KONKURENTNÉ PROGRAMOVANIE

Cvičenie 9 : Swing a JavaFx

Používateľ chce svižné GUI

- Responsiveness
  - Okamžitá ~ 100ms
    - Klik na tlačidlo spôsobí okamžité stlačenie
  - Bezprostredná ~ 0,5s – 1s
    - Doba medzi odoslaním a prijatím príkazu
  - Priebežná ~ 2s – 5s
    - Perióda informovanosti o progrese
  - Pútavá ~ do 10s
    - Maximálna doba pozornosti používateľa

Typicky pomalé operácie

- Sieťové operácie
  - Sokety, ťahanie z internetu
- Kopírovanie súborov
- Spracovanie videa
- Refresh zdrojov

Všetko nad
500 ms

Princíp fungovania aplikácii s GUI

- Vlákno EDT (event dispatch thread)
  - Spracovanie udalostí v GUI
    - Klik na button, posun slidera, pozícia myšky, výber položky, editácia textu, ...
    - Všetky listenery Swingových komponentov
  - Zmena vzhľadu GUI
    - Pridanie a dobratie komponentov
  - Refresh GUI
    - Vizualizácia zmeny vlastností komponentov (posun progress baru, zvýraznenie vybraného komponentu,...)
- Na všetko ostatné iné vlákna

Princíp fungovania aplikácii s GUI

- Vlákno EDT má rad úloh
  - Úlohy na zmenu GUI, spracovanie udalostí komponentov
  - Vykonáva ich postupne
  - Nové úlohy na koniec
- V EDT beží nekonečný cyklus
  - Vyberie úlohu z radu
  - Vykoná ju
  - Prekreslí GUI

Zásada práce v okienkovej apke

- V EDT nevykonávajte dlhotrvajúce operácie
  - Dlhotrvajúce úlohy blokujú rad úloh pre GUI
  - Používateľ má pocit, že aplikácia vytuhla
    - začne zbesilo klikať, lenže tým si nepomôže, lebo iba generuje udalosti (úlohy) radené na koniec radu!
- Stav GUI komponentov nemeňte inde než v EDT
  - Komponenty Swingu väčšinou nie sú thread-safe
  - Viac vlákien by mohlo volať metódy toho istého komponentu a spôsobiť nekonzistentnosť jeho stavu, deadlocky, zvláštne chovanie,..

Zaslanie úlohy do radu EDT

- Ak som v inom vlákne ako v EDT:
  - SwingUtilities.**invokeLater**(Runnable úloha)
    - Zaradí úlohu na koniec radu EDT a ďalej sa nestará
  - SwingUtilities.**invokeAndWait**(Runnable úloha)
    - Zaradí úlohu na koniec radu EDT a čaká na jej vykonanie (zatiaľ zaspí)
- V JaveFx:
  - Platform.*runLater*(Runnable úloha)
- Tieto úlohy sa spustia, až keď sa vybavia úlohy pred tým

Korektné spustenie GUI aplikácie

- main sa nespúšta v EDT, ale v hlavnom vlákne

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

- Metóda vytvorGUI() bude spustená v EDT

Zadanie

- Stiahnite si z GitHubu poslednú verziu:
  - https://github.com/PeterGursky/kopr2017
- nasledujúce zadanie:
  - sk.gursky.kopr.cviko07.zadanie
- Zrealizujte korektné spustenie GUI

Dlhé úlohy iniciované v EDT

- Ak chceme v EDT iniciovať dlhý výpočet, musíme ho spustiť v samostatnom vlákne (Thread), ale Runnable úlohy si žijú vlastným životom
  - Nevedia vrátiť výsledok
  - Nevieme ich zrušiť
  - Nevieme odchytiť výnimky

Dlhé úlohy iniciované v EDT

- Štandardný exekútor nám nepomôže
  - Nemôžme zaspať a čakať na výsledok cez get(), lebo zaspí celé EDT!
- Opačná filozofia: Keď dlhá úloha skončí, iniciuje úlohu pre EDT
  - Čo však, ak dlhá úloha vyhodí výnimku a chcem ju spracovať pre EDT?
  - Ak dlhú úlohu chcem z EDT ukončiť (napr. používateľ stlačí cancel), ako to tej úlohe poviem?
- Riešenie pre všetko menované:
  - SwingWorker (od Javy 6)
    - Runnable a Future dohromady s jedným worker vláknom mimo EDT – samostatne bez exekútora, teda správcu vlákien

SwingWorker<A,B>

- Dedíme a prekrývame metódy(aspoň prvú)
- A **doInBackground**() throws Exception
  - metóda spustená v samostatnom (worker) vlákne mimo EDT
- void **done**()
  - metóda zaslaná do radu EDT po dobehnutí doInBackground
  - môžeme získať výsledok vrátený z doInBackground cez **get**(), prípadne takto získať výnimku vyhodenú z doInBackground (zabalenú v ExecutionException)
- void **process**(List<B> updates)
  - metóda zaslaná do radu EDT ak doInBackground spustil **publish**(B... update)

Minimalistický príklad

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

- doInBackground() musí niečo vrátiť
  - ak nemáme čo, uvedieme ako návratový typ java.lang.Void
  - vrátime null
- execute() spúštame vždy len raz
  - SwingWorker je na jedno použitie

Príklad s návratovou hodnotou

```
JLabel label;
...
SwingWorker<String, Void> w = new SwingWorker<>() {
  protected String doInBackground() {
    return findTheMeaningOfLife();
  }
  protected void done() {
    try {
        label.setText(get());
    } catch (Exception ignore) {
    }
  }
}
w.execute();
```

spustené vo worker vlákne

spustené v EDT vlákne

SwingWorker<A,B>

- Dedíme a prekrývame metódy(aspoň prvú)
- A **doInBackground**() throws Exception
  - metóda spustená v samostatnom (worker) vlákne mimo EDT
- void **done**()
  - metóda zaslaná do radu EDT po dobehnutí doInBackground
  - môžeme získať výsledok vrátený z doInBackground cez **get**(), prípadne takto získať výnimku vyhodenú z doInBackground (zabalenú v ExecutionException)
- void **process**(List<B> updates)
  - metóda zaslaná do radu EDT ak doInBackground spustil **publish**(B... update)

publish(T…) → process(List<T>)

- publish(T…)
  - Mimo EDT v doInBackground
  - Hádžeme priebežné výsledky pre EDT
- process(List<T>)
  - V rámci EDT
  - Vyzdvihujeme priebežný výsledok
  - Kvôli efektivite kumuluje výsledky jedného alebo viacerých volaní publish()

```
publish("1", "2");
publish("3");
publish("4","5","6");
```

```
process(["1", "2","3","4","5","6"])
```

Príklad s návratovou hodnotou

```
JProgressBar progressBar;
...
SwingWorker<Void, Integer> w = new SwingWorker<>() {

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

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

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

spustené vo worker vlákne

spustené v EDT vlákne

Rušenie SwingWorkera

- Prerušiť vieme iba vlákno workera, ktoré vykonáva metódu doInBackground()
- swingWorker.cancel(boolean interruptnúť)
  - pošle interrupt na worker vlákno, ak na vstupe je true
  - Rovnaký princíp ako pri iných úlohách v cudzích vláknach
    - Úloha v doInBackground() musí predpokladať, že ju niekto bude rušiť
    - Vieme testovať prerušený stav cez isCancelled() == true
    - Blokované operácie vyhadzujú InterruptedException

Možné výnimky vyhodené z doInBackground()

```
protected void done() {
  try {
    get();
  } catch (InterruptedException e) {
    // úloha bola prerušená, nemá zmysel robiť nič
  } catch (CancellationException e) {
    /* úloha bola zrušená cez cancel()
       nemá zmysel robiť nič
    */
  } catch (ExecutionException e) {
    throw new RuntimeException(
         "Chyba pri vykonavani ulohy.", e.getCause());
  }
}
```

Zadanie - pokračovanie

- Zrealizujte kontrolu gramatiky cez SwingWorker

JavaFx

- Namiesto SwingWokrera máme **Task** a Service

```
Task<Boolean> spellCheckTask = new Task<Boolean>() {
   protected Boolean call() throws Exception { // in background
      SpellChecker spellChecker = new SpellChecker();
      List<SpellChecker.SpellcheckBoundary> kontrola = spellChecker.check(newValue);
      if (kontrola.isEmpty())
         return true; // korektny text
      else
         return false; // zly text
   }
};

Thread th = new Thread(spellCheckTask);
th.start();
```

JavaFx

- Namiesto SwingWokrera máme Task a **Service**

```
Service<Boolean> spellCheckService = new Service<Boolean>() {
   protected Task<Boolean> createTask() {
      return new Task<Boolean>() {
         protected Boolean call() throws Exception {// in background
            ...
         }
      };
   }
};

spellCheckService.start();
```

Service aj Task

- metódy na registráciu obslužných úloh:
  - setOnCancelled(event)
  - **setOnFailed(event)**
  - setOnRunning(event)
  - setOnScheduled(event)
  - **setOnSucceeded(event)**

JavaFx

- Získanie výsledku v GUI po skončení úlohy:

```
Task<Boolean> spellCheckTask = new Task<Boolean>() {
   protected Boolean call() throws Exception {   // in background
      ...
   }
};

spellCheckTask.setOnSucceeded(new EventHandler<WorkerStateEvent>() {
   public void handle(WorkerStateEvent event) {  // in EVT
      Boolean isOK = spellCheckTask.getValue();
      redGreenPane.setGreenState(isOK);
   }
});
```

Service aj Task

- Niektoré „observable“ premenné, ktoré vieme použiť ako model komponentov, resp. na nich odchytávať udalosti zmeny aj vo vlákne GUI:
  - ReadOnlyStringProperty messageProperty()
    - meníme cez updateMessage(String message)
  - ReadOnlyDoubleProperty progressProperty()
  - ReadOnlyDoubleProperty totalWorkProperty()
  - ReadOnlyDoubleProperty workDoneProperty()
    - všetky 3 meníme cez updateProgress(double workDone, double max)
  - ReadOnlyStringProperty titleProperty()
    - meníme cez updateTitle(String title)
  - ReadOnlyObjectProperty<V> **valueProperty()**
    - meníme cez **updateValue(V value)**
  - ReadOnlyObjectProperty<Worker.State> stateProperty()
  - ReadOnlyObjectProperty<Throwable> exceptionProperty()