame
{
       String mousemsg = "";
       String keymsg = "Łańcuch testowy.";
       int mouseX=30, mouseY=30;
    public Okno_AWT()
       {
       addWindowListener(new Moje_zdarzenia_okno());
       addMouseListener(new Moje_zdarzenia_myszka(this));
       }
    public void paint(Graphics g) {
       g.drawString(keymsg, 10, 40);
       g.drawString(mousemsg, mouseX, mouseY);
       }

// Tworzy okno.
public static void main(String args[])
{
 Okno_AWT okienko = new Okno_AWT();
okienko.setSize(500, 300);
// okienko.setSize(new Dimension(300, 200));
okienko.setTitle("Aplikacja oparta na AWT");
okienko.setVisible(true);
}
class Moje_zdarzenia_okno extends WindowAdapter {
public void windowClosing(WindowEvent we) {
System.exit(0);
}
}
class Moje_zdarzenia_myszka extends MouseAdapter {
       Okno_AWT okienko_m;
       public Moje_zdarzenia_myszka(Okno_AWT okienko_m) {
       this.okienko_m = okienko_m;
       }
public void mousePressed(MouseEvent me) {
okienko_m.mouseX = me.getX();
okienko_m.mouseY = me.getY();
okienko_m.mousemsg = "Naciśnięto przycisk myszy w punkcie (" + okienko_m.mouseX
+
", " + okienko_m.mouseY + ")";
okienko_m.repaint();
}
}
}
```

Obsługa zdarzeń związanych z klawiaturą

```
// Tworzy aplikację okienkową GUI.
import java.awt.*;
import java.awt.event.*;

public class Okno_AWT extends Frame implements  WindowListener, MouseListener,
KeyListener
{      String keymsg = "Łańcuch testowy.";
       String mousemsg = "";
       String msg ="";
       int k_X = 10, k_Y=140;
       int mouseX=30, mouseY=30;
    public Okno_AWT() {
       this.addKeyListener(this);
       this.addMouseListener(this);
       this.addWindowListener(this);
       }
    public void paint(Graphics g) {
       g.drawString(keymsg, 10, 40);
       g.drawString(mousemsg, mouseX, mouseY);
       g.drawString(msg, k_X, k_Y);
       }
// Tworzy okno.
public static void main(String args[])
{
 Okno_AWT okienko = new Okno_AWT();
okienko.setSize(500, 300);
// okienko.setSize(new Dimension(300, 200));
okienko.setTitle("Aplikacja oparta na AWT");
okienko.setVisible(true);
}

public void windowOpened(WindowEvent e) {
}
public void windowClosing(WindowEvent e) {
       System.exit(0);
}
public void windowClosed(WindowEvent e) {
}
public void windowIconified(WindowEvent e) {
}
public void windowDeiconified(WindowEvent e) {
}
public void windowActivated(WindowEvent e) {
}
public void windowDeactivated(WindowEvent e) {
}
public void mouseClicked(MouseEvent e) {
}
public void mousePressed(MouseEvent e) {
       //Okno_AWT okienko_m = new Okno_AWT();
       //this.okienko_m = okienko_m;
       mouseX = e.getX();
       mouseY = e.getY();
       mousemsg = "Naciśnięto przycisk myszy w punkcie (" + mouseX +
       ", " + mouseY + ")";
       repaint();
}
public void mouseReleased(MouseEvent e) {
}
public void mouseEntered(MouseEvent e) {
```

```
}
public void mouseExited(MouseEvent e) {
}
public void keyTyped(KeyEvent e) {
	k_X=150;
	k_Y=150;
	msg += e.getKeyChar();
	repaint();
}
public void keyPressed(KeyEvent e) {
	k_X=100;
	k_Y=200;
	//msg = "klawisz wciśnięty ";
	repaint();
}
public void keyReleased(KeyEvent e) {
	k_X=200;
	k_Y=100;
	//msg = "klawisz zwolniony";
	repaint();
}
}
```

Obsługa zdarzeń związanych z klawiaturą przy użyciu abstrakcyjnej klasy `KeyAdapter`

```
// Tworzy okno.
public static void main(String args[])
{
 Okno_AWT okienko = new Okno_AWT();
okienko.setSize(500, 300);
// okienko.setSize(new Dimension(300, 200));
okienko.setTitle("Aplikacja oparta na AWT");
okienko.setVisible(true);
}
class Moje_zdarzenia_okno extends WindowAdapter {
public void windowClosing(WindowEvent we) {
System.exit(0);
}
}
class Moje_zdarzenia_myszka extends MouseAdapter {
	Okno_AWT okienko_m;
	public Moje_zdarzenia_myszka(Okno_AWT okienko_m) {
	this.okienko_m = okienko_m;
	}
public void mousePressed(MouseEvent me) {
okienko_m.mouseX = me.getX();
okienko_m.mouseY = me.getY();
okienko_m.mousemsg = "Naciśnięto przycisk myszy w punkcie (" + okienko_m.mouseX
+
", " + okienko_m.mouseY + ")";
okienko_m.repaint();
}
}
class Moje_zdarzenia_klawiatura extends KeyAdapter {
	Okno_AWT okienko_m;
	public Moje_zdarzenia_klawiatura(Okno_AWT okienko_m) {
	this.okienko_m = okienko_m;
	}
	public void keyTyped(KeyEvent ke) {
		okienko_m.keymsg += ke.getKeyChar();
		okienko_m.repaint();
		};
}
}
```

Obsługa zdarzeń związanych z niezależnymi od użytkownika zdarzeniami klasa Timer

Klasy Timer i TimerTask

Planowanie zadań w przyszłości to jedno z częstszych zdarzeń obsługiwanych przez zegar czasu rzeczywistego jest to element niezwykle przydatny a oferowany przez pakiet java.util
Klasy umożliwiające realizację planowania zadań w przyszłości to klasy Timer i TimerTask. Stosowanie tych klas umożliwia tworzenie wątków, które będą oczekiwały w tle na określony wcześniej moment. Kiedy ten moment nadejdzie, zostanie wykonane zadanie powiązane z takim wątkiem. Rozmaite opcje wspomnianych klas oferują możliwość nie tylko planowania wielokrotnego wykonywania tych samych zadań, ale także odkładania realizacji zadań na określony dzień.
Klasy Timer i TimerTask zawsze funkcjonują razem. Klasa Timer służy do planowania zadań wykonywanych w przyszłości. Planowane w ten sposób zadanie musi być egzemplarzem klasy TimerTask.
Oznacza to, że aby planowanie zadań było możliwe, musimy najpierw utworzyć obiekt klasy TimerTask, a dopiero potem zaplanować jego wykonanie w oparciu o egzemplarz klasy Timer.
Klasa TimerTask implementuje interfejs Runnable, zatem może być wykorzystywana do tworzenia wykonywalnych wątków.
Jedną z podstawowych metod klasy TimerTask. jest metoda run(), która jest metodą abstrakcyjną, co oznacza, że należy ją przykryć. Zdefiniowana przez interfejs Runnable metoda run() zawiera kod przeznaczony do wykonania. Najprostszym sposobem stworzenia zadania, którego realizacja jest planowana na przyszłość, jest więc stworzenie klasy dziedziczącej po klasie TimerTask i przykrywającej metodę run().

Przykład:

```
// Tworzy aplikację okienkową GUI.
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.awt.Dialog;

public class Okno_AWT extends Frame implements  WindowListener, MouseListener,
KeyListener
{      String keymsg = "Łańcuch testowy.";
       String mousemsg = "";
       String msg ="";
       int k_X = 10, k_Y=140;
       int mouseX=30, mouseY=30;
// Deklaracja i inicjacja zmiennych klasy Timer i TimerTask
       Timer Zegar = new Timer();
       Zadania_Zegara Moje_zadania = new Zadania_Zegara();

    public Okno_AWT() {
       this.addKeyListener(this);
       this.addMouseListener(this);
       this.addWindowListener(this);
// Wywołanie metody schedule dla klasy Timer
       Zegar.schedule(Moje_zadania, 0, 1000);
       }
    public void paint(Graphics g) {
       g.drawString(keymsg, 10, 40);
       g.drawString(mousemsg, mouseX, mouseY);
       g.drawString(msg, k_X, k_Y);
       }
// Tworzy okno.
public static void main(String args[])
{
 Okno_AWT okienko = new Okno_AWT();
```

```
okienko.setSize(500, 300);
// okienko.setSize(new Dimension(300, 200));
okienko.setTitle("Aplikacja oparta na AWT");

okienko.setVisible(true);
}

//Timer Zegar;
//Zadania_Zegara Moje_zadania;

// Definicja klasy Zadania_zegara dziedziczące po klasie TimerTask

class Zadania_Zegara extends TimerTask
{
private int i = 0;
// Definicja metody run () pokrywającej metodę abstrakcyjną klasy TimerTask

      public void run()
      {
         switch (i)
         {
         case 0: setBackground(Color.ORANGE);
         break;
         case 1: setBackground(Color.BLUE);
         break;
         case 2: setBackground(Color.GREEN);
         break;
         case 3: setBackground(Color.DARK_GRAY);
         break;
         case 4: setBackground(Color.CYAN);
         break;
         default:
         // Nieprawidłowe dane.
               setBackground(Color.RED);
               msg ="Nieoczekiwana wartość";
               k_X=40;
               k_Y=50;
               repaint();
         break;
         }
            i++;
            if (i>5) i=0;
      }
}

public void windowOpened(WindowEvent e) {
}
public void windowClosing(WindowEvent e) {
      System.exit(0);
}
public void windowClosed(WindowEvent e) {
}
public void windowIconified(WindowEvent e) {
}
public void windowDeiconified(WindowEvent e) {
}
public void windowActivated(WindowEvent e) {
}
public void windowDeactivated(WindowEvent e) {
}
@Override
public void mouseClicked(MouseEvent e) {
      this.setBackground(Color.GREEN);
}
```

```
@Override
public void mousePressed(MouseEvent e) {
	//Okno_AWT okienko_m = new Okno_AWT();
	//this.okienko_m = okienko_m;
	mouseX = e.getX();
	mouseY = e.getY();
	mousemsg = "Naciśnięto przycisk myszy w punkcie (" + mouseX +
	", " + mouseY + ")";
	repaint();
}
@Override
public void mouseReleased(MouseEvent e) {
}
@Override
public void mouseEntered(MouseEvent e) {
}
@Override
public void mouseExited(MouseEvent e) {
}
@Override
public void keyTyped(KeyEvent e) {
	k_X=150;
	k_Y=150;
	msg += e.getKeyChar();
	repaint();
}
@Override
public void keyPressed(KeyEvent e) {
	k_X=100;
	k_Y=200;
	//msg = "klawisz wciśnięty ";
	repaint();
+

}

@Override
public void keyReleased(KeyEvent e) {
	k_X=200;
	k_Y=100;
	//msg = "klawisz zwolniony";
	repaint();

}
}
```