ŽILINSKÁ UNIVERZITA V ŽILINE

Fakulta riadenia a informatiky

BAKALÁRSKA PRÁCA

Marek Zaťko

Grafická modelácia štruktúry vybraných uhľovodíkov v programovacom jazyku JAVA

Vedúci práce: RNDr. Denisa Maceková, PhD.

Registračné číslo: 181/2017

Žilina, 2018

Poďakovanie

Rád by som poďakoval všetkým, ktorí mi pomáhali pri tvorbe tejto práce. Predovšetkým vďaka patrí RNDr. Denise Macekovej, PhD a RNDr. Petrovi Varšovi PhD. za cenné rady a pripomienky.

Ďakujem

ABSTRAKT V ŠTÁTNOM JAZYKU

ZAŤKO, Marek: *Grafická modelácia štruktúry vybraných uhľovodíkov v programovacom jazyku JAVA*. [Bakalárska/ Diplomová]. – Žilinská univerzita v Žiline. Fakulta Riadenia a Informatiky; Katedra Informatiky. –Vedúci: RNDr. Denisa Maceková PhD. – Stupeň odbornej kvalifikácie: bakalár– Žilina: FRI ŽU,2018. Počet strán: 54

Práca je zameraná na vytvorenie výučbovej pomôcky organickej chémie konkrétne na problematiku uhľovodíkov. Program je naprogramovaný v jazyku JAVA hlavne s využitím knižníc JavaFX. Čitateľ sa v práci dozvie základy o uhľovodíkoch, niečo o knižniciach JavaFX a o princípoch tvorenia vzorcov a modelov uhľovodíkov.

Kľúčové slová: uhľovodíky, grafika, JAVA, FXML, JavaFX, 3D modelovanie, chémia, vzorce, chemické modely

ABSTRAKT V CUDZOM JAZYKU

ZAŤKO, Marek: *Graphical modeling of the structure of selected hydrocarbon compounds in the JAVA programming language* . [Bachelor thesis]. – University of Žilina.Faculty of Management Science and Informatics; Department of informatics. –Supervisor: RNDr. Denisa Maceková PhD. – Qualification: bachelor– Žilina: FRI ŽU,2018. Number of pages: 54

The point of the thesis was to create a learning tool for organic chemistry, mainly for branch called hydrocarbons. The application is written in JAVA programming language with use of JavaFX libraries. In this thesis we will focus on explaining the basics of hydrocarbons, JavaFX libraries and we will also focus on the principle of creating formulas and hydrocarbon models..

Key words: hydrocarbons, JAVA, FXML, JavaFX, 3D modeling, chemistry, formulas, models

Obsah

Úvod

Na úvod by som rád oboznámil čitateľa o zámere mojej záverečnej práce a jej možné využitie v obore.

Zámerom práce bolo vytvorenie počítačovej aplikácie o prehľade **základných *informácií a štruktúry*** vybraných skupín uhľovodíkov a vybraných zástupcov týchto skupín.

Pri hľadaní podstatných informácií o konkrétnych uhľovodíkoch sme narazili na problém, že informácie nie sú zhromaždené na jednom internetovom portály alebo knihe, informácie ako zápach, toxicita, použitie atď.. sú častokrát v inom zdroji ako vzorce prípadne modely uhľovodíka.

Takže aplikácia je akýmsi zjednotením rôznych zdrojov týchto informácií čím používateľovi uľahčí pátranie po nich a tak ušetrí čas **z čoho vyplýva aj možné využitie v odbore:**

- Výučba základov organickej chémie na rôznych školách, napríklad aj na výučbu v novo pripravovanom študijnom programe **biomedicínska informatika** na našej fakulte.

- Keďže aplikácia obsahuje vzorce aj 3D modely vybraných uhľovodíkov, vyučujúcemu môže aplikácia napríklad pomôcť vysvetliť princípy skladania týchto vzorcov a modelov.

V súčasnosti existuje mnoho zdrojov informácií o uhľovodíkoch, existujú rôzne publikácie, internetové portály, časopisy, ale pri hľadaní rôznych zdrojov sme nezaznamenali jednoduchú desktopovú aplikáciu, ktorá by nezachádzala až príliš do detailov s jednoduchým používateľským rozhraním, ktorého ovládanie zvládne aj laik čo sa ovládania počítača týka.

Takže výsledná aplikácia je ***vhodná pre používateľa začiatočníka***, ktorý sa chce dozvedieť podstatné **informácie, vzorce a modely** niektorých dôležitých uhľovodíkov.

1 Stručný úvod do organickej chémie

Organická chémia je odvetvie chémie, ktoré si prešlo historicky mnohými zmenami a neustále sa vyvíja. V súčasnosti sa organická chémia zaoberá organickými zlúčeninami.

Organické zlúčeniny sú zlúčeniny, ktoré vo svojej štruktúre obsahujú *väzby uhlíku a vodíku*. Práve organické zlúčeniny sú základným stavebným kameňom všetkých známych živých organizmov. Táto skutočnosť je hlavným dôvodom vzniku samostatného odvetvia chémie, organická chémia.

1.1 Uhlík

Uhlík sa považuje za najdôležitejší prvok biosféry. Základný stavebný kameň všetkého živého tvorí vďaka jeho špecifickým chemickým a fyzickým vlastnostiam, ktoré sú z pomedzi všetkých známych prvkov najideálnejšie čo sa štruktúry života týka.

Najdôležitejšou vlastnosťou pre našu prácu je fakt, že z dôvodu stability si uhlík vytvára **štyri väzby**,

1.2 Vodík

Vodík je v základnom stave najľahším a najmenším prvkom periodickej tabuľky prvkov. Je to najrozšírenejší prvok vo vesmíre a tretí najrozšírenejší na Zemi.

Z jeho mnohých vlastností si spomenieme len pre nás najdôležitejšiu, ktorou je vlastnosť, že vodík opäť z dôvodov stability je v prírode prítomný vždy vo forme H_2 alebo vo väzbe s iným atómom.

1.3 Väzby uhlíku a vodíku

O fyzikálnych dôvodoch ako aj o princípe tvorenia väzieb medzi atómami nemá zmysel hovoriť v našej práci, preto si povieme len najdôležitejšie fakty o väzbách medzi atómami uhlíku a atómami vodíku v organických zlúčeninách

- Uhlík môže s ďalším atómom uhlíku tvoriť jednoduchú väzbu od jednej až po štyri, keďže v organických zlúčeninách je štvorväzbový, dvojitú väzbu, t.j. maximálne 2 dvojité väzby alebo trojitú a jednoduchú väzbu.

- Vodík tvorí len jednoduchú väzbu s ďalším prvkom, v našej práci bude týmto ďalším prvkom atóm uhlíku.

- Súčet väzieb atómu uhlíku je v jednoduchých uhľovodíkoch štyri, , v našej práci nebudeme uvažovať o väzbách s iným súčtom ako štyri.

1.4 Uhľovodíky

Ako z názvu vyplýva ide o zlúčeniny, skladajúce sa výhradne z atómov uhlíku a vodíku. Uhľovodíky sú hlavným zdrojom energie modernej civilizácie. Rozdeliť ich môžeme podľa rôznych kritérií, z ktorých sme pre účely našej práce zvolili nasledujúce rozdelenie:

Rozdelenie uhľovodíkov:

Acyklické uhľovodíky– uhľovodíky, ktoré vo svojom reťazci neobsahujú cyklus, patria sem skupiny:

- **Alkány** – obsahujú len jednoduché väzby
- **Alkény** – obsahujú jednu dvojitú väzbu, inak sú všetky ostatné väzby jednoduché
- **Alkíny** – obsahujú jednu trojitú väzbu, inak sú všetky ostatné jednoduché

Cyklické uhľovodíky – vo svojom reťazci obsahujú cyklus:

- **Cykloalkány** – obsahujú len jednoduché väzby
- **Cykloalkény** – obsahujú jednu dvojitú väzbu, inak sú všetky ostatné väzby jednoduché
- **Cykloalkíny** – obsahujú jednu trojitú väzbu, inak sú všetky ostatné jednoduché

Aromatické uhľovodíky – Jednoducho povedané sú to cyklické uhľovodíky, v ktorých je systém dvojitých väzieb alternujúcu t.j. dvojitá väzba sa vyskytuje medzi každým druhým párom susediacich atómov uhlíku.

1.4.1 Vybrané uhľovodíky

V definícií uhľovodíkov ako takých nie je žiadne obmedzenie čo sa dĺžky reťazca týka, z čoho vyplýva, že zástupcov daných skupín je veľa, preto je práca zameraná z dôvodu prehľadnosti a vytvorenia len ukážkovej formy na určitých zástupcov spomínaných skupín.

Tabuľka 1 Vybrané uhľovodíky

Alkány	**Alkény**	**Alkíny**	**Cykloalkány**	**Cykloalkény**	**Cykloalkíny**	**Aromatické**
metán	etén	etín	cyklopropán	cyklohexén	cyklohexín	benzén
etán	hexén	hexín	cyklohexán			
propán						
bután						
hexán						

Jednotlivých zástupcov sme vyberali tak, aby sme používateľovi ukázali rôzne štruktúry, ktoré uhľovodíky môžu nadobúdať.

Implementácia programu

Táto kapitola je najdôležitejšou časťou celej práce, predstavíme si použité softvérové platformy, povieme si o najdôležitejších častiach programu a spôsob akým boli implementované, predstavíme si kľúčové myšlienky samotného modelovania uhľovodíkov a ukážeme si najpodstatnejšie časti kódu.

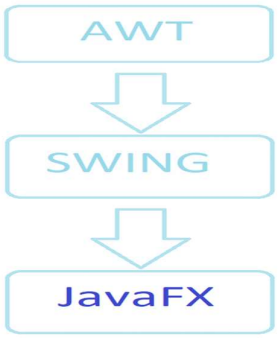
2 Použitá softvérová platforma

Ako už zo zadania vyplýva, aplikácia je napísaná v programovacom jazyku JAVA, ktorú si určite nemusíme bližšie predstavovať, bližšie si však predstavíme použitú Java knižnicu na tvorbu samotnej grafiky výslednej aplikácie a povieme si o niektorých jej kľúčových vlastnostiach a mechanizmov a dôvody jej zvolenia.

2.1 JavaFX

JavaFX je softvérová platforma vybudovaná na programovacom jazyku JAVA. Používa sa na hlavne na vytváranie desktopových ale aj na tvorbu RIA(Rich Internet Application) aplikácií. Pred verziou JavaFX 2.0 sa k vývoju používal statický skriptovací jazyk JavaFX Script, ktorý nebol príliš vhodný a tak je JavaFX od ďalších verzií implementovaná ako **natívna JAVA knižnica** a JavaFX Script už nebol ďalej vyvíjaný.

JavaFX je v súčasnosti najpoužívanejšia knižnica na tvorbu užívateľských rozhraní (GUI) v jazyku JAVA,. JavaFX takisto podporuje aj **2D a 3D grafiku**, ktorú v práci využívame, čo bolo dôvodom jej zvolenia ako hlavnej grafickej knižnice v našej aplikácií. V roku 2014 JavaFX oficiálne nahradila predtým najpoužívanejšiu knižnicu Swing.



Obrázok 1 História najpoužívanejších grafických knižníc v JAVE.

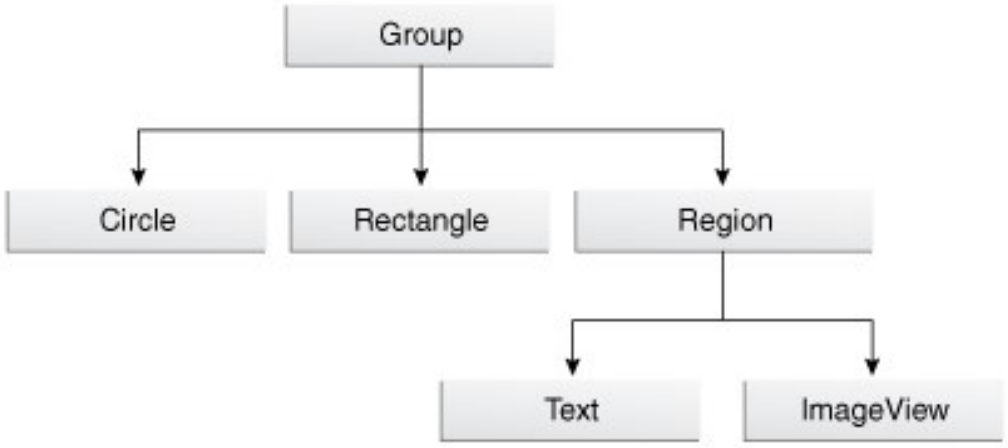
V našej aplikácií sme použili JavaFX na úplne všetko čo sa grafiky týka, vytvorili sme pomocou nej GUI, vykreslili vzorce a namodelovali spomínané uhľovodíky, predtým ako sa pustíme do samotnej implementácie, ukážeme si niektoré kľúčové prvky a spôsob fungovania knižnice JavaFX.

2.1.1 SceneGraph

SceneGraph je stromová štruktúra, ktorá definuje všetky grafické prvky JavaFX aplikácie (3D objekty, tlačidlá, panely, atď...). Každý tento grafický prvok má spoločného predka *Node*. Potomkov triedy *Node* môžeme rozdeliť na dve časti:

1. ***Rodičovské triedy*** – predstavujú kontajner pre ďalších potomkov triedy Node, ktoré sú z hľadiska tohto kontajnera jeho deti, tieto deti si kontajner drží v *ObservableList<Node>*, ktorý získame metódou *getChildren(),* ďalšou vlastnosťou je to, že akákoľvek geometrická transformácia volaná nad určitým druhom kontajnera(napríklad *Group*) aplikuje túto transformáciu aj na jej deti, táto vlastnosť je v našej aplikácií veľmi dôležitá. Každý kontajner si drží svoje deti usporiadané podľa definovanej hierarchie napríklad *HBox* (Horizontálny Box), si deti drží horizontálne usporiadané.

2. ***Listové triedy*** – potomkovia triedy *Node*, ktorí nemôžu mať ďalšie deti, napríklad *Circle, ImageView, Text*, atď...



Obrázok 2 Príklad štruktúry SceneGraph

SceneGraph je v skutočnosti taktiež API, ktoré si v sebe udržiava vnútorný model všetkých komponentov čo znamená, že API určuje kedy je rozumné vyvolať *render*. Táto skutočnosť v konečnom dôsledku ušetrí programátorovi prácu, pretože by musel tento mechanizmus napísať sám čo sa nemusí vždy podariť spraviť ideálne.

2.1.2 FXML

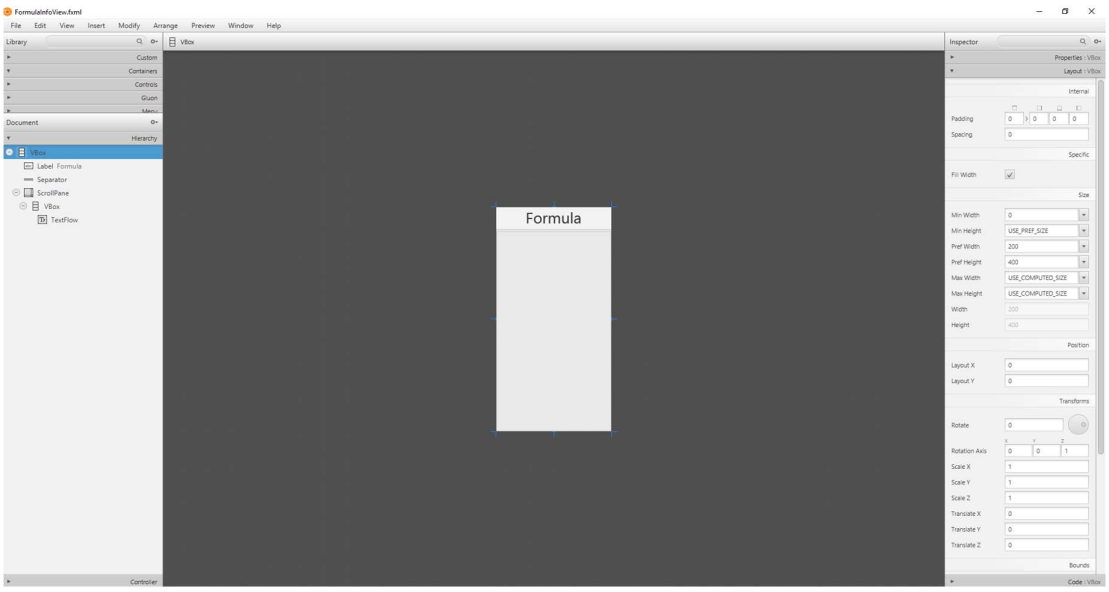
Ďalším dôležitým prvkom knižnice JavaFX je skriptovací jazyk FXML, ktorý nám umožňuje definovať GUI. Avšak celý GUI, ktorý napíšeme pomocou FXML, vieme napísať aj pomocou funkcií JavaFX.

FXML aj napriek tejto vlastnosti prináša výhodu, ktorou je logické odčlenenie jednotlivých častí GUI a to za pomocou mechanizmu Controller, každý FXML dokument môže mať totiž definované meno triedy, ktorá je zodpovedná za správu časti GUI, ktorú definuje spomínaný FXML dokument a tak značne pomáha spraviť kód viac prehľadnejším. Controller môžeme FXML dokumentu definovať aj v programe pri jeho inicializovaní a tak mu doplniť rôzne vlastnosti.

Príklad je súbor **FormulaInfoView.fxml** , ktorý zobrazuje informácie o momentálne zobrazovanom vzorci v našej aplikácii:

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.
3.  <?import javafx.scene.control.Label?>
4.  <?import javafx.scene.control.ScrollPane?>
5.  <?import javafx.scene.control.Separator?>
6.  <?import javafx.scene.layout.VBox?>
7.  <?import javafx.scene.text.Font?>
8.  <?import javafx.scene.text.TextFlow?>
9.
10.
11. <VBox fx:id="mainBox" alignment="TOP_CENTER" minHeight="-
    Infinity" minWidth="0.0" prefHeight="400.0" prefWidth="200.0" xmlns="http://javaf
    x.com/javafx/9.0.1" xmlns:fx="http://javafx.com/fxml/1">
12.    <children>
13.       <Label fx:id="labelFormula" text="Formula">
14.          <font>
15.             <Font size="27.0" />
16.          </font>
17.       </Label>
18.       <Separator prefWidth="200.0" />
19.       <ScrollPane fx:id="scrollPane" fitToHeight="true" fitToWidth="true" VBox.vg
    row="ALWAYS">
20.          <content>
21.             <VBox alignment="TOP_CENTER">
22.                <children>
23.                   <TextFlow fx:id="tfFormulaInfo" />
24.                </children>
25.             </VBox>
26.          </content>
27.       </ScrollPane>
28.    </children>
29. </VBox>
```

Všetky FXML súbory v našej aplikácií sú vytvorené cez nástroj SceneBuilder, ktorý nám na základe navrhnutého GUI vygeneruje FXML súbor.



Obrázok 3 Nástroj SceneBuilder s otvoreným súborom FormulaInfoView.fxml

Teraz keď máme základnú predstavu o fungovaní knižnice JavaFX , môžeme si začať predstavovať samotnú aplikáciu.

3 Základné časti aplikácie

Predtým, ako si ukážeme použité implementačné techniky, si musíme povedať niečo o základných častiach z ktorých sa naša výsledná aplikácia skladá.

Ako sme už spomínali, aplikácia má byť schopná zobrazovať informácie o uhľovodíkoch a to konkrétne nasledujúce tri druhy:

1. **Zobrazovanie základných informácií** – pod pojmom základné informácie si v tejto práce budeme predstavovať informácie ako zápach, skupenstvo, využitie a pod... . Skrátka všetky podstatné informácie, ktorým porozumie aj začiatočník. Všetky základné informácie sme čerpali z rôznych publikácií, spoľahlivých internetových portálov a podobne.

2. **Zobrazovanie vzorcov uhľovodíkov** – aplikácia umožňuje prezeranie nasledujúcich vzorcov o ktorých si bližšie povieme, keď budeme riešiť ich samotnú implementáciu: Sumárny vzorec, Empirický vzorec, Racionálny vzorec, Štruktúrny vzorec.

3. **Zobrazovanie modelov uhľovodíkov** – aplikácia umožňuje 3D zobrazovanie vybraných modelov o ktorých si taktiež povieme pri implementácií tejto funkčnej časti : Kalótový model , Guľôčkový model, Trubičkový model.

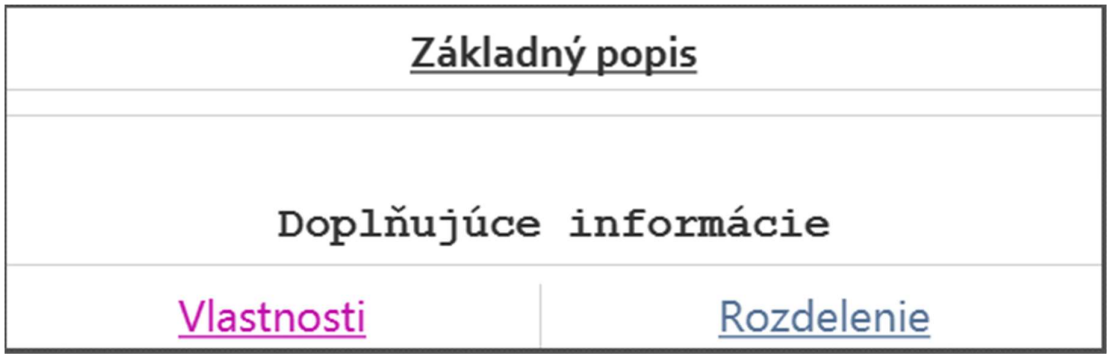
V každej nasledujúcej kapitole si rozoberieme detaily implementácie jednotlivých spomínaných častí.

4 Zobrazovanie základných informácií

Spomínané základné informácie prezentujeme okrem konkrétnych uhľovodíkov aj pre jednotlivé skupiny(alkány, alkény, atď...) a aj celkovo pre uhľovodíky ako také. Čo vlastne predstavuje jednoduchú **stromovú štruktúru** kde koreňom sú informácie všeobecne o uhľovodíkoch, prvou úrovňou sú spomínané informácie o skupinách uhľovodíkov (alkány, alkény, ...), a treťou úrovňou sú informácie o samotných zástupcoch týchto skupín.

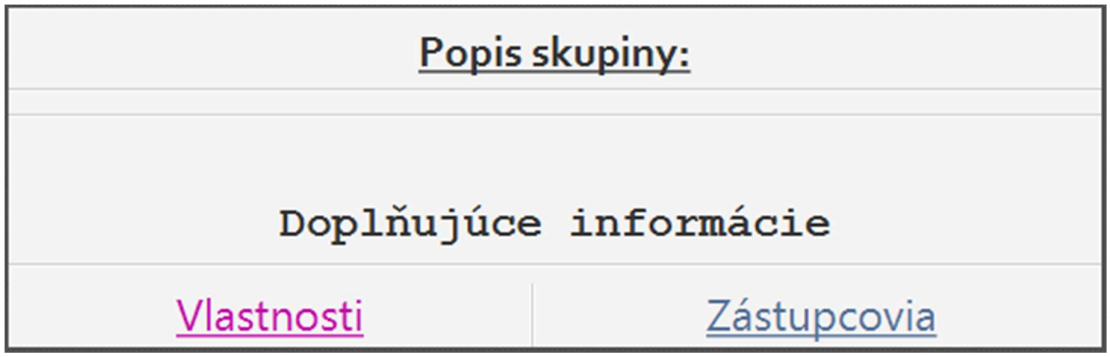
Z tejto skutočnosti samotnej vyplýva potreba existencie troch rôznych tried a troch rôznych častí GUI, pretože napríklad podstatné informácie pre skupinu uhľovodíkov(napr. alkány) sú iné ako pre nejakého zástupcu skupiny, pre skupinu je dôležitou informáciou napr. zoznam zástupcov tejto skupiny ale tento zoznam nemá zmysel mať už v samotnom zástupcovi, práve preto sme vytvorili tri rôzne triedy a tri rôzne FXML dokumenty:

- ***HydroCarbonsBase*** – je trieda, ktorá obsahuje informácie o uhľovodíkoch všeobecne, to znamená, že tvorí koreň tohto spomínaného hypotetického stromu, časť GUI (HCsView.fxml) prezentujúca tieto informácie vyzerá takto:



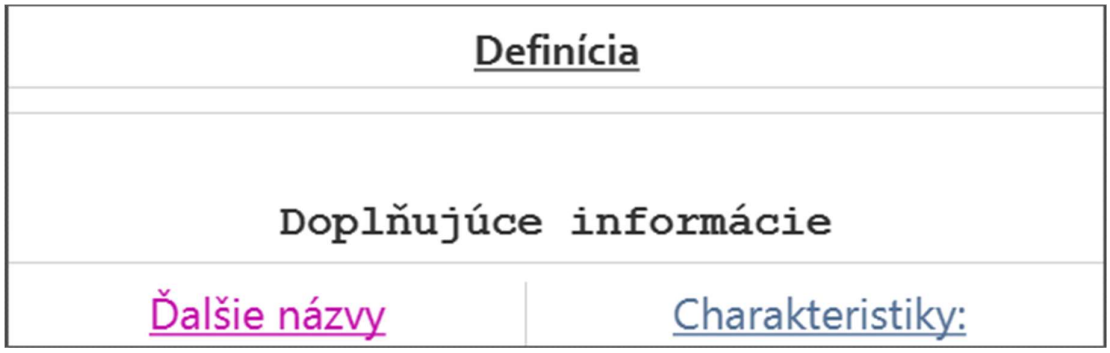
Obrázok 4 Vizualizácia HCsView.fxml

- ***HydroCarbonGroup*** – trieda, ktorá drží informácie o konkrétnej skupine uhľovodíkov (napr. alkány, alkény, ...), na zobrazovanie týchto informácií využívame súbor GroupView.fxml :



Obrázok 5 Vizualizácia GroupView.fxml

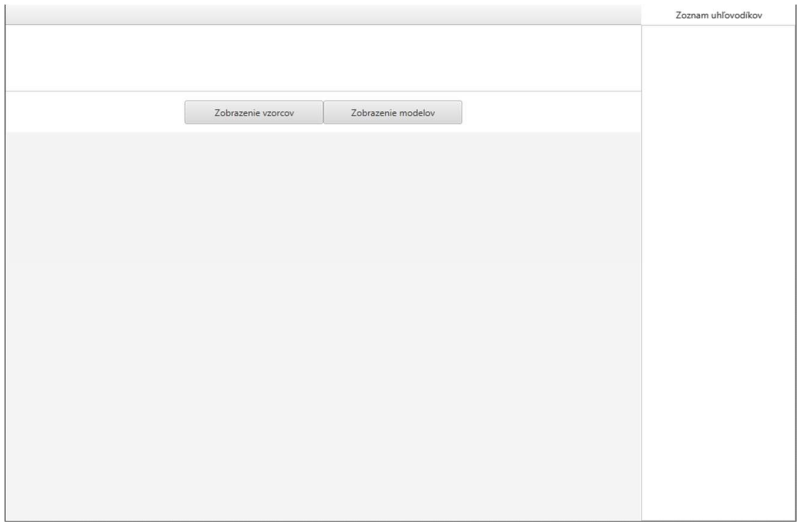
- ***HydroCarbon*** – trieda reprezentujúca konkrétny uhľovodík, počíta sa v nej celková štruktúra uhľovodíku, takisto drží základné informácie o uhľovodíku. Pre zobrazenie základných informácií máme súbor HCView.fxml:



Obrázok 6 Vizualizácia HCView.fml

Túto stromovú štruktúru sme v JavaFX zachytili pomocou kontajneru ***TreeView<T>***, ktorý uživateľovi prezentuje štruktúru ako vertikálny panel. Takto definovaný TreeView môže obsahovať položky len jednej triedy. Preto sme vytvorili interface ***TreeItemViewable***, ktorý má len jednu metódu, ktorou je *toString(),* ktorú volá samotný TreeView, získa z nej text, čo má zobraziť v samotnom TreeView panely, to znamená, že všetky tri vyššie spomenuté triedy implementujú tento interface a tak môžu vystupovať v samotnom strome.

Je logické, že strom nemôžeme naplniť priamo v FXML súbore ale musíme ho naplniť priamo v programe a práve to má za úlohu metóda *setupTreeView()* v triede mainController, ktorá je už spomínaným Controllerom pre základný FXML súbor celého zobrazovania základných informácií , tento FXML súbor vyzerá nasledovne:



Obrázok 7 Vizualizácia mainGUI.fxml

Takéto rozčlenenie nám dáva možnosť jednoducho dynamicky meniť momentálne zobrazované GUI, po kliknutí na nejakú položku stromu jednoducho odoberieme momentálne zobrazovanú časť zo spomínaného SceneGraphu čím sa táto časť prestane zobrazovať a pridáme do neho novú potrebnú časť, ktorú predtým naplníme potrebnými informáciami. Toto naplnenie informáciami realizujeme opäť za pomoci Controlleru, ktorého inštanciu si ale musíme odložiť, potom jednoducho zavoláme nejakú metódu nad týmto Controllerom, ktorá naplní danú časť GUI informáciami, ktoré dostane ako parameter, ukážeme si to na príklade GroupView:

1. **Inicializovanie potrebných GroupView častí:**

```
1. private void setupSubFXMLParts() throws IOException {
2.     groupController = new GroupController();
3.     fxGroup = new FXMLLoader(getClass().getResource("GroupView.fxml"));
4.     fxGroup.setController(groupController); //groupController je controllerom pre
   GroupView.fxml
5.     vboxGroup = fxGroup.load(); // vboxGroup obsahuje časť GUI pre GroupView
6. // ďalej rovnako inicializujeme aj ostatné dve časti
7. }
```

2. **Po kliknutí na skupinu uhľovodíkov**

```
1. private void handleInfoTAB(TreeItemViewable tr) {
2.         if (tr instanceof HydroCarbonsBase) {
3.             scrollPane.setContent(vboxHcs);
4.         } else {  // ak bola kliknutá nejaká skupina (alkány,alkény,...)
5.             groupController.setGroup((HydroCarbonGroup) tr);  //nastavíme zobraze
   né informácie
6.             scrollPane.setContent(vboxGroup);// pridáme do SceneGraphu
7.
8.         }
9.     }
```

Ďalej pokladáme za dôležité aby čitateľ pochopil princíp tzv. *properties*, pretože tento koncept používame často, vysvetlíme si to na príklade akým reagujeme na kliknuté položky spomínaného *TreeView<TreeItemViewable>*.

TreeView si drží vnútorný model všetkých položiek, k tomuto modelu môžeme pristúpiť pomocou metódy *getSelectionModel()* , tento model obsahuje okrem iného aj *selectedItemProperty*, čo predstavuje špeciálnu premennú, ktorá sa zmení vždy, keď sa zmení vybraná položka stromu, znie to ako obyčajná bežná premenná, ale má jednu šikovnú vlastnosť, môžeme jej priradiť **funkciu(*Listener*)**, ktorá sa vykoná vždy, keď sa zmení táto premenná, a taktiež ju môžeme **nadviazať (*nabindovať*)** na inú property. Takéto nabindovanie spôsobí, že ak sa zmení hodnota premennej ktorú obaľuje nabindovaná property, tak sa zmení aj hodnota premennej v druhej property, a tak šetrí programátorovi prácu, lebo nemusí riešiť tento mechanizmus on sám.

Najskôr si ukážeme príklad pre Listener v selectedItemProperty stromu:

```
1.  treeMain.getSelectionModel().selectedItemProperty().addListener(((observable, old
    Value, newValue) -> {
2.
3.      handle(newValue);  // funkcia sa zavolá ak sa zmení premenná v
    selectedItemProperty
4.
5.  }));
```

Property je z funkčného hľadiska obaľovacou triedou, obaliť môže akýkoľvek objekt ale aj primitívny dátový typ.

Ukážme si príklad využitia bindingu, ktorý využívame na **zväčšenie veľkosti písma**, po stlačení klávesovej kombinácií **CTRL + MOUSEWHEEL** v našej aplikácií:

1. **Inicializovanie fontProperty**

```
1.  private SimpleObjectProperty<Font> fontProperty = new SimpleObjectProperty<Font>(
    FontSizeBinder.BASE_FONT)// inicializovanie fontProperty
```

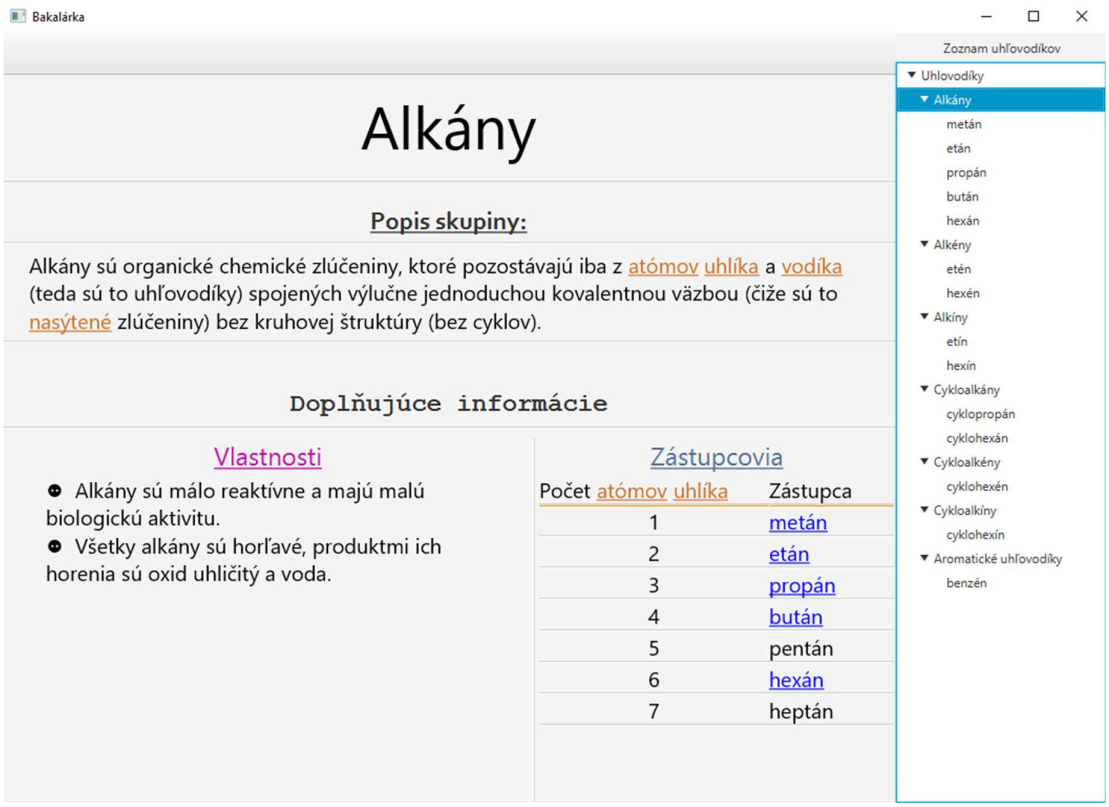
2. **Definovanie listeneru na klávesovú kombináciu**

```
2.  mainBox.setOnScroll(e->{
3.       if(e.isControlDown()){
4.            if(e.getDeltaY() > 0)  // ak užívateľ scrollol hore
5.            currentFontSize +=FontSizeBinder.FONT_SIZE_INCREMENT;
6.            else
7.             currentFontSize -=FontSizeBinder.FONT_SIZE_INCREMENT;
8.
9.            fontProperty.set(Font.font(FontSizeBinder.FONT_FAMILY_USED,currentFontS
    ize));
10.       }
11.    });
```

3. Nabindovanie fontProperty na fontProperty iných Node

```
12. Text zastupca = new Text("Zástupca");
13.       FontSizeBinder.bindFontSizeAll(fontProperty, zastupca);
14.
15.  //Metóda bindFontSizeAll v triede FontSizeBinder
16.    public static void bindFontSizeAll (ObjectExpression<Font> fontTracking,Text..
    . args){
17.        for (Text arg : args) {
18.
19.            arg.fontProperty().bind(fontTracking);
20.        }
21.    }
```

Rovnaký mechanizmus využívame aj v iných častiach programu na bindovanie fontov.



Obrázok 8 Informácie o alkánoch

Výsledný naplnený *TreeView* môžeme vidieť na obrázku 8 v pravej časti, a GroupView.fxml naplnený konkrétnymi informáciami v ľavej časti. Slová oranžovou farbou predstavujú doplnkové informácie tzv. **ToolTips**, modré sú **odkazy na strom**, o spôsobe fungovania týchto dvoch mechanizmov si povieme až keď si povieme niečo o dátovom modeli aplikácie.

4.1 Dátový model aplikácie

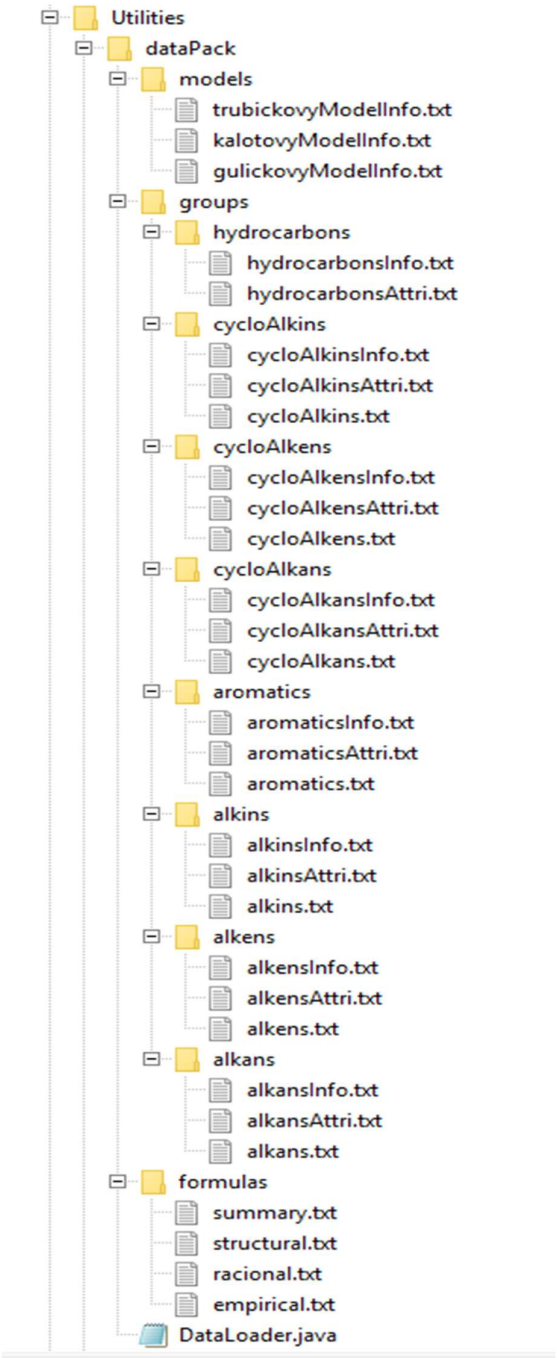
V zadaní našej práce je spomínaný bod vytvorenia **databázy** vybraných uhľovodíkov. Pod týmto pojmom si skoro každý z nás predstaví relačnú databázu v ktorej manažujeme dáta cez nejaký programovací jazyk napríklad SQL, takáto databáza je veľmi efektívna na manažovanie veľkého množstva dát čo však v našej aplikácií nie je potreba. V našej aplikácií nemáme veľké množstvo dát, tak sme sa rozhodli, že **na úschovu informácií v našej aplikácií použijeme obyčajné textové súbory**, čo má niekoľko výhod:

- Nepotrebujeme knižnice na pripojenie k takejto databáze, čo ovplyvní veľkosť aplikácie

- Jednoduchá zmena informácií, napríklad keby používateľ z nejakých dôvodov potreboval zmeniť text v aplikácií, stačí prepísať informáciu v konkrétnom textovom súboru, čo zvládne aj úplný laik a nepotrebuje tak vedieť syntax jazyka, napríklad spomínaného SQL.

Náš systém textových súborov je veľmi jednoduchý. Všetky textové súbory, z ktorých čítame v programe informácie sa nachádzajú v balíčku **bakalarka.Utilities.dataPack ,** súbory majú nasledovnú hierarchiu:

- V podpriečinku models sa nachádzajú popisy jednotlivých druhov modelov, o ktorých si viac povieme, pri ich modelovaní.
- V podpriečinku groups sa nachádzajú informácie o uhľovodíkoch, kde prípony súborov znamenajú nasledovné, uvedieme si to na príklade pre alkány (podpriečinok s názvom alkans) :
    - alkansInfo.txt – obsahuje základné informácie o skupine alkánov, Popis skupiny a nenamodelovaných zástupcov(pozri obrázok 8)
    - alkansAttri.txt – obsahuje vlastnosti skupiny alkánov
    - alkans.txt – obsahuje informácie o všetkých zástupcov skupiny alkánov, spolu s ich základnými informáciami
- V podpriečinku formulas sa nachádzajú popisy jednotlivých druhov vzorcov.



Obrázok 9 Hierarchia dátového modelu aplikácie

Najdôležitejšou triedou pre správu týchto textových súborov je trieda *DataLoader* , nachádzajúca sa taktiež v balíčku bakalarka.Utilities.dataPack. DataLoader je trieda, ktorá má definované viaceré statické podtriedy, ktoré spravujú a čítajú jednotlivé údaje. Každá takáto podtrieda má definovaný statický blok , v ktorom vždy načíta informácie

zo zodpovedných textových súborov a uloží si ich do globálnych premenných, ku ktorým potom v programe pristupujeme ak potrebujeme získať konkrétnu informáciu, ako príklad si uvedieme podtriedu *ModelsData*, ktorá spravuje informácie o modeloch:

```
1.  public static class ModelsData {
2.  
3.        public static String kalotModelLabel;
4.        public static String kalotInfo;
5.        public static String basModelLabel;
6.        public static String basInfo;
7.        public static String stickModelLabel;
8.        public static String stickInfo;
9.  
10.       static {
11.           InputStream urlKalotModel = DataLoader.class.getResourceAsStream("model
    s/kalotovyModelInfo.txt");
12.           InputStream urlBaSmodel = DataLoader.class.getResourceAsStream("models/
    gulickovyModelInfo.txt");
13.           InputStream urlStickModel = DataLoader.class.getResourceAsStream("model
    s/trubickovyModelInfo.txt");
14.           try {
15.               BufferedReader br = new BufferedReader(new InputStreamReader(urlKal
    otModel, "UTF-8"));
16.  
17.               kalotModelLabel = br.readLine();
18.               kalotInfo = br.readLine();
19.  
20.               br = new BufferedReader(new InputStreamReader(urlBaSmodel, "UTF-
    8"));
21.               basModelLabel = br.readLine();
22.               basInfo = br.readLine();
23.  
24.               br = new BufferedReader(new InputStreamReader(urlStickModel, "UTF-
    8"));
25.               stickModelLabel = br.readLine();
26.               stickInfo = br.readLine();
27.  
28.           } catch (Exception ex) {
29.               Logger.getLogger(DataLoader.class.getName()).log(Level.SEVERE, null
    , ex);
30.           }
31.       }
```

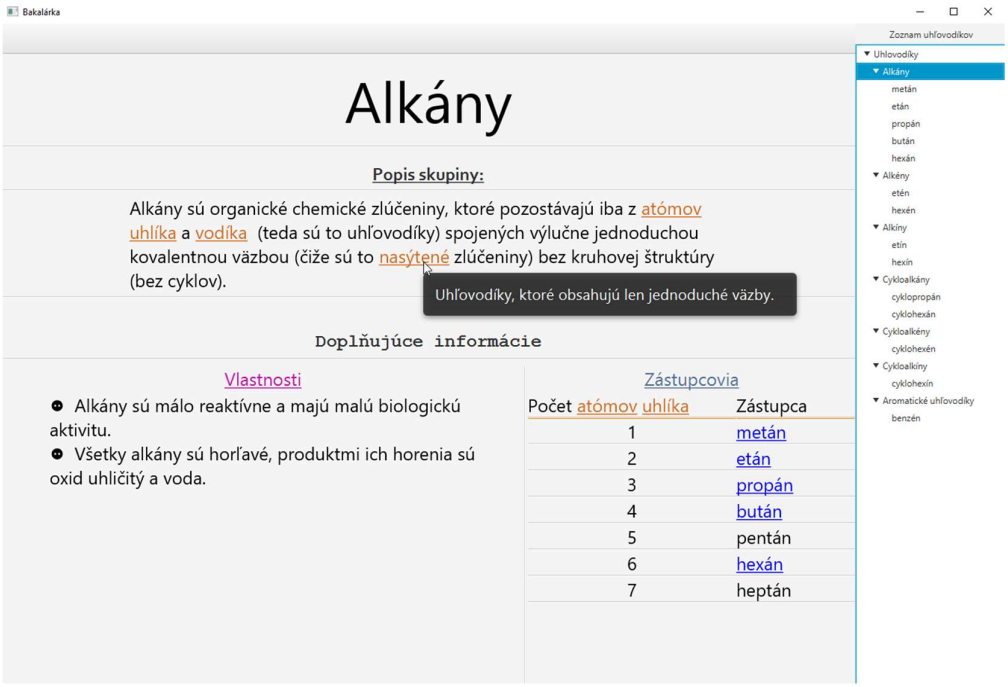
S takto definovanou triedou jednoducho pristúpime k jej globálnym statickým premenným :

```
1.  String kalotInfo = DataLoader.ModelsInfo.kalotInfo;
```

4.2 Tooltipy

Tooltipy sú textové vysvetlivky k niektorým potomkom triedy Node, ktoré sa majú zobraziť ak užívateľ vstúpi kurzorom do priestoru vyhradenom týmto potomkom.

JavaFX ponúka jednoduchý mechanizmus na ich tvorbu. Je ním funkcia *Tooltips.install(Node,String)* , kde prvým parametrom je nejaký potomok triedy Node, v našej aplikácií ním je len objekt Text, a druhým parametrom je String, ktorý sa má zobraziť po vstupe kurzorom. Ako príklad si uvedieme tooltip pre slovíčko „nasýtené“ :



Obrázok 10 Príklad tooltipu

Avšak tento mechanizmus má jednu nevýhodu, tooltip sa zobrazí až po nejakom časovom intervale, ktorý nie je pre užívateľa prirodzený, takže si zobrazovanie tooltipu musíme vyriešiť my v kóde nasledovne:

```
1.  public static void bindTooltip(final Node node, final Tooltip tooltip) {
2.          node.setOnMouseMoved(e -> {
3.              tooltip.show(node, e.getScreenX(), e.getScreenY() + 15);
4.          });
5.          node.setOnMouseExited(e -> {
6.              tooltip.hide();
7.          });
8.      }
```

Ďalším problémom pre tvorbu tooltipu je ten, že každý text, ktorý má byť zobrazený musí byť atribútom JavaFX triedy *Text*, to znamená, že musíme prísť na to ako z nejakej dlhej vety vytvoriť objekty Text tak aby bolo kľúčové slovo pre zobrazenie tooltipu v samostatnom objekte Text.

Zlým riešením je pre každé slovo vety vytvoriť jeden objekt Text, čo je zbytočne pamäťovo náročné. **Tak sme prišli na nasledujúci algoritmus**, ktorý vráti *ObservableList<Text>* **s najmenším možným počtom vytvorených objektov Text** s tým, že každému kľúčovému slovo zo vstupného textu vytvorí tooltip:

```
1.  public static ObservableList<Text> constructText(String baseText){
2.          ObservableList<Text> returnList = FXCollections.observableArrayList();
3.          String[] words = baseText.split(" |\\.|,|!|-|:|");
4.          String remeaningPart = baseText;
5.
6.          for(String word : words){
7.              if(keyWordsMap.containsKey(word.toLowerCase())){
8.
9.                  String tooltipDefinition = keyWordsMap.get(word.toLowerCase());

10.                 String previousPart = remeaningPart.substring(0, remeaningPart.in
    dexOf(word));
11.                 Text textNonKeyWord = new Text(previousPart);
12.                 Text keyWord = new Text(word);
13.                 Styler.styleAsTooltip(keyWord);
14.                 Tooltip tt = new Tooltip(tooltipDefinition);
15.                 tt.setWrapText(true);
16.                 tt.setPrefWidth(500);
17.                 ToolTips.bindTooltip(keyWord, tt);
18.                 returnList.addAll(textNonKeyWord,keyWord);
19.                 remeaningPart = remeaningPart.substring(remeaningPart.indexOf(wor
    d)+word.length());
20.             }
21.         }
22.         Text textRemaining = new Text(remeaningPart);
23.         returnList.add(textRemaining);
24.         return returnList;
25.     }
26.
```

Algoritmus rozdelí vstupný reťazec na slová, ak nájde kľúčové slovo, tak z reťazca pred kľúčovým slovom spraví objekt Text a reťazec za kľúčovým slovom si odloží. Tieto kroky opakuje až do konca vstupného reťazca, čím zaistí najmenší počet vytvorených objektov. Na zistenie či je dané slovo kľúčovým využívame ***hashmap keyWordsMap*** v ktorej kľúčom je kľúčové slovo pre tooltip, a hodnotou je samotný text tooltipu. Túto mapu plníme v statickom bloku triedy ToolTips a plníme ju z textového súboru tooltips.txt, ktorý sa nachádza v balíčku bakalarka.Utilities. Metóda *Styler.styleAsTooltip* nám text prefarbí na oranžovú farbu a podčiarkne text (viď. Obrázok 10).

Uvedená metóda nám slúži na vytváranie takmer všetkých Text objektov v applikácií.

4.3 Stromové odkazy

Na obrázku 10 si môžeme všimnúť slová modrým písmom a podčiarknutím. Kliknutie na tieto slová simuluje vybratie položky s tým istým názvom v TreeView na pravej strane.

Tieto odkazy používame pri informáciách o skupinách uhľovodíkov a to konkrétne ako odkazy na konkrétnych zástupcov týchto skupín.

Implementácia je veľmi jednoduchá, vytvoríme novú triedu, ktorá bude potomkom Text v našom prípade sa trieda volá *TreeLink* s jediným vlastným atribútom, ktorým je položka *TreeItem<TreeItemViewable>*, TreeItem je trieda, ktorá predstavuje položku stromu, takže po kliknutí na tento TreeLink stačí v spomínanom selectionModel TreeView hodnotu *selectedItemProperty* na kliknutý *TreeLink*, čím simulujeme priamo kliknutie na položku v strome.

Trieda *TreeLink*

```
1.  public class TreeLink extends Text {
2.      public TreeItem<TreeItemViewable> link;
3.      public TreeLink(TreeItem<TreeItemViewable> plink,String url){
4.          super(url);
5.          link = plink;
6.      }
7.  }
```

Vytvorenia TreeLink-ov v triede *HydroCarbonGroup*

```
1.  private void makeLinks() {
2.      for (TreeItem<TreeItemViewable> treeItem : supportedGroup) {
3.          TreeLink link = new TreeLink(treeItem, treeItem.getValue().toString());
4.          Styler.styleAsTreeLink(link);
5.          link.onMouseClickedProperty().set(e->{
6.              handleHClink(link);
7.          });
8.          links.add(link);
9.      }
10. }
11.
12. private void handleHClink(TreeLink link) {
13.     selectionModel.select(link.link);
14.
15. }
```

Následne máme v *ObservableList<TreeLink> links*, všetky odkazy na uhľovodíky, ktoré sú podporované našou aplikáciou, ktoré môžeme priamo zobrazovať keďže TreeLink je potomkom triedy Text. *Styler.styleAsTreeLink(link)* zafarbí text modrou farbou a podčiarkne.

5 Zobrazovanie vzorcov

Ďalšou dôležitou požiadavkou pre aplikáciu bolo zobrazovanie vzorcov pre vybraných spomínaných zástupcov z tabuľky 1. **Aplikácia podporuje štyri najzákladnejšie typy chemických vzorcov** a to konkrétne:

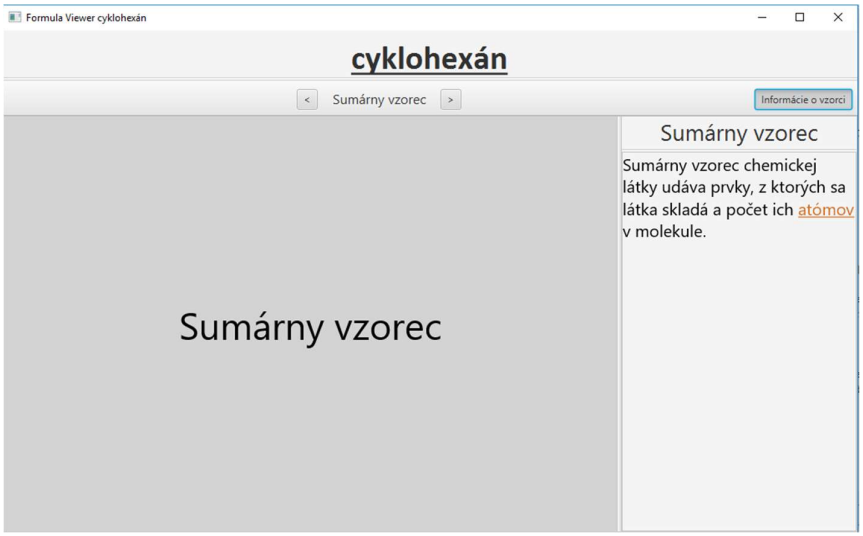
- **Sumárny vzorec** – hovorí iba o počte jednotlivých atómov v molekule
- **Empirický vzorec** – udáva pomer medzi počtom jednotlivých atómov v molekule
- **Racionálny vzorec** – ukazuje charakteristické skupiny a väzby medzi týmito skupinami
- **Štruktúrny vzorec** – ukazuje väzby medzi jednotlivými atómami a čiastočne aj ich usporiadanie v priestore

Túto požiadavku sme sa rozhodli realizovať pomocou otvorenia nového okna po stlačení tlačidla Zobraziť vzorce, ktoré môžeme vidieť pri prezeraní základných informácií konkrétneho uhľovodíku. Otvorenie nového okna prináša výhody z používateľského hľadiska a to napríklad, že používateľ nie je obmedzení na používanie iba jedného okna, čo znamená, že môže mať naraz otvorené vzorce viacerých uhľovodíkov a tak môže napríklad vidieť medzi nimi rozdiely.

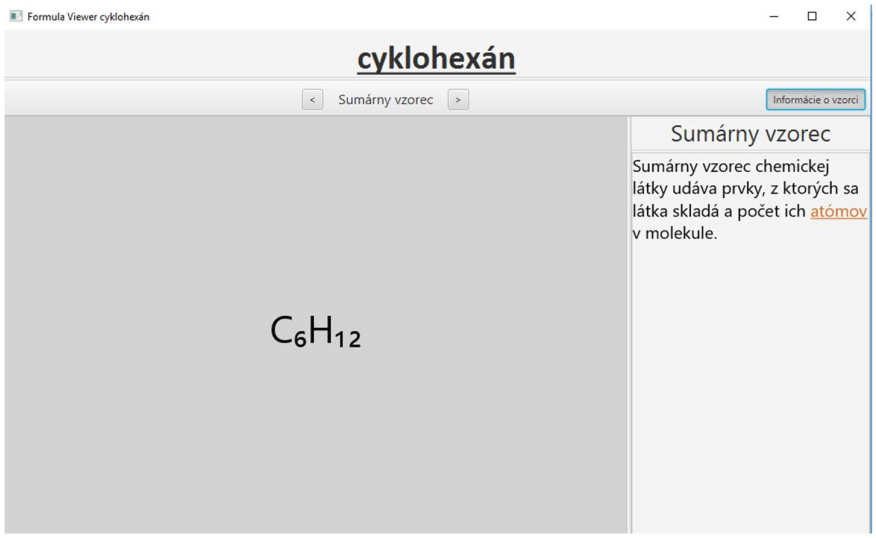
Rovnako ako okno pre zobrazovanie základných informácií je aj toto okno **plne responzívne**, čo znamená, že veľkosť okna je možno meniť, pričom sa zachová jeho definovaná štruktúra. Takisto veľkosť písma a veľkosť samotného zobrazenia vzorca je možné meniť klávesovou skratkou **CTRL+MOUSEWHEEL,** tento mechanizmus je realizovaný opäť pomocou mechanizmu bindingov o ktorých sme si povedali v predchádzajúcej kapitole.

Ako bolo spomínané v úvode, aplikácia by mala mať jednoduchý používateľský interface a preto sme sa rozhodli, že v jednom čase bude zobrazovaný len jeden vybraný druh vzorca a prepínanie medzi nimi môže používateľ realizovať pomocou tlačidiel v hornom panely okna alebo jednoducho pomocou **klávesových šípok LEFT/RIGHT**. Vzorce sú v tomto systéme zoradené podľa jednoduchosti od najjednoduchšieho, čo je vhodné pre používateľa začiatočníka pre, ktorého je smerovaná naša práca.

Ďalej sme sa rozhodli samotné zobrazovanie vzorca realizovať až po kliknutí na zobrazovaciu plochu alebo po stlačení medzerníka a to hlavne z pedagogického hľadiska, pretože samotné okno na zobrazovanie vzorcov obsahuje aj tlačidlo, ktorým zobrazíme informácie, ktoré hovoria čo má daný vzorca znázorňovať a tak ponúka používateľovi možnosť vyskúšať si vzorec podľa týchto informácií skonštruovať a našu aplikáciu použiť len na kontrolu.



Obrázok 11 Okno zobrazenia vzorcov bez zobrazovania vzorca



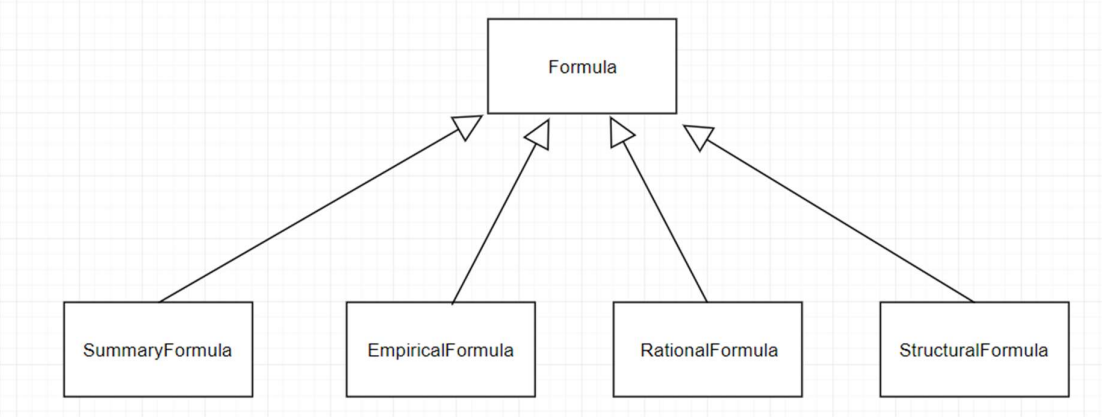
Obrázok 12 Okno zobrazenie vzorcov so zobrazením sumárneho vzorca

Taktiež sme sa rozhodli pri zobrazovaní vzorcov uhľovodíku, ktorý sa nachádza v skupine s nejakou násobnou väzbou (dvojitou alebo trojitou), zakomponovať tlačidlo,

ktoré zmení pozíciu tejto násobnej väzby v reťazci, keď je tlačidlo aktívne tak väzba je umiestnená na začiatku reťazca (medzi prvými dvoma atómami uhlíku), ak je tlačidlo neaktívne, tak väzba je umiestnená v strede reťazca. Táto funkcionalita používateľovi ukáže rôzne variácie jedného uhľovodíka, príklad si ukážeme keď budeme hovoriť o konkrétnych vzorcoch.

5.1 Spôsob implementácie vzorcov

Ak chceme aby program bol prehľadnejší a jednoduchšie sa vyvíjal a ak si uvedomíme skutočnosť, že každý druh vzorca musí mať funkciu v ktorej si vytvorí štruktúru ktorú budeme vo vhodnej chvíli vykresľovať na obrazovku tak nám z toho vyplynie spoločná vlastnosť každej triedy, ktorá bude zodpovedná za konkrétny druh vzorca, túto vlastnosť môžeme vyniesť do spoločného predka a ušetriť tak kód na zavolanie samotného vykreslenia tejto štruktúry. Túto dedičnosť znázornime UML diagramom :



Obrázok 13 UML diagram dedičnosti pre triedy vzorcov

Predkovi Formula definujeme abstraktnú triedu ***drawFormula(),*** ktorú si každý potomok implementuje podľa potreby a tak trieda, ktorá rieši samotné zobrazovanie, ktorou je v našej aplikácií *ResizableCanvas* nemusí vedieť konkrétnu triedu vzorca, ktorý ma zobraziť ale stačí mu spomínaný spoločný predok.

Ako sme spomínali v kapitole 4.1 Dátový model aplikácie, jedinou relevantnou informáciou o štruktúre uhľovodíku je počet jeho atómov uhlíku, to nám ale nestačí k úspešnému modelovaniu a vypočítaniu jeho vzorcov, keby modelujeme len acyklické štruktúry so samými jednoduchými väzbami tak by nám stačila len táto informácia.

Vieme však ešte zistiť do akej skupiny uhľovodík patrí, skupina nám povie či sa jedná o cyklickú alebo acyklickú štruktúru a takisto nám povie, či sa v štruktúre nachádza násobná väzba (viac v kapitole 1.4 Rozdelenie uhľovodíkov).

To znamená, že k dispozícií máme tri informácie:

- Počet atómov uhlíku
- Cyklická / Acyklická / Aromatická štruktúra
- Informácia či uhľovodík obsahuje nejakú násobnú väzbu

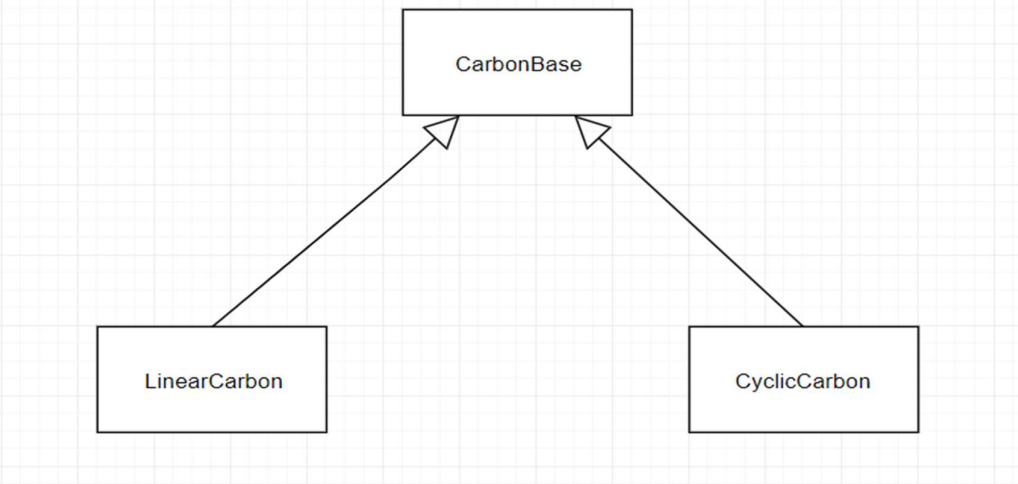
Tieto tri základné informácie nám stačia na úspešné namodelovanie a taktiež aj na výpočet samotných vzorcov všetkých vybraných zástupcov.

Predtým ako si povieme o spôsobe výpočtu týchto vzorcov, musíme si povedať niečo o samotnej reprezentácií uhľovodíku v našej aplikácií.

Hlavnou triedou, ktorá je zodpovedná za uhľovodík je už spomínaná trieda HydroCarbon. Trieda HydroCarbon si v **atribúte structure** drží štruktúru, ktorá zodpovedá atómovej štruktúre uhľovodíku.

Atribút **structure** je jednorozmerné pole objektov *CarbonBase*. Keďže uhľovodíky v našej aplikácií sú dvoch druhov a to cyklické(pod ktoré spadajú aj aromatické) alebo acyklické je CarbonBase spoločný predok tried *LinearCarbon* a *CyclicCarbon* :

- ***LinearCarbon*** – predstavuje jeden atóm uhlíku v acyklickom uhľovodíku
- ***CyclicCarbon*** – predstavuje jeden atóm uhlíku v cyklickom uhľovodíku



Obrázok 14 UML diagram tried Carbon

Predok *CarbonBase* obsahuje okrem iného dve dôležité jednorozmerné polia, ktorými sú :

- ***neighbors*** – jednorozmerné pole objektov CarbonBase, ktoré predstavuje ďalšie atómy uhlíku s ktorými je CarbonBase naviazaný, toto pole je o veľkosti 2, iná veľkosť nemá v našej aplikácií zmysel, atóm uhlíku má vždy maximálne dva susediace atómy uhlíku.
- ***bounds*** – pole dvoch integerov, ktoré predstavujú násobnosť väzby k susediacim atómom uhlíku

Takáto reprezentácia štruktúry pre účely našej aplikácie stačí. Všimnime si, že nikde nemáme informáciu o atómoch vodíku, vieme však, že aplikácia je zameraná len na atómy uhlíku a vodíku z tejto skutočnosti a z vlastností tvorenia väzieb o ktorých sme hovorili v kapitole 1.3 dokážeme počet atómov vodíku v molekule odvodiť pomocou jednoduchého výpočtu:

$$PH = 4 - (LV + PV)$$

Kde jednotlivé premenné majú nasledovný význam:

- **PH** – počet naviazaných atómov vodíku na konkrétny atóm uhlíku
- **LV** – násobnosť väzby atómu uhlíku k ľavému susedovi
- **PV** – násobnosť väzby atómu uhlíku k pravému susedovi

Ako príklad výpočtu uvedieme acyklický uhľovodík etán, etán má dva atómy uhlíku, etán patrí do skupiny alkánov, čo znamená, že všetky jeho väzby sú jednoduché (násobnosť je 1)

1. **Prvý atóm uhlíku etánu** – nemá ľavého suseda, má pravého suseda naviazaného jednoduchou väzbou z čoho vyplýva, že musí mať

$$PH = 4 - (0 + 1) = 3$$

2. **Druhý atóm uhlíku etánu** – má ľavého suseda naviazaného jednoduchou väzbou, pravého suseda nemá, čo znamená:

$$PH = 4 - (0 + 1) = 3$$

Na rovnakom princípe počítame v aplikácií počty atómov vodíku pre každý uhľovodík.

V ďalších podkapitolách si ukážeme príklady samotného zobrazenia vzorcov a to hlavne na príklade uhľovodíku s názvom hexín.

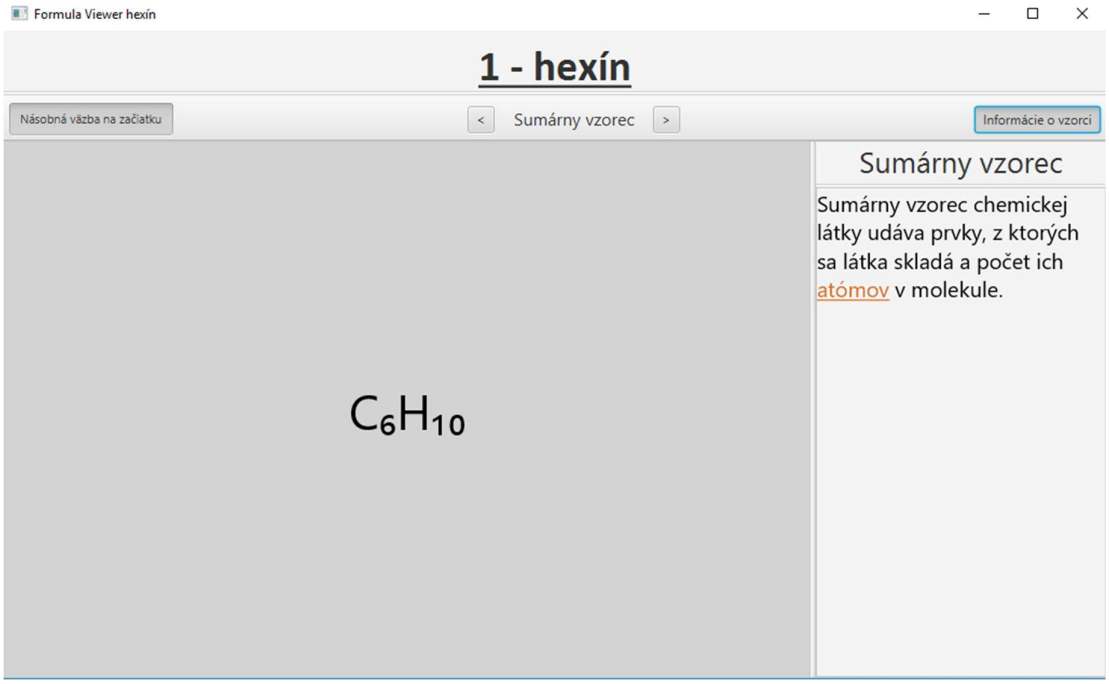
5.2 Sumárny vzorec - SummaryFormula

Spôsobom akým v aplikácií reprezentujeme uhľovodík nám umožňuje veľmi jednoducho vytvoriť sumárny vzorec.

Informáciu o počte atómov uhlíku máme z textových súborov, o ktorých sme hovorili v podkapitole 1.4, takže k správnemu sumárnemu vzorcu potrebujeme už len počet atómov vodíku. Po aplikovaní výpočtu z predchádzajúcej kapitoly na každý jeden atóm uhlíku a sčítaním týchto hodnôt dostaneme celkový počet atómov vodíka v uhľovodíku.

Keďže všetky atómové počty v chémií sa píšu ako dolný index, musíme zaistiť premenu čísla na UTF-8 reťazec, ktorý zodpovedá dolným indexom reprezentujúcim toto číslo, na túto premenu nám slúži statická metóda *toSubscript(int num),* ktorá vráti daný dolný index. Metóda sa nachádza v triede *SubscriptHelper* v balíku Utilities.

Teraz, keď už máme poskladaný reťazec, stačí ho zobraziť na obrazovku v spomínanej metóde ***drawFormula().***



Obrázok 15 Sumárny vzorec hexínu

Stlačenie tlačidla Násobná väzba na začiatku nemá žiaden efekt na sumárny vzorec, keďže počet atómov sa nemení, mení sa len ich rozloženie.

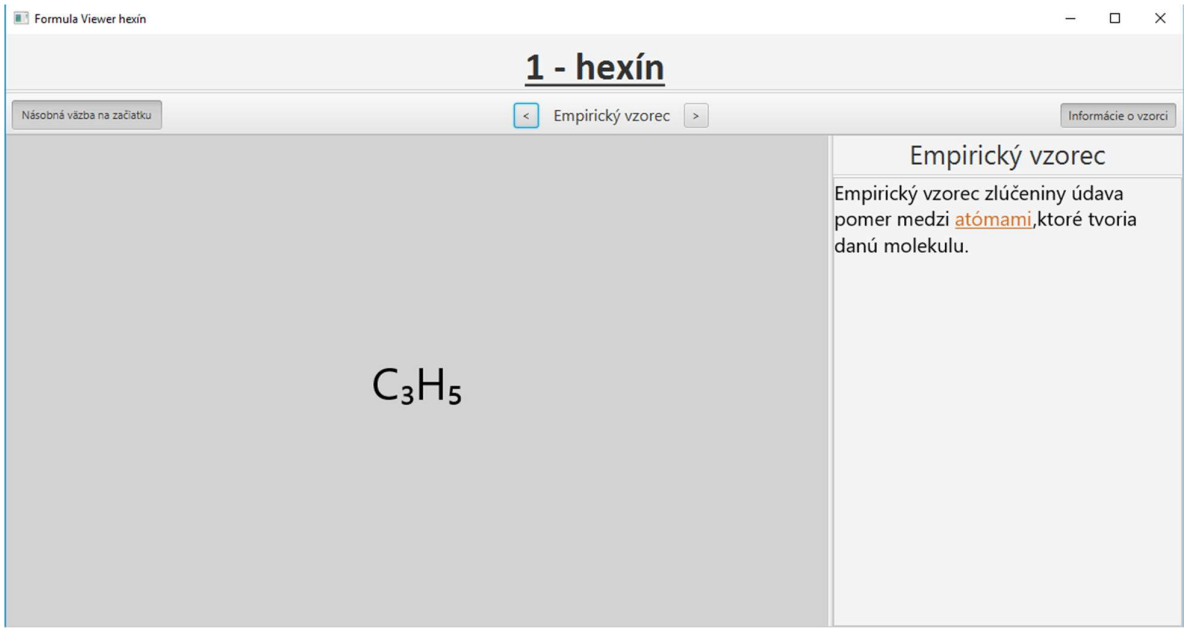
5.3 Empirický vzorec – EmpiricalFormula

Princíp tvorby empirického vzorca je rovnaký ako pri sumárnom vzorci s jednou výnimkou, neukazujeme úplne počty jednotlivých atómov, ale musíme ukázať ich pomer, na výpočet najjednoduchšieho pomeru medzi dvoma číslami existuje mnoho algoritmov, my sme si vybrali nasledujúci:

1. Nájdeme najväčšieho spoločného deliteľa dvoch čísel – pre tento krok sme implementovali Euklidov algoritmus

2. Vydelíme obe čísla týmto deliteľom

Následne rovnako ako pri sumárnom vzorci prevedieme čísla na dolné indexy, spravíme výstupný reťazec a ten vykreslíme pomocou metódy *drawFormula()*.



Obrázok 16 Empirický vzorec hexínu

5.4 Racionálny vzorec – RationalFormula

Na rozdiel od predchádzajúcich dvoch druhoch vzorcov, **racionálny vzorec nemôžeme vyjadriť len jednoduchými znakovými reťazcami**, pretože racionálny vzorec obsahuje aj znázornenie väzieb v štruktúre uhľovodíku takže vzorec musíme vykresliť pomocou 2D grafických funkcií, ktoré JavaFX ponúka. Jednotlivé väzby tak znázorňujeme čiarou/čiarami podľa násobnosti väzby, ktorú/é vykreslíme na ***ResizableCanvas*** pomocou funkcie *drawLine()*.

Racionálny vzorec má znázorňovať časti z ktorých sa skladá uhľovodík a taktiež znázorňovať aj väzby medzi týmito časťami, tieto časti získame z našej programovej štruktúry veľmi jednoducho, opäť použijeme spomínaný výpočet pre počet vodíkov, tentokrát ale nesčítame tieto hodnoty ako v prípade sumárneho alebo empirického vzorca, ale použijeme ich každú samostatne, tým získame spomínané časti molekuly, ostáva nám už len tieto časti prepojiť spomínanými čiarami.

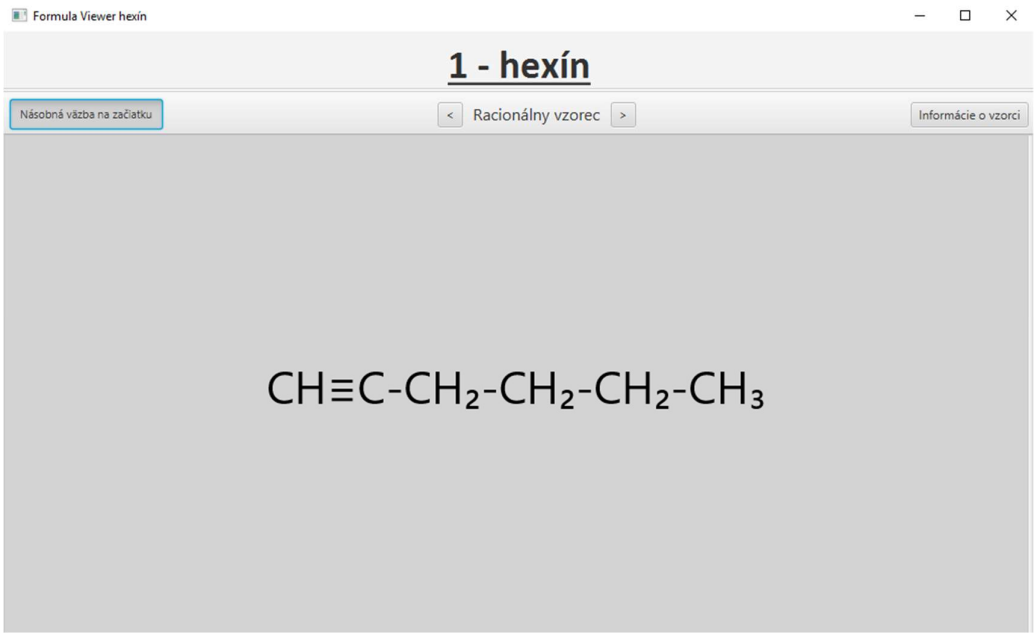
Je tu však ešte jeden podstatný rozdiel medzi racionálnym vzorcom a medzi predchádzajúcimi dvoma vzorcami, predchádzajúce vzorce vyzerali rovnako aj v prípade cyklického alebo acyklického uhľovodíku, táto vlastnosť, ale v prípade racionálneho vzorca neplatí a tak musíme rozlišovať medzi týmito dvoma prípadmi.

Acyklický racionálny vzorec totiž znázorňujeme na jednej priamke, čo v prípade cyklického neplatí, cyklický racionálny vzorec musí znázorňovať aj cyklus, ktorý atómy uhlíku vytvárajú v našom programe rozlišujeme tri druhy cyklov a to konkrétne:

- **Rovnostranný trojuholník**
- **Pentagon**
- **Hexagon**

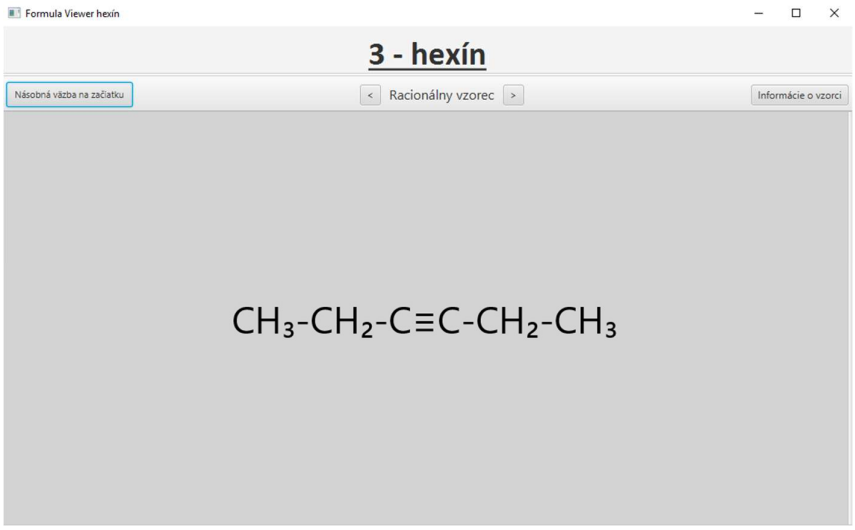
Cyklus, ktorý cyklický uhľovodík vytvára zistíme jednoducho podľa počtu atómov uhlíku, ak sú atómy tri, tak vytvárajú trojuholník ak ich je päť tak vytvárajú pentagon a ak je ich šesť vytvárajú hexagon. Túto skutočnosť v programe vyjadruje *Enum CyclicShape* v triede *HydroCarbon*, ktorá nám povie na základe počtu atómov uhlíku o ktorý cyklus sa jedná.

Ak poznáme cyklus, ktorý cyklický uhľovodík vytvára, môžeme jednoducho vypočítať pozície jednotlivých časti uhľovodíku. Tieto pozície predstavujú x a y súradnice na plátne v ktorom ich vykresľujeme, ktorým je spomínaný ResizableCanvas. Počítanie týchto pozícií realizuje statická metóda *calculateCyclicPoints2D*, ktoré vráti pole objektov Point2D, kde Point2D prezentuje 2D bod, metóda sa nachádza v triede *GeometryHelper* v balíku Utilities.



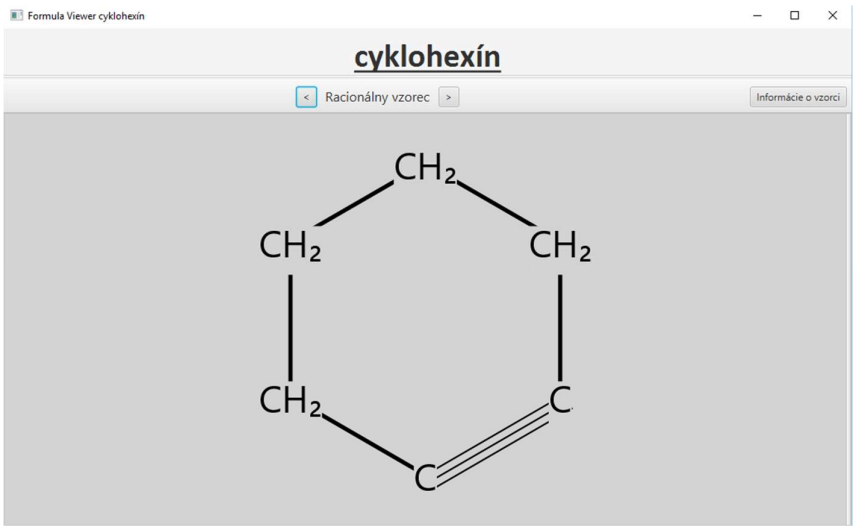
Obrázok 17 Racionálny vzorec 1-hexínu

Keďže racionálny vzorec nám znázorňuje aj samotné väzby v molekule, môžeme si tak všimnúť efekt tlačidla Násobná väzba na začiatku. Na obrázku 17 vidíme, racionálny vzorec 1-hexínu, predpona 1- znamená, že násobná väzba sa nachádza medzi prvým párom susediacich uhlíkov, