Programowanie wielowątkowe

Wątki i procesy, stany wątku, synchronizacja wątków, priorytety wątków, klasa MediaTracker, data i czas, zdarzenia od myszy - aplet do układania puzzli, podwójne buforowanie - przykład animacji tekstu "scroll".

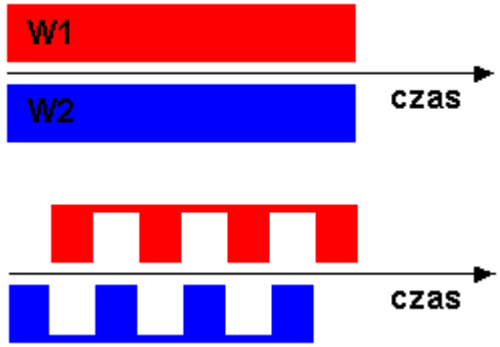
FUW

Wątki i procesy

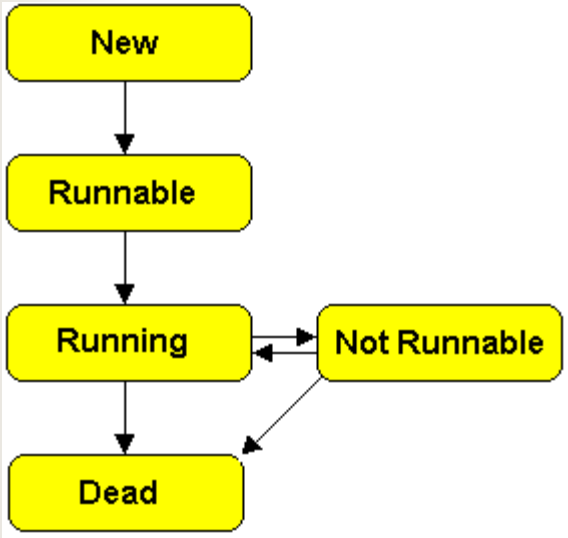
- **Proces to program w trakcie wykonywania. W skład procesu wchodzi uruchomiony program oraz przydzielone mu zasoby systemowe: licznik rozkazów, stos i sekcja danych.**
- **Wątek definiuje się jako pojedynczy, sekwencyjny przepływ sterowania w obrębie procesu.**
- **Wszystkie wątki działające wewnątrz procesu mogą współdzielić przestrzeń adresową, dane oraz zasoby systemowe.**
- **Mówimy, że dwa wątki są współbieżne, jeśli wykonywanie jednego z nich zaczęło się po rozpoczęciu, ale przed zakończeniem działania drugiego.**

Równoległe wykonywanie wątków jest możliwe tylko w systemach wieloprocesorowych.

Na maszynach jednoprocesorowych wątki są realizowane współbieżnie poprzez okresowe przydzielanie cykli procesora.



Stany wątku



Wątek może się znajdować w jednym z kilku stanów:

- **New - nowy wątek;**
- **Runnable - gotowy do uruchomienia;**
- **Running - wykonywany;**
- **Not Runnable - oczekujący;**
- **Dead - zakończony.**

1. **New - występuje bezpośrednio po utworzeniu wątku: Thread thread = new Thread(this);**
2. **Runnable - zachodzi po wykonaniu metody start() gdy wątek czeka na przydzielenie mu czasu procesora**
   thread.start();
3. **Running - wykonywany jest kod wątku umieszczony wewnątrz metody run().**

Stany wątku

4. **Not Runnable - wątek nie jest wykonywany. Przydzielony mu czas procesora jest zwolniony do chwili zajścia jakiegoś zdarzenia. Wątek może być:**
   - **zablokowany - na przykład czeka na zakończenie wykonywania się operacji wejścia-wyjścia;**
   - **uśpiony poprzez wywołanie jego metody sleep()**
     thread.sleep(ms, ns);

     Obudzenie nastąpi po upływie określonej liczby milisekund i nanosekund;
   - **zawieszony za pomocą metody wait() - wznowienie nastąpi, gdy jakiś inny wątek wywoła notify() lub notifyAll();**
   - **zawieszony za pomocą metody suspend() do chwili, gdy jakiś inny wątek wywoła metodę resume() - metody te zostały już zdeprecjonowane i nie powinno się ich używać.**
5. **Dead - wątek zakończył działanie i nie zostanie wznowiony.**

Synchronizacja wątków

- **Synchronizacja jest potrzebna wtedy, gdy dwa wątki żądają dostępu do tego samego zasobu, który może zostać przydzielony tylko jednemu z nich.**
- **Sekcję krytyczną można określić na poziomie instrukcji:**

```
synchronized  (zasób) {
    // instrukcje
}
```

Fragment kodu programu zawierający odwołania do chronionego zasobu nazywany jest sekcją krytyczną.

