

Wieże Hanoi

Przykładowa aplikacja wykorzystująca J2ME Wireless Toolkit

Jarosław A. Miszczak

17 maja 2002 roku

MIDletem nazywamy aplikację stworzoną dla Java 2 Micro Edition (J2ME) Mobile Information Device Profile (MIDP). Sepsyfikację MIDP można znaleźć na stronie <http://java.sun.com/products/midp>. Klasy oraz narzędzia potrzebne do tworzenia i testowania (w tym emulatory urządzeń przenośnych) oprogramowania dostępne są pod adresem <http://java.sun.com/products/j2mewtoolkit/>. Firma NOKIA udostępnia na stronie <http://www.forum.nokia.com/main.html> zestaw rozszerzeń dla pakietu J2ME.

1 MIDProfile

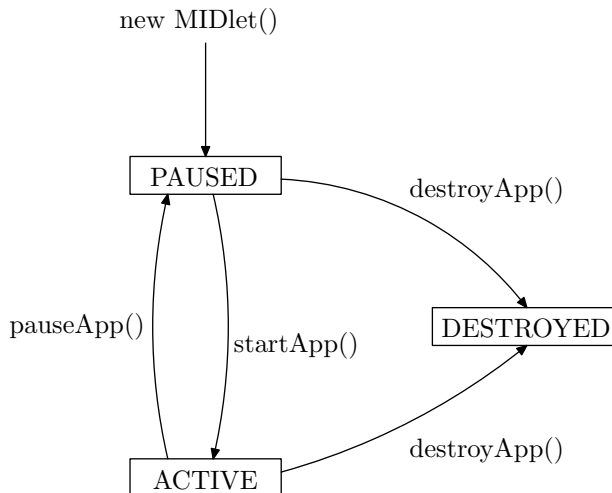
W skład MIDP wchodzi następujące pakiety

- java.lang oraz java.util - niektóre klasy włączone z pakietu Java 2 Standard Edition
- java.io - dostarcza obsługi wejścia/wyjścia
- javax.microedition.io - wsparcie dla obsługi połączeń sieciowych oraz protokołu HTTP
- javax.microedition.lcdui - klasy do obsługi interfejsu użytkownika: obsługa komend, dostęp do wyświetlacza, rysowanie, obsługa formatu PNG, pewne standardowe elementy interfejsu użytkownika (patrz 1.2)
- javax.microedition.midlet - klasa MIDlet zawierająca metody pozwalające na kontrolowanie przez oprogramowanie nadzorujące (ang. *application management software*) cyklu życia MIDletu (patrz 1.1)
- javax.microedition.rms - obsługa składowania informacji (ang. *record management system*).

1.1 Klasa MIDlet

Każdy MIDlet musi rozszerzać klasę javax.microedition.midlet.MIDlet. Klasa ta zawiera funkcje pozwalające oprogramowaniu kontrolującemu MIDlet na uruchomienie i sterowanie aplikacją. Oprogramowanie zarządzające wywołuje odpowiednie funkcje i zmienia stan MIDletu. Dodatkowo każdy MIDlet może samodzielnie dokonywać zmiany swojego stanu i powiadamiać o tym oprogramowanie

zarządzające za pomocą metod `notifyPaused()` oraz `notifyDestroyed()`.



Cykl życia MIDletu

Cykl życia MIDletu rozpoczyna się od utworzenia instancji klasy dziedziczącej po klasie `MIDlet`. Stanem początkowym MIDletu jest stan uśpienia (ang. *paused*). Wywołanie metody `startApp()` powoduje przejście MIDletu w stan aktywności (ang. *activ*) i zarezerwowanie potrzebnych zasobów – utworzenie ekranu, obsługa interfejsu użytkownika. W razie konieczności wstrzymania działania MIDletu wywoływana jest metoda `pauseApp()` i MIDlet przechodzi ponownie w stan uśpienia. Zmiana stanu aktywny \longleftrightarrow uśpiony może dokonywać się wielokrotnie. Jeżeli oprogramowanie kontrolujące aplikację stwierdzi że MIDlet może zostać usunięty z pamięci wywoływana jest metoda `destroyApp(boolean unconditional)` umożliwiającą MIDletowi zakończenie działania. Jeżeli parametr `unconditional` ma wartość `true` MIDlet musi zwolnić zarezerwowane zasoby; w przeciwnym wypadku MIDlet może zgłosić wyjątek `MIDletStateChangeException`.

