8. Aplikacje wielowątkowe w Swingu

Aplikacje wykorzystujące graficzny interfejs użytkownika, a w szczególności aplikacje oparte na bibliotece Swing są aplikacjami wielowątkowymi. Umiejętne wykorzystanie wątków pozwala na tworzenie programów oddzielających obsługę zdarzeń od interfejsu graficznego jak i również efektywną realizację zadań obciążających system.

Wątki w Swingu

Aplikacje zbudowane za pomocą pakietu Swing wykorzystują wątki do efektywnego zarządzania zasobami. Główne wątki, które możemy odnaleźć w każdej aplikacji to:

- wątek główny aplikacji (metoda `main()`, tworzenie GUI),
- wątek obsługujący zdarzenia (ang. EDT – Event Dispatching Thread),
- wątki robocze związane z obsługą złożonych zadań lub też obsługą wejścia-wyjścia.

Większość kodu aplikacji Swing zawarta jest w metodach obsługujących zdarzenia związane z interfejsem użytkownika (odświeżanie ekranu, zdarzenia same w sobie, rysowanie komponentów itp.). Zadania czasochłonne powinny być uruchamiane jako osobne wątki, tak by nie powodować „zawieszeń" interfejsu graficznego (aplikacja cały czas powinna reagować na akcje użytkownika). Wątki robocze (osobne) nie powinny bezpośrednio modyfikować interfejsu graficznego (zawartość komponentów, zmiana ich rozmiaru itp.). Operacje te należy umieszczać bezpośrednio w wątku EDT.

Metody `invokeLater()` i `invokeAndWait()`

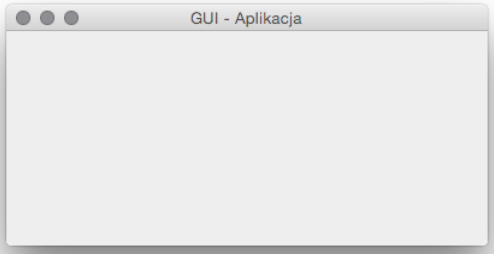
Zgodnie z zasadami, które zostały podane powyżej, wszelkie operacje związane z interfejsem graficznym powinny odbywać się w wątku obsługującym zdarzenia. W klasie `SwingUtilities` istnieją dwie metody umożliwiające poprawne wykonanie tych czynności.

- `invokeLater(Runnable)` – obiekt implementujący interfejs `Runnable` jest kolejkowany w wątku EDT (bezpośrednio zostanie wykonany kod metody `run()`). Metoda zwraca od razu sterowanie do programu,
- `invokeAndWait(Runnable)` – metoda różni się od powyższej jedynie tym, że sterowanie jest oddawane dopiero po wykonaniu metody `run()` przez EDT.

W większości aplikacji Swing wykorzystuje się metodę `invokeLater()`. Jej druga wersja pojawia się głównie przy tworzeniu apletów (metoda `init()`). Przykładowa aplikacja wykorzystująca poprawnie metodę `invokeLater()`

```
import javax.swing.*;

public class AplikacjaGUI {
    public static void main(String[] args) {

       SwingUtilities.invokeLater(new Runnable() {
        public void run() {
          JFrame okno = new JFrame("GUI - Aplikacja");
          okno.setSize(400, 200);
          okno.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
          okno.setVisible(true);
        }
       });
    }
}
```

i jej efekt:



Bardziej złożone aplikacje wymagają częstego komunikowania się z interfejsem użytkownika (pobieranie danych, wyświetlanie danych itp.). Procedura komunikacji sprowadza się do wymiany informacji pomiędzy GUI, a pozostałymi wątkami. Z pomocą przychodzi znowu metoda `invokeLater()`.

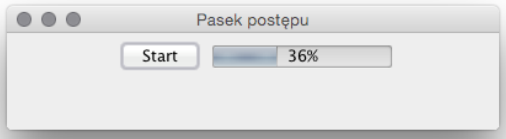
Przykładowa aplikacja – Pasek postępu

Poniższa aplikacja tworzy aplikację zbudowaną z jednego przycisku oraz paska postępu. Celem poniższego kodu jest zaprezentowanie sposobów komunikacji pomiędzy osobnymi wątkami, a interfejsem graficznym. Wszystkie operacje wpływające na GUI realizowane są w wątku EDT, pozostałe w osobnych wątkach (w tym przypadku jeden wątek zmieniający wartości paska postępu).

```
import java.awt.event.*;
import javax.swing.*;

class Okno extends JFrame {

    JButton przycisk;
    JProgressBar pasek;

    public Okno(String nazwa) {
        super(nazwa);

        przycisk = new JButton("Start");
        pasek = new JProgressBar();

        JPanel panel = new JPanel();

        panel.add(przycisk);
        panel.add(pasek);

        przycisk.addActionListener(new ObslugaZdarzen());

        add(panel);

        setSize(400, 100);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }

    private class ObslugaZdarzen implements ActionListener {

        public void actionPerformed(ActionEvent e) {
              pasek.setValue(0);
              pasek.setStringPainted(true);

              new Thread(new Runnable() {
                public void run() {
```

```
                    for (int i = 0; i <= 100; i++) {
                      final int postep = i;

                      SwingUtilities.invokeLater(new Runnable() {
                        public void run() {
                          pasek.setValue(postep);
                        }
                      });

                      try {
                        Thread.sleep(500);
                      }
                      catch(Exception e) { }
                    }
                  }
                }).start();
            }

        }

}

public class PasekPostepu {

    public static void main(String[] args) {
       SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            new Okno("Pasek postępu");
        }
       });
    }
}
```

Efekt działania programu:

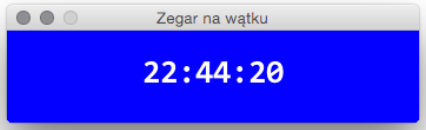


Zadania

1. Przeanalizuj aplikację Pasek Postępu. Sprawdź co się stanie jak kilkukrotnie naciśniesz przycisk Start. Dokonaj zmian w aplikacji tak by tworzył się tylko jeden wątek, albo nie dało się wcisnąć przycisku powtórnie, aż do zakończenia działania paska.

2. Przeanalizuj aplikację Pasek Postępu. Utwórz cztery przyciski start oraz cztery paski postępu o różnym opóźnieniu (różne wartości metody `sleep()`). Sprawdź poprawność działania aplikacji.

3. Chcemy dodać do standardowej kolekcji komponentów Swinga komponent Zegar wyświetlający aktualny czas. Zastanów się jaką klasę będzie najprościej rozszerzyć (może klasę `JLabel`?) oraz jak zaimplementować obsługę wątku odpowiedzialnego za poprawną prezentację czasu.



4. Rozwiń aplikację **Zegar** dodając przyciski **Start**(uruchomienie zegara) oraz **Stop**(zatrzymanie zegara). Dopisz odpowiednie metody do nowego komponentu.

5. Na bazie nowego komponentu **Zegar** utwórz program wyświetlający aktualny czas w czterech różnych miastach (np. Warszawa, Tokio, Moskwa, Nowy York). Wykorzystaj metodę `setTimeZone()` dostępną w klasie `DateFormat`.

6. Napisz aplikację symulującą działanie wygaszacza ekranu z filmu „Matrix". Wykorzystaj wątki i środowisko graficzne. Jakie problemy napotykasz? Wykorzystaj metodę `paintComponent()` oraz prawidłowe wykorzystanie wątków.