- **Zwykle jednak synchronizuje się całe metody, umieszczając ich kod w synchronizowanym bloku lub deklarując je jako synchronized:**

```
void mojaMetoda()
{
    synchronized (this)
    {
        // instrukcje
    }
}
```

```
synchronized void mojaMetoda()
{
        // instrukcje
}
```

FUW

Priorytety wątków

- **Każdy wątek posiada swój priorytet, który informuje system operacyjny o jego "ważności".**
- **Priorytet jest liczbą z przedziału MIN_PRIORITY-MAX_PRIORITY. Aktualnie, wartości te wynoszą 1 i 10. Domyślny priorytet nadawany wątkom określa poziom NORM_PRIORITY o wartości 5.**
- **Wątki o wysokich priorytetach zwykle dostają więcej czasu procesora niż wątki o niskich priorytetach. Nie zawsze jednak tak się dzieje. Na przykład, gdy wątek o wysokim priorytecie czeka na jakiś zasób, zostaje on zablokowany, a czas procesora jest przydzielany wątkowi o niższym priorytecie.**
- **Nowy wątek ma taki sam priorytet jak wątek, który go utworzył.**
- **Klasa Thread zawiera metody służące do zmiany priorytetu.**
  - **Do ustawiania priorytetu wątku służy metody setPriority() z argumentem w postaci żądanego priorytetu.**
  - **Do pobrania aktualnego priorytetu wątku służy metoda getPriority().**

MediaTracker

Duże obrazki pochodzące z plików są wyświetlane stopniowo. Aby temu zapobiec należy skorzystać z metod klasy MediaTracker, które pozwalają monitorować proces wczytywania obrazka, a także nadzorować ładowanie wielu obrazków jednocześnie. Schemat postępowania jest następujący:

1. **Na poziomie klasy tworzymy referencje:**

```
MediaTracker tracker;
Image image;
```

2. **W metodzie init() tworzymy obiekt klasy MediaTracker**

```
tracker = new MediaTracker tracker(this);
image = getImage(getCodeBase(), "plik.gif");
tracker.addImage(image, 0);  //0 - nr identyfikacyjny obrazka
```

3. **W metodzie run() zawieszamy wyświetlanie obrazka do chwili jego pełnego załadowania**

```
try { tracker.waitForID(0);}
catch (InterruptedException e) {}
```

4. **W metodzie paint() wyświetlamy obrazek po załadowaniu**

```
if ( tracker.checkID(0) ) g.drawImage(image, 0, 0, this);
```

Przykład - MediaTracker

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

public class Track extends Applet implements Runnable
{
 Image image; Thread loader; MediaTracker tracker;
 int w,h;

 public void init() {
  tracker = new MediaTracker(this);
  image = getImage(getCodeBase(), "kulki.gif");
  tracker.addImage(image, 0);
  w = getSize().width;
  h = getSize().height;
 }
 public void start() {
  loader = new Thread(this);
  loader.start();
 }
```

Przykład - MediaTracker

```
 public void stop() {
  loader = null;
 }

 public void run() {
  try {
    tracker.waitForID(0);
  } catch (InterruptedException e) {}
  repaint();
 }

 public void paint(Graphics g) {
   if (tracker.checkID(0) )
    g.drawImage(image, 0, 0, w, h, this);
   else
    g.drawString("Ładowanie grafiki", 40, 40);
 }
}
```

Data i czas

- **Pakiet java.util zawiera dwie klasy związane z datą i czasem systemowym:**
  - Calendar
  - GregorianCalendar
- **Obiekt reprezentujący kalendarz z aktualną datą i czaseë można więc utworzyć na dwa sposoby:**
  - `Calendar calendar = Calendar.getInstance();`
  - `GregorianCalendar calendar = new GregorianCalendar ();`
- **Pobranie daty i czasu umożliwia metoda get():**