1.2 Niektóre klasy pakietu `javax.microedition.lcdui`

1.2.1 Klasy `Display` i `Displayable`

Klasa `Display` reprezentuje wyświetlacz oraz urządzenie wejściowe danego systemu. Do każdego MIDletu przypisany jest dokładnie jeden obiekt klasy `Display`. Referencję do niego można uzyskać wywołując metodę `Display.getDisplay(MIDlet m)`

Klasa `Displayable` reprezentuje obiekt który może zostać wyświetlony. Jej bezpośrednimi podklasami są klasy `Canvas` oraz `Screen`.

1.2.2 Klasa `Canvas`

Klasa `Canvas` Pozwala ona na bezpośrednie rysowanie elementów interfejsu użytkownika na wyświetlaczu. Udostępnia ona także metody `keyPressed(int code)`, `keyReleased(int keyCode)` i `keyRepeated(int keyCode)` umożliwiające obsługę klawiatury oraz funkcje obsługi wskaźnika.

1.2.3 Klasa Command

Klasa Command zawiera informacje związane z komendą tj. jej etykietę, typ oraz priorytet. Akcja związana z komendą definiowana jest przez obiekt nasłuchujący (implementujący interfejs `CommandListener`) związany z ekranem do którego przypisana jest komenda.

1.2.4 Klasa Font

Klasa ta reprezentuje obiekt zawierający informacje o czcionce. Klasa Font pakietu `javax.microedition.lcdui`, w przeciwieństwie do klasy `java.awt.Font`, jest klasą finalną (`public final class Font`). W związku z tym aplikacja może jedynie zażądać na podstawie atrybutów dostarczenia czcionki. System dostarcza czcionek najlepiej odpowiadających temu żądaniu.

1.2.5 Klasa Image

Klasa Image służy do przechowywania informacji graficznych. Specyfikacja wymaga obsługi formaty PNG (Portable Network Graphics), w wersji 1.0. Stworzenie obiektu klasy Image możliwe jest poprzez wskazanie źródła (pliku PNG), na podstawie obiektu klasy Graphics stworzonego poprzez wywołanie metody `Image.getGraphics()` bądź na podstawie danych zawartych w tablicy bitowej. Tykło obiekt stworzony z wykorzystaniem tej drugiej metody może być zmieniany. Utworzenie obiektu klasy Image odbywa się poprzez wywołanie metody `createImage(...)` z parametrami zależnymi od sposobu tworzenia obiektu. Aplikacja Wieże Hanoi wykorzystuje dwa sposoby operowania na klasie Image – za pomocą bezpośredniego rysowania oraz poprzez grafikę w formacie PNG.

1.2.6 Podklasy klasy Screen

Klasa Screen jest wspólną nadklasą dla elementów interfejsu użytkownika takich jak list czy formularz. Zachowanie się poszczególnych elementów jest zdefiniowane w jej podklasach.

- Klasa List definiuje ekran zawierający listę wyboru. Wraz z klasą `ChoiceGroup` posiada ona wspólny interfejs `Choice`
- Klasa `TextBox` pozwala na wyświetlanie ekranu z polem tekstowym.
- Klasa Form definiuje ekran składający się z różnego rodzaju elementów takich jak pola tekstowe, etykiety czy obrazki. Nadklasą dla elementów które mogą być dodawane do ekranu rozszerzającego Form jest klasa `Item`.
- Klasa `Alert` reprezentuje ekran który wyświetla pewien zestaw danych i oczekuje pewien okres czasu na wyświetlenie kolejnego ekranu.

1.2.7 Klasa Item

Klasa Item definiuje elementy które mogą być dodane do ekranów rozszerzających klasę Form lub klasę `Alert`. Jej bezpośrednimi podklasami są `ChoiceGroup`, `DateField`, `Gauge`, `ImageItem`, `StringItem` oraz `TextField`.

1.2.8 Interfejs `CommandListener`

Interfejs ten jest wykorzystywany przez aplikacje obsługujące komunikaty użytkownika. Deklaruje on metodę `void commandAction(Command c, Displayable d)` wywoływana gdy na ekranie d zaszło zdarzenie związane z komendą. Obiekt nasłuchujący dla zdarzeń zachodzących na ekranie jest ustawiany za pomocą metody `Displayable.setCommandListener(CommandListener cl)`.

