

- **Swing** je štandardná Java knižnica na vývoj grafických používateľských rozhraní
- bežná jednoduchá aplikácia narábajúca s klávesnicou a konzolou beží v jednom vlákne
- to platí aj pre jednoduché Swing aplikácie
- v zložitejších prípadoch však musíme vlákna zvládnuť



Komponenty Swingu vo všeobecnosti nie sú thread-safe!

- je to zámerom návrhárov
- historická skúsenosť: dodržiavanie automatickej thread-safety v odvodených komponentoch vyžaduje špeciálnu pozornosť
- udalosti na komponentoch sú vyvolávané v predvídateľnom poradí
 - inak je veľmi náročné ladenie
- automatická správa zámkov môže byť zložitá

Swingová aplikácia a vlákna

- **hlavné vlákno** – naštartuje ostatné
- **initial threads** – iniciálne vlákna. pripraví okná, nainicializujú ich
- **event dispatch thread (EDT)**
 - používateľ svojim klikaním, písaním, pohybom myšou vyvoláva udalosti,
 - tie sú radení do frontu
 - EDT ich vyberá z frontu, spracováva a prekresľuje užívateľské rozhranie





Základné problémy pri práci so Swingom

<http://today.java.net/pub/a/today/2007/08/30/debugging-swing.html>

1. V EDT nespúšťajte dlhotrvajúce operácie!
2. Stav užívateľského rozhrania nemeňte inde než v EDT!





Dlhotrvajúce operácie nie v EDT!

Dlhotrvajúce používateľské operácie nesmú bežať v EDT!

- **blokovali** by používateľské rozhranie. Prečo?
 - dlhotrvajúca akcia zablokuje frontu
 - EDT ju začne vykonávať, lenže ostatné akcie čakajú!
 - prestanú sa prekresľovať ovládacie prvky
 - rozhranie "vytuhne" – sivé okno!
 - užívateľ má pocit, že aplikácia *zamrzla*
 - *začne zbesilo klikať, lenže tým si nepomôže, lebo opäť generuje udalosti radené na koniec frontu!*



Dlhotrvajúce operácie nie v EDT!

```
// kód v nejakom ActionListeneri
void actionPerformed(ActionEvent e) {
    while(true) {
        // cyklí sa do nemoty
    }
}
```

- užívateľ klikne na gombík, vyvolá udalosť
- tá sa zaradí na koniec fronty v EDT
- lenže potom UI vytuhne, pretože nekonečný cyklus zabráni vykonávaniu ďalších udalostí vo fronte

Dlhotrvajúce operácie nie v EDT!

```
Runnable task = new Runnable() {  
    public void run() {  
        while(true) {};  
    }  
}
```

```
// kód v nejakom ActionListeneri  
void actionPerformed(ActionEvent e) {  
    Thread infiniteThread = new Thread(task);  
    • infiniteThread.start();  
}
```

- kód beží v separátnom vlákne, neblokuje EDT
- stačí pre jednoduché prípady
- zložitejšie riešime inak (vid' neskôr)



Stav UI nemeňte inde než v EDT!

- čo ak dlhotrvajúca akcia bežiaca v inom vlákne chce meniť stav používateľského rozhrania?
- napr. vypisovať hlásenia do textového políčka?

```
Runnable task = new Runnable() {  
    public void run() {  
        int i = 0;  
        while(true) {  
            i++;  
            jTextField.setText(i + "-ty beh.");  
        }  
    }  
}
```

```
// kód v nejakom ActionListeneri  
void actionPerformed(ActionEvent e) {  
    Thread infiniteThread = new Thread(task);  
    infiniteThread.start();  
}
```

nekorektné
použitie!

Stav UI nemeňte inde než v EDT!

- zmena UI z iného vlákna než EDT môže spôsobovať **problémy!**
- deadlocky, zvláštne chovanie či vzhľad...

Zmena stavu komponentov sa musí diať v EDT!

- ako však dosiahnuť vykonanie kódu v EDT, ak sme v inom vlákne?





Stav UI nemeňte inde než v EDT!

```
SwingUtilities.invokeLater(new Runnable() {  
    public void run() {  
        spustiAkciuVEDT();  
    }  
});
```

- operácia sa spustí **asynchrónne** v EDT
- kód v **Runnable** sa zaradí na koniec fronty udalostí a spustí sa vtedy, keď sa odbavia čakajúce udalosti
- v podstate odošleme kus kódu do fronty EDT, aby sa tam vykonal.
- takto musíme vykonávať kód aktualizujúci UI



Stav UI nemeňte inde než v EDT!

- klasický príklad z úvodu do práce v Swingu

```
public class SwingTest {  
    public static void main(String[] args) {  
        JFrame frame = new JFrame();  
        JButton button = new JButton("OK");  
        button.addActionListener(new ActionListener() {  
            public void actionPerformed(ActionEvent e) {  
                System.out.println(Thread.currentThread());  
            }  
        });  
        frame.add(button);  
        frame.pack();  
        frame.setVisible(true);  
    }  
}
```

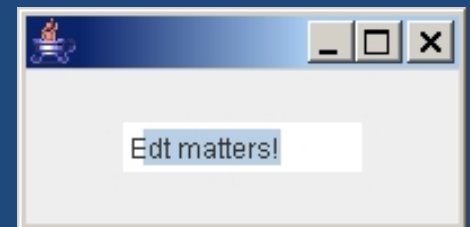
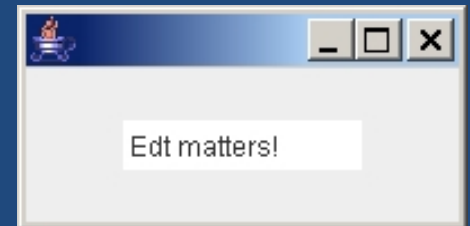
- Aplikácia pobeží zvyčajne v poriadku
- Je tu však chyba!
- Kód v modrých boxoch sa vykonáva v *hlavnom vlákne*. Musí však byť v EDT!
- Tuto chybu nie je až taká zjavná.

Stav UI nemeňte inde než v EDT!

- v prípade komplexnejších zmien v inom vlákne než EDT nastávajú divné chyby!

```
public class zlýKód{  
    public static void main(String args[]) {  
        vytvorGui();  
    }  
  
    private static void createGui() {  
        // tento kód MUSÍ bežať v EDT.  
        // Teraz beží v hlavnom vlákne.  
    }  
}
```

pokus o zmenu
vybraného textu raz
funguje, raz nie





Stav UI nemeňte inde než v EDT!

- **riešenie:** korektné spustenie UI
- akcie v *main()* metóde spúšťame pomocou *invokeLater()*
- nasledovný kód sa spustí v EDT, po inicializácii aplikácie a hlavného okna

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

- čo ak chce úloha priebežne aktualizovať GUI?
- čo ak úloha vracia nejaký výsledok?
- čo ak chceme zrušiť bežiacu úlohu?
- čo ak chceme z GUI zistiť, či úloha ešte stále beží?

Riešenie:

SwingWorker!



- **SwingWorker** je trieda, ktorá rieši všetky podobné problémy
- zabudovaná od verzie 6
- možno stiahnuť a použiť aj vo verzii 5
- základné použitie: oddedíme a prekryjeme metódy
- príklad identický s predošlým

```
SwingWorker<Void, Void> w = new SwingWorker<Void,Void>() {  
    protected Void doInBackground() {  
        hľadajNajvyššiePrvočíslo();  
        return null;  
    }  
}  
w.execute();
```

- Void s veľkým V! (to nie je preklep)
- po dobehnutí musíme vrátiť nejakú hodnotu (null sa hodí)

- kód v metóde `doInBackground()` sa vykoná asynchrónne mimo EDT
- `SwingWorker` je generická trída
 - prvý `Void` zodpovedá návratovej hodnote z `doInBackground()`
 - teda operácii, ktorá beží na pozadí
 - druhý `Void` zodpovedá návratovej hodnote z metód vracajúcich čiastkové výsledky
- dôležité metódy:
 - `doInBackground()` – vykoná sa asynchrónne a môže vrátiť hodnotu
 - `done()` – vykoná sa po dobehnutí `doInBackground()` vo vlákne EDT, môže vrátiť nejakú hodnotu
 - `done()` môže získať výsledok z `doInBackground()` pomocou metódy `get()`



SwingWorker

```
SwingWorker<ImageIcon[], Void> swingworker
= new SwingWorker<ImageIcon[], Void>()
{
    protected ImageIcon[] doInBackground() throws Exception {
        ImageIcon[] icons = ... načítaj z internetu
        return icons
    }

    protected void done() {
        try {
            ImageIcon[] icons = get();
            swingworker.execute();

            aktualizujGUI(ikony);
        } catch (InterruptedException e) {
            // nerob nič
        } catch (ExecutionException e) {
            e.printStackTrace();
        }
    }
}
```

```
swingworker.execute();
```

- niekedy chceme sledovať priebeh
- použitím metódy `publish(T)` vieme odosielať priebežné výsledky do vlákna EDT
 - parameter metódy je identický s druhým generickým parametrom pri vytváraní `SwingWorker`a
- výsledky si vieme vyzdvihnúť v metóde `process()`
- `process(List<T> výsledky)` je vykonávaná v EDT
 - do parametra dostaneme niekoľko výsledkov
 - tie sú zoskupované z viacerých volaní metódy `publish()`, kvôli efektivite



SwingWorker

```
SwingWorker<Void, Integer> task = new SwingWorker<Void, Integer>() {
    protected void doInBackground() throws Exception {
        File file = new File("track.mp3");
        double fileLength = file.length();
        for (int i = 0; i < fileLength; i++) {
            int percents = (int) ((i / fileLength) * 100);
            publish(percents);
        }
        return null;
    }

    protected void process(List<Integer> chunks) {
        // v liste máme viaceré percent, zaujíma nás len posledná
        progressBar.setValue(chunks.get(chunks.size() - 1));
    }

    protected void done() {
        progressBar.setValue(100);
    }
};
task.execute();
```



SwingWorker a prerušenie úloh

- podobne ako vlákna je možné prerušovať úlohy
- `swingWorker.cancel(boolean prerušenieVláknaPovolené)` – ukončí úlohu
 - parameter `true`: pokúsi sa prerušiť vlákno
 - `false`: nechá dobehnúť úlohu
- v `doInBackground()` môžeme kontrolovať, či `isCancelled() == true`
 - ak áno, úloha bola prerušená a mali by sme skončiť



SwingWorker a prerušenie úloh

```
SwingWorker<Void, Integer> task
= new SwingWorker<Void, Integer>()
{
    protected Void doInBackground() throws Exception {
        while(!isCancelled()) {
            System.out.println(new Date());
        }
        return null;
    }
};
```

- Úlohu môžeme prerušiť:

```
task.cancel(false);
```

- úloha musí spolupracovať pri ukončení (nemožno ju odstrelit)
- zabezpečíme to testovaním, či `isCancelled()`



SwingWorker a prerušenie úloh

```
task.cancel(true);
```

- pokúsi sa **interrupt()**núť vlákno vykonávajúce úlohu
- ak v úlohe spíme, vyhodí sa **InterruptedException**
 - spať môžeme cez `Thread.sleep()`
 - alebo cez `TimeUnit.[jednotka].sleep()`
- ak nespíme, ani netestujeme `isInterrupted()`, vlákno je **neodstreliteľné**
 - to je užívateľský veľmi neprívetivé