```
int dzien = calendar.get(Calendar.DAY_OF_MONTH);
int miesiac = calendar.get(Calendar.MONTH)+1;
int rok = calendar.get(Calendar.YEAR);
int godz = calendar.get(Calendar.HOUR_OF_DAY);
int min = calendar.get(Calendar.MINUTE);
int sek = calendar.get(Calendar.SECOND);
```

Przykład - zegarek cyfrowy

```
import java.awt.*;
import java.applet.Applet;
import java.util.Calendar;

public class Zegarek extends Applet
implements Runnable {
   String string = "";
   Thread thread = null;
   Calendar calendar;
   boolean stopFlag;

   public void start() {
      if (thread == null) {
         thread = new Thread(this);
         stopFlag = false;
         thread.start();
      }
   }

   public void stop() {
      stopFlag = true;
      thread = null;
   }
```

Przykład - zegarek cyfrowy

```
    public void run() {
       while (thread != null) {
         if ( stopFlag ) break;
         calendar = Calendar.getInstance();
         int g = calendar.get(Calendar.HOUR_OF_DAY);
         int m = calendar.get(Calendar.MINUTE);
         int s = calendar.get(Calendar.SECOND);
         string = Integer.toString(g);
         string += ":"+Integer.toString(m);
         string += ":"+Integer.toString(s);
         try {
           Thread.sleep(1000);
         } catch (InterruptedException e){}
         repaint();
       }
    }

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

FUW

Modyfikacja

Napisz funkcję obliczającą ile dni upłynęło od początku roku do podanej daty. Oto nagłówek funkcji:

public static int odpoczatku (int dd, int mm, int rr)

- **Zastosuj uproszczoną metodę sprawdzania, czy podany rok jest rokiem przestępnym, badając jedynie podzielność przez 4.**
- **Zmodyfikuj aplet tak, aby podawał aktualną datę oraz liczbę dni, jaka upłynęła od początku roku.**

Rozwiązanie

```
public static int odpoczatku (int dd, int mm, int rr)
{
 int wynik=dd;
 switch (mm-1)
 {
  case 11: wynik+=30;
  case 10: wynik+=31;
  case 9: wynik+=30;
  case 8: wynik+=31;
  case 7: wynik+=31;
  case 6: wynik+=30;
  case 5: wynik+=31;
  case 4: wynik+=30;
  case 3: wynik+=31;
  case 2: if (rr% 4==0) wynik+= 29; else wynik+= 28;
  case 1: wynik+=31;
 }
 return wynik;
}
```

Ponieważ słowo kluczowe **break** zostało pominięte, zatem po wybraniu instrukcji case odpowiadającej podanej wartości miesiąca, zostaną zsumowane dodatkowo wszystkie liczby do końca instrukcji switch - czyli dodamy ilości dni dla wszystkich poprzednich miesięcy.

Ćwiczenie - zdarzenia od myszy

Aplet do układania puzzli wykorzystuje obrazek o wymiarach 300 na 300 pix podzielony na cztery części.

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

public class Puzzle extends Applet implements MouseListener
{
   Image[] a = new Image[4];
   int[] x = new int[4]; int[] y = new int[4];
   int pierwszy, drugi;

   public void init()
   {
     a[0] = getImage(getCodeBase(), "rys0.gif");
     a[1] = getImage(getCodeBase(), "rys1.gif");
     a[2] = getImage(getCodeBase(), "rys2.gif");
     a[3] = getImage(getCodeBase(), "rys3.gif");
     x[0]=0;   y[0]=0;
     x[1]=151; y[1]=0;
     x[2]=0;   y[2]=151;
     x[3]=151; y[3]=151;
     tasuj();
     addMouseListener(this);
     repaint();
   }
```

Ćwiczenie - c.d.

```
public void tasuj()
{
  for (int i=0; i<4; i++ )
  {
    int los = (int)(Math.random()*4);
    Image pom = a[los];
    a[los] = a[i];
    a[i] = pom;
  }
}

public void mousePressed(MouseEvent e)
{
  int x = e.getX();
  int y = e.getY();
//określenie numeru obrazka na podstawie jego współrzędnych
  pierwszy = x/150 + y/150*2;
}
```

Ćwiczenie - c.d.

```
  public void mouseReleased(MouseEvent e)
  {
    int x = e.getX();
    int y = e.getY();
    drugi = x/150 + y/150*2;
    Image pom = a[drugi];
    a[drugi] = a[pierwszy];
    a[pierwszy] = pom;
    repaint();
  }

  public void mouseEntered(MouseEvent e) {}
  public void mouseExited(MouseEvent e) {}
  public void mouseClicked(MouseEvent e) {}

  public void paint(Graphics g)
  {
    for (int i=0; i<4; i++)
      g.drawImage(a[i], x[i], y[i], 150, 150, this);
  }
}
```

Podwójne buforowanie

- **W animacjach często można zauważyć efekt migotania, który wynika z tego, że w krótkim odstępie czasu obraz jest rysowany i ścierany w jednym miejscu ekranu. Powstają też zniekształcenia spowodowane opóźnieniem pomiędzy rozpoczęciem wykonywania metody paint() a czasem rysowania obrazu w poprzednim jej wywołaniu. Sposobem wyeliminowania tego efektu jest zastosowanie techniki podwójnego buforowania.**
- **W technice tej posługujemy się dwoma obrazami. Jeden z nich jest wyświetlany na ekranie, a drugi jest rysowany w buforze. Kiedy obraz pozaekranowy jest gotowy, następuje zamiana obrazów - stary obraz jest zastępowany nowym (kopiowanym z bufora) bez konieczności czyszczenia ekranu.**
- **Dodatkowo należy przedefiniować metodę update(), aby uniknąć migotania spowodowanego wymazywaniem panelu. Domyślna metoda update() najpierw wypełnia panel kolorem tła, a dopiero potem wywołuje paint(). W tym przypadku jest to niepotrzebne, gdyż cały panel jest pokrywany obrazem pozaekranowym.**

Przykład - scroll

Scrolle są prostymi animacjami, polegającymi na przesuwaniu tekstu w pionie lub poziomie.

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

public class Scroll extends Applet implements
Runnable
{
  Thread thread;
  String napis;
  Font font;
  Image image;
  Graphics gc;
  int x;   // aktualna pozycja scrolla
  int dl;  // długość scrolla
  int w;   // szerokość apletu
  int h;   // wysokość apletu
```

Przykład - scroll

```
public void init()
{
  thread = null;
  String strParam = getParameter("Tekst");
  if (strParam == null)
  {
    napis="WPISZ TEKST SCROLLA";
  } else napis=strParam;

  w=getSize().width;
  h=getSize().height;
  x=w;

  image=createImage(w,h);
  gc = image.getGraphics();
  gc.fillRect(0, 0, w, h);
  font = new Font("Helvetica",Font.BOLD, h/8);
  gc.setFont(font);
  FontMetrics fm = gc.getFontMetrics();
  dl = fm.stringWidth(napis);
}
```

Tekst jest przekazywany apletowi z pliku HTML za pomocą znacznika <param>.

Metoda getParameter( ) -przyjmuje argument w postaci nazwy parametru (podanej w pliku HTML) - zwraca wartość parametru.

FUW

Przykład - scroll

```
public void start()
{
  if (thread == null) {
    thread = new Thread(this);
    thread.start();
  }
}

public void stop()
{
  if (thread != null){
    thread = null;
 }
}

public void run()
{
  while(thread!=null)
  {
    gc = image.getGraphics();
    x -=2;
    if(x < -1*dl) x = w;
```

Metoda getGraphics() przypisuje obraz do zmiennej gc dzięki czemu możemy na utworzonym obrazie rysować tak, jakbyśmy robili to na ekranie

FUW

Przykład - scroll

```
      gc.setColor(Color.orange);
      gc.fillRect(0,0,w,h);       //czyszczenie bufora
      gc.setColor(Color.blue);
      gc.setFont(font);
      gc.drawString(napis,x,h/8);
      repaint();   //lub paint(getGraphics());
      try {
        Thread.sleep(50);
      } catch(InterruptedException e){}
    }
  }

  public void paint(Graphics g){
    g.drawImage(image,0,0,this);
  }

  public void update(Graphics g){
    paint(g);
  }
}
```

Kod HTML

```
<applet code = "Scroll.class" width="300" height="300">
<param name="Tekst" value="Java">
</applet>
```

FUW