2 Program Wieże Hanoi

Program demonstruje wykorzystanie pakietów `javax.microedition.midlet` i `javax.microedition.lcdui`. Obsługa wież jest zaimplementowana za pomocą klasy `java.util.Stack`. Ruch gracza obsługiwany jest za pomocą metod `Canvas.keyPressed(int code)` oraz `Canvas.getGameAction(int keyCode)`.

2.1 Klasa `MIDlet`

```
import javax.microedition.midlet.MIDlet;
import javax.microedition.lcdui.*;

public class WiezeHanoiMIDlet extends MIDlet{

    private WiezeHanoiTitle title;
    private WiezeHanoiOptions optionScreen;
    private WiezeHanoiGame gameScreen;
    private WiezeHanoiScore scoreScreen;
    public Thread gameThread;
    private Display display;

    public WiezeHanoiMIDlet(){
    }

    //przygotowanie ekranu powitalnego
    protected void startApp(){
        title = new WiezeHanoiTitle(this);
        display = Display.getDisplay(this);
        Displayable current = display.getCurrent();
        if(current == null){
            display.setCurrent(null);
            display.setCurrent(title);
        }
    }

    //pokaż ekran z opcjami
    public void showOptionScreen(){
        optionScreen = new WiezeHanoiOptions(this);
        display.setCurrent(optionScreen);
    }
}
```

```

//pokaż ekran z grą i rozpocznij grę
    public void gameReady(int howHard){
        gameScreen = new WiezeHanoiGame(this,howHard);
        display.setCurrent(gameScreen);
        gameThread = new Thread(gameScreen);
        gameThread.start();
    }
//pokaż ekran końcowy jeżeli gracz zakończył pomyślnie
    public void gameOver(long score){
        scoreScreen = new WiezeHanoiScore(this,score);
        display.setCurrent(scoreScreen);
        try{
            gameThread.join();
        }
        catch(InterruptedException ex){
        }
    }
//pokaż ekran końcowy jeżeli gracz przewał grę
    public void gameOver(String napis){
        scoreScreen = new WiezeHanoiScore(this,napis);
        display.setCurrent(scoreScreen);
        try{
            gameThread.join();
        }
        catch(InterruptedException ex){
        }
    }

    protected void pauseApp(){

    }

    protected void destroyApp(boolean b){
        notifyDestroyed();
    }

    public void exitApp(){
        destroyApp(true);
    }
}

```

2.2 Klasa ekranu powitalnego

```

import javax.microedition.lcdui.*;

class WiezeHanoiTitle extends Canvas
    implements CommandListener, WiezeHanoiConstants{

```

```

private int WIDTH = getWidth();
private int HEIGHT = getHeight();
private Command commandOptions;
private Command commandExit;
private Display display;
private WiezeHanoiMIDlet midlet;
private final String wh[] = {"Wieże", "Hanoi"};
private final Font font = Font.getFont(Font.FACE_PROPORTIONAL,
                                         Font.STYLE_BOLD,
                                         Font.SIZE_LARGE);
private final Font smallFont = Font.getFont(Font.FACE_SYSTEM,
                                              Font.STYLE_PLAIN,
                                              Font.SIZE_SMALL);

public WiezeHanoiTitle(WiezeHanoiMIDlet midlet){
    this.midlet = midlet;
    this.display = display;
    commandOptions = new Command("Start", Command.OK, 1);
    commandExit = new Command("Wyjście", Command.EXIT, 1);
    addCommand(commandOptions);
    addCommand(commandExit);
    setCommandListener(this);
}

public void paint(Graphics g) {
    g.setColor(WHITE);
    g.fillRect(0, 0, WIDTH, HEIGHT);
    g.setColor(BLUE);
    g.setFont(font);
    g.drawString(wh[0], WIDTH/2, HEIGHT/3,
                g.HCENTER | g.TOP);
    g.drawString(wh[1], WIDTH/2, HEIGHT/3+16,
                g.HCENTER | g.TOP);
    g.setFont(smallFont);
    g.setColor(BLACK);
    g.drawString("by J. A. Mischczak", WIDTH/2, HEIGHT/3+35,
                g.HCENTER | g.TOP);
    g.drawString("(c) 2002", WIDTH/2, HEIGHT/3+45,
                g.HCENTER | g.TOP);
}

public void commandAction(Command c, Displayable d){
    if(c == commandOptions) midlet.showOptionScreen();
    if(c == commandExit) midlet.exitApp();
}

```

```

    }

    public void keyPressed(int code){
        midlet.showOptionScreen();
    }
}

```

2.3 Klasa ekranu opcji

```

import javax.microedition.lcdui.*;

public class WiezeHanoiOptions extends Form
    implements CommandListener{

    private Command commandReady;
    private Command commandExit;
    private WiezeHanoiMIDlet midlet;
    private final String[] st = {"Wybierz stopień", "trudność (3-8):"};
    private final TextField field = new TextField("", "3", 1, 2);
    private final String[] opcje =
        {"LEFT - w lewo\n", "RIGHT - w prawo\n", "FIRE - podnieś/upuść\n"};
    private int wynik;
    private boolean empty;

    public WiezeHanoiOptions(WiezeHanoiMIDlet midlet){
        super("Opcje");
        this.midlet = midlet;
        commandReady = new Command("Gotowe", Command.OK, 1);
        commandExit = new Command("Wyjście", Command.EXIT, 1);
        addCommand(commandReady);
        addCommand(commandExit);
        append(st[0]);
        append(st[1]);
        append(field);
        for(int i=0; i<opcje.length; i++)
            append(opcje[i]);
        setCommandListener(this);
    }

    public void commandAction(Command c, Displayable d){
        if(c == commandReady){
            try{
                wynik = Integer.parseInt(field.getString());
            }
            catch(NumberFormatException ex){

```

```

        //jeżeli nic nie zostało wpisane
        //nie można wystawić następnego ekranu
        wynik = 0;
    }
    if(wynik<9 && wynik >2){
        midlet.gameReady(wynik);
    }
}
if(c == commandExit) midlet.exitApp();
}
}

```

2.4 Klasa obsługująca grę

```

import javax.microedition.lcdui.*;
import java.util.Stack;

public class WiezeHanoiGame extends Canvas
    implements CommandListener, WiezeHanoiConstants, Runnable{

    private WiezeHanoiMIDlet midlet;
    private Command exitCommand;
    //rysunek wieży
    private int HEIGHT = getHeight();
    private int WIDTH = getWidth();
    //szerokość wieży
    private int towerWidth = 1;
    //wysokość wieży
    private int towerHeight = HEIGHT/2;
    //pozycje wież
    private int towerPos[] = {15, (WIDTH-towerWidth)/2, WIDTH-(15+towerWidth)};
    //czas rozgrywki
    private long startTime;
    private long currentTime;
    private long time;
    //zmienne potrzebne do gry
    private boolean isGameOver = false;
    private boolean playerWins = false;
    //ile mamy bloczków do przestawiania
    private int numberOfBlocks;
    //bloczek trzymany przez gracza
    private Bloczek mamBloczek;
    //trzy wieże
    private WiezaHanoi wieze[];
    //ustawiamy wszystko na środkowej wieży
    private int aktywnaWieza = 1;

```



```

//gracz nic nie ma
    private int numerBloczka = -1;
//bloczki potrzebne do gry
    private Bloczek[] bloczki;
    private boolean playerMoved;
//inne
    private int n;

    public WiezeHanoiGame(WiezeHanoiMIDlet midlet,int howHard){
        numberOfBlocks = howHard;
        this.midlet = midlet;
//zbuduj wieze
        wieze = new WiezaHanoi[3];
        for(int i=0;i<wieze.length;i++)
            wieze[i] = new WiezaHanoi();
        bloczki = new Bloczek[numberOfBlocks];
//zbuduj bloczki
        for(int i=0;i<numberOfBlocks;i++)
            bloczki[i] = new Bloczek(i);
//ustaw wszystkie na pierwszej wieży
        for(int i=0;i<numberOfBlocks;i++){
            wieze[1].push(bloczki[i]);
            bloczki[i].ktoraWieza = 1;
        }
        wieze[1].ileElementow = numberOfBlocks;
//ustaw obrazki bloczków
        setImages();
//dodaj komendy
        exitCommand = new Command("Zakończ",Command.EXIT,1);
        addCommand(exitCommand);
        setCommandListener(this);
//rozpocznij odliczanie czasu
        startTime = System.currentTimeMillis();
    }

    public void paint(Graphics g){
        g.setColor(WHITE);
        g.fillRect(0,0,WIDTH,HEIGHT);
        g.setColor(BLACK);
//zegarek
        currentTime = System.currentTimeMillis() - startTime;
        g.drawString(currentTime/60000+": "+
            (currentTime/1000)%60,10,10,g.LEFT | g.TOP);
//rysuje wieże
        g.fillRect(towerPos[0],HEIGHT/2,towerWidth,towerHeight);
    }

```

```

        g.fillRect(towerPos[1],HEIGHT/2,towerWidth,towerHeight);
        g.fillRect(towerPos[2],HEIGHT/2,towerWidth,towerHeight);
//po kazdym ruchu wykonaj odmalowanie
    do{
        drawPlayer(g,aktywnaWieza);
        for(int numer=0;numer<numberOfBlocks;numer++){
            drawBloczek(g,numer);
        }
        playerMoved = false;
    }while(playerMoved == true);
}

//zdarzenia
public void commandAction(Command c, Displayable d){
    if(c == exitCommand)
        isGameOver = true;
        midlet.gameOver("Nie udało się ;(");
}

//obsługa ruchu wskaźnikiem oraz operacji na bloczkach
public void keyPressed(int code){
    int j = getGameAction(code);
    switch(j){
        //ruch w lewo
        case LEFT :
            switch(aktywnaWieza){
                case 0 : aktywnaWieza = 2;
                    break;
                case 1 : aktywnaWieza = 0;
                    break;
                case 2 : aktywnaWieza = 1;
                    break;
                default : break;
            } //skaczemy dookoła
            if(numerBloczka != -1) mamBloczek.x = towerPos[aktywnaWieza];
            playerMoved = true; //wykonano ruch
            repaint();
            break;
        //ruch w prawo
        case RIGHT :
            aktywnaWieza = (aktywnaWieza + 1) % 3;
            if(numerBloczka != -1) mamBloczek.x = towerPos[aktywnaWieza];
            playerMoved = true; //wykonano ruch
            repaint();
            break;
        //podniesienie lub upuszczenie bloczka

```

```

case FIRE :
    if(wieze[aktywnaWieza].empty()){//pusta wieża
        if(numerBloczka == -1){
            //nie mam nic do upuszczenia i pobrania
        }else{
            //mogę upuścić cokolwiek
            wieze[aktywnaWieza].push(mamBloczek);
            numerBloczka = -1;
            ((Bloczek)wieze[aktywnaWieza].peek()).y = HEIGHT - 5;
            wieze[aktywnaWieza].ileElementow = 1;
            repaint();
        }
    }else{ //niepusta wieża
        if(numerBloczka == -1){
            //mogę cokolwiek podnieść
            mamBloczek = (Bloczek)wieze[aktywnaWieza].pop();
            mamBloczek.y = HEIGHT/2-5;
            mamBloczek.x = towerPos[aktywnaWieza];
            numerBloczka = mamBloczek.numer;
            wieze[aktywnaWieza].ileElementow -= 1;
            repaint();
        }else if(numerBloczka > ((Bloczek)wieze[aktywnaWieza].peek()).numer){
            //mogę upuścić tylko jeżeli spełniony jest warunek
            wieze[aktywnaWieza].push(mamBloczek);
            numerBloczka = -1;
            ((Bloczek)wieze[aktywnaWieza].peek()).y =
                HEIGHT - 5*(wieze[aktywnaWieza].ileElementow + 1);
            wieze[aktywnaWieza].ileElementow += 1;
            repaint();
        }
    }
    default : break;
}
}
//wątek gry
public void run(){
    Thread current = Thread.currentThread();
    //gra jest zakończona jeżeli wieża 0 i 1 lub 1 i 2 są puste
    //i gracz nic nie trzyma
    while(!isGameOver){
        repaint(10,10,30,20);
        if(isGameOver == (numerBloczka == -1) && (wieze[1].ileElementow == 0)
            && (wieze[0].ileElementow == 0 || wieze[2].ileElementow == 0))
            playerWins = true;
        try{
            current.sleep(100);
        }
    }
}

```

```

        }
        catch(InterruptedException e){
        }
    }
    if(isGameOver && playerWins) midlet.gameOver(currentTime);
}

//stos do obsługi wieży
class WiezaHanoi extends Stack{
    //wszystko co potrzebne jest w klasie java.util.Stack
    public int ileElementow;
}

//zrób obrazki bloczków
public void setImages(){
    for(int i=0;i<numberOfBlocks;i++){
        int w = towerWidth+32-i*4;//odchudzanie bloczków
        bloczki[i].obrazek = Image.createImage(w,5);
        bloczki[i].obrazek.getGraphics().setColor(BLACK);
        bloczki[i].obrazek.getGraphics().fillRect(0,0,w,5);
    }
}

//bloczek
class Bloczek{
    //na której jest bloczek
    public int ktoraWieza = 1;
    //liczba elementów które mogą być pod danym bloczkiem
    public int numer;
    //obrazek przypisany do bloczka
    public Image obrazek;
    //pozycja bloczka
    public int x,y;
    public Bloczek(int num){
        numer = num;
        x = towerPos[ktoraWieza];
        y = HEIGHT - 5*(num + 1);
    }
    public Bloczek(){
    }
}

//rysuje element wskazujący gracza
private void drawPlayer(Graphics g, int nadKtora){
    g.setColor(WHITE);
    g.fillRect(0,towerHeight-15,WIDTH,10);
    g.setColor(BLACK);
    g.fillRect(towerPos[nadKtora],towerHeight-15,towerWidth,5);
}

//rysuje bloczki
private void drawBloczek(Graphics g, int num){

```

```

        g.drawImage(bloczki[num].obrazek,
                    bloczki[num].x, bloczki[num].y,
                    g.TOP | g.HCENTER);
    }
}

```

2.5 Klasa ekranu z wynikami

```

import javax.microedition.lcdui.*;

public class WiezeHanoiScore extends Canvas
    implements CommandListener, WiezeHanoiConstants{

    private int WIDTH = getWidth();
    private int HEIGHT = getHeight();
    private WiezeHanoiMIDlet midlet;
    private Command commandExit;
    private Command jeszczeRaz;
    private long score;
    private String napis = null;

    public WiezeHanoiScore(WiezeHanoiMIDlet midlet, long score){
        this.midlet = midlet;
        commandExit = new Command("Wyjście", Command.EXIT, 1);
        jeszczeRaz = new Command("Powtórz", Command.OK, 1);
        addCommand(commandExit);
        addCommand(jeszczeRaz);
        setCommandListener(this);
        this.score = score;
    }

    public WiezeHanoiScore(WiezeHanoiMIDlet midlet, String napis){
        this.midlet = midlet;
        commandExit = new Command("Wyjście", Command.EXIT, 1);
        jeszczeRaz = new Command("Powtórz", Command.OK, 1);
        addCommand(commandExit);
        addCommand(jeszczeRaz);
        setCommandListener(this);
        this.napis = napis;
    }

    public void paint(Graphics g){
        g.setColor(WHITE);
        g.fillRect(0, 0, WIDTH, HEIGHT);
        g.setColor(BLACK);
        if(napis != null){

```

```

        g.drawString(napis,10,10,g.LEFT | g.TOP);
    }
    else{
        g.drawString("Twój czas:",10,10,g.LEFT | g.TOP);
        g.drawString(score/60000+"":"+score/1000,WIDTH/2,HEIGHT/2,
            g.HCENTER | g.TOP);
    }
}

public void commandAction(Command c, Displayable d){
    if(c == commandExit){
        midlet.exitApp();
    }
    if(c == jeszczeRaz){
        midlet.showOptionScreen();
    }
}
}

```

2.6 Interfejs zawierający stałe

```

public interface WiezeHanoiConstants {
    final int BLACK = 0x000000;
    final int WHITE = 0xFFFFFFFF;
    final int BLUE  = 0x0000FF;
    final int RED   = 0xFF0000;
}

```

Literatura

- [1] Dokumentacja J2ME dostępna na stronie <http://java.sun.com/products/midp/>
- [2] *A brief introduction to MIDP programming* oraz *A brief introduction to MIDP graphics* i dokumentacja dostępna na stronie Forum Nokia <http://www.forum.nokia.com/main.html>