

Nedokončený fragment !!!

Grafické prostředí

JavaFX/CZ>

Napřed myslet, potom programovat

S programovacím jazykem Java se úplně změnil (resp. měl by se úplně změnit) způsob, jakým programátor pracuje. Java zavádí tak silnou vazbu na technologii objektového programování, že objektové postupy prostě nelze ignorovat.

Ukážeme si to na jednoduchém příkladu. Mějme naprogramovat model křižovatky, na kterém budou svítit semaforey, jako kdyby řídily dopravu. Příklad takové křižovatky ukazuje obrázek

<..\files\03-21 Křižovatka 1.png>

Klasický programátorský postup by byl, že bychom nejprve vytvořili čtyři zelené plochy (trávníky), kolem kterých bychom z kroužků, obdélníků a oválů začali vytvářet semaforey. Ti tvořivější by si vytvořili jen jeden semafor a umístili by ho ve čtyřech různých polohách.

Načež by začli programovat logiku křižovatky: když má vodorovná ulice volno, musí na obou vodorovných semaforech být zelená. Tedy na nich rozsvítíme zelenou a všechny ostatní barvy zhasneme. Na svislých semaforech zase musí být červená atd.

To je úplně špatně.

Tento postup totiž k zadání přistupuje jako k obrázku, ale vůbec nerespektuje funkcionalitu. To znamená, že jednotlivé prvky obrázku nepopisuje jako věci z reálného světa, ale jen jako jejich obrazy. Na nejvyšší úrovni nerespektuje, že pro dopravní světla existují jistá omezení, že například nelze oba směry pustit současně. Na nižší úrovni také nerespektuje realitu, například na semaforu nikdy nesmí svítit současně červená i zelená.

Takový způsob psaní programů je typický pro programy v BASICu.

Ano, je věcí programu, aby se o to postaral a aby model fungoval korektně. Jenže to je poměrně pracné, nepřehledné a je velké nebezpečí, že v tom uděláme chybu. A navíc, takový program by

se jen velmi obtížně modifikoval. Představme si, že bychom měli křižovatku rozšířit podle [obrázku](#), to znamená doplnit další semaforey tak, aby dolní cesta měla samostatný semafor pro odbočení doleva. To by znamenalo úplně přepracovat celý algoritmus řešení, tedy bylo by to pracné a nebezpečné kvůli riziku chyby.

Správný postup je, rozdělit logiku aplikace na jednotlivé **úrovně abstrakce** a na každé úrovni spojit vzhled (obrázek) s funkcionalitou.

Například, na úrovni semaforů bychom viděli, že semafor má tři světla a že může mít čtyři stavy: červená, žlutá, zelená a červená se žlutou. Případně ještě pátý stav (nesvítí nic). Semafor by pro

nás byl objekt, který má nějakou **grafickou podobu** (šedivý ovál s barevnými kroužky) a současně se schopností vykonávat nějakou **činnost**. V našem případě by tou činností bylo „přejít do stavu 1“, „přejít do stavu 2“ atd., až „přejít do stavu 5“.

Navyšší úrovni bychom si všimli, že na křižovatce jsou čtyři semaforey a že se střídají podle nějakých pravidel. Proto bychom vytvořili třídu TrafficLights a ta by řídila jednotlivé semaforey. To znamená, volala by jejich funkce „přejít do stavu 1“, „přejít do stavu 2“ atd. Důležité je, že tato třída nepotřebuje nic vědět o vnitřním uspořádání semaforu (a také to neví). Kdybychom, dejme tomu, vyměnili semafor za jiný, který by místo kulatých

světél měl textové nápisy, třída TrafficLights by o tom nepotřebovala nic vědět. Model by fungoval dál.

Java přímo podporuje správný styl programování, protože nutí programátory, aby kód rozdělovali do tříd. Samozřejmě, i v Javě se dá napsat špatně strukturovaný kód, ale je to spíše neobvyklé. Platí známá pravda, že BASICovský program se dá napsat v jakémkoliv jazyce 😊.

Důležité

V praxi bychom třídu Semaphore realizovali jako samostatnou třídu, která by obsahovala i grafiku. Pracovalo by se s ní jako s ostatními komponentami, tzn. z knihovny by se přetahovala na kreslicí plochu stejně, jako to děláme třeba s komponentou

Button. Jenže k tomu, abychom něco takového naprogramovali, naše znalosti nestačí. Proto použileme náhradní řešení, že všechny třídy umístíme přímo v souboru *FXMLDocumentController.java*.

Třída Semafor

Nyní si ukážeme, jak se dá vytvořit jednoduchá třída „Semaphore“. Vybavíme ji 5 funkcemi (schopnostmi) pro přepnutí do 5 přípustných stavů.

Grafika

Začneme tím, že vytvoříme [nový projekt](#) a nazveme ho [TrafficLights](#). Samozřejmě se otevře náš známý [HelloWorld](#). [Otevřeme SceneBuilder](#) a [Button](#) i [Label](#) v projektu ponecháme, jen je odsuneme jinam, aby nepřekážely. (Kdybychom je chtěli odstranit, [vybrali](#) bychom je a klávesou Del smazali.)

Na levé straně klikněte na Shapes. Najděte [Rectangle](#) (obdélník) a umístěte ho doprostřed pracovní plochy. Najít přesný střed je snadné, pomohou nám k tomu červené pomocné středové čáry. Tento obdélník bude představovat grafickou podobu semaforu. Proto ho trochu zúžíme.

Na pravé straně obrazovky jsou Properties, vlastnosti. Například položka Fill označuje barvu, jakou má být tělo semaforu vybarveno. Změňte ji na světle [šedivou](#).

Mimochodem, upravovat můžeme všechny vlastnosti. Například [Rotate](#) znamená otočit a upraví nám natočení grafického prvku.

Podobně jako předtím, na levé straně najděte Circle a přetáhněte ho na pracovní plochu. Na poloze nezáleží, [kruh](#) se tam stejně nevejde.

Proto na pravé polovině obrazovky, kde jsou jeho vlastnosti, otevřeme záložku Layout (vzhled nebo tvar) a nastavíme Radius=25. Kruh umístěte a [změňte jeho barvu](#) na černou (bezpečný stav). Tím máme základ pro třídu Semaphore.

SceneBuilder ↔ FXML ↔ NetBeans

Data mezi NetBeans a SceneBuilder se předávají prostřednictvím FXML souboru. Proto nezapomeňte obrázek, který jsme právě vytvořili, uložit!

Nyní se přepněte do NetBeans.

V NetBeans se změny neobjeví automaticky, musíme je ze souboru FXML nejdřív načíst!

Uděláme to tak, že pravým tlačítkem klepneme na soubor [FXMLDokument.xml](#) a vybereme **Edit**. Otevře se nám text souboru, ve kterém ([červená šipka](#)) vidíme, že třída Semaphore se skládá z objektu rectangle, ze tří objektů Circle a jednoho objektu Button. Každý z objektů má své parametry. Když se nyní pokusíme program spustit, ukáže se obrázek semaforu.

Jenže to není všechno. Protože NetBeans a SceneBuilder jsou samostatné programy, tak jeden neví nic o proměnných, které jsme definovali ve druhém. A naopak. Musíme tedy vytvořit

obousměrné propojení mezi NetBeans a SceneBuilder pro předávání proměnných. Toto propojení se děje přes soubor FXML.

Nejprve se podívejme na propojení NetBeans ↔ FXML. V souboru *FXMLDocumentController.java* vidíme, že tam jsou dva řádky:


```
@FXML  
private Label label;
```

Druhý řádek říká, že Java bude pracovat s proměnnou, která se jmenuje „label“ a je typu Label. A že to bude privátní proměnná.

První řádek obsahuje tag `@FXML`. Tímto tagem se označuje proměnná, která se má propokit do souboru FXML. **Tag `@FXML` musí být před každou proměnnou, která se má do souboru FXML propojit.**

Něco podobného musíme udělat také. Nadefinujeme tři proměnné, které vtipně nazveme `redLight`, `greenLight` a `yellowLight`. Všechny budou typu `Circle` (kruh) a před každou bude tag `@FXML`.

Když to zkusíme udělat, objeví se [chyba](#), protože nejdříve musíme importovat knihovnu pro `Circle`. Snadno to uděláme tak, že přijmeme návrh, který nám dává NetBeans. Program tedy bude vypadat [takto](#). To byl první krok.

Ale když se podíváme do FXML souboru, nic takového tam nenajdeme! Aby se změna přenesla do souboru FXML, musíme program nejdřív přeložit. Vhodná je ikona 

Nyní **znovu (!) otevřeme** soubor FXML ve ScreenBuilderu, vybereme některé světlo a na záložce **Code:** ve volbě **fx:id** přiřadíme [název](#), který k tomu světlu patří. To postupně zopakujeme pro všechna světla. Nakonec nezapomeneme výsledek uložit.

Když si nyní znovu (volbou *Edit*) otevřete soubor FXML, vidíte, že tam přibyla [informace](#) k propojení programu s grafilou, např.

```
fx:id="redLight".
```

Pro účely testování můžete zkusit do obsluhy Buttonu přidat řádku jako:

```
redLight.setFill(Color.BLUE) ;
```

ale nezapomeňte současně importovat knihovnu

```
import javafx.scene.paint.Color;
```

Programová logika

Nyní do třídy Semaphore začneme přidávat funkcionalitu.

Naprogramovat funkci semaforu je úloha do cvičení. Příklad, jak může vypadat hotový program, je [zde](#).

Příklad: kalkulačka

Scene Builder slouží k názorné editaci oken obsahujících grafiku. Dobře spolupracuje s NetBeans, ale není to součást NetBeans, je to samostatná aplikace.

Vazba mezi NetBeans a SceneBuilder je zprostředkována souborem FXML. Když na **FXMLDocument.fxml** klikneme pravým tlačítkem a vybereme **Open**, spustí se [SceneBuilder](#) a v grafické podobě ukáže obsah tohoto souboru. V zásadě: vlevo je navigace, uprostřed obrázky a napravo vlastnosti.

Každý projekt JavaFX na začátku obsahuje program Hello World. Vyjděme z něj jako z dobrého startovacího bodu.

Vyzkoušejme si několik úprav. Nejdřív klikněte na položce Button v panelu Hierarchie (vlevo dole) a tím vyberte tlačítko Button, které je uprostřed panelu (viz [obrázek](#)). Tlačítko můžeme vybrat také tak, že přímo na něj klikneme. Myší ho přemístíme trochu dolů a doleva, abychom udělali místo pro další tlačítko.

Nyní na panelu Library (je vlevo nahoře) najděte Button a přetáhněte ho na plochu. Bude to naše druhé tlačítko. pomocí červených pozičních čar jej můžeme vhodně umístit (viz [obrázek](#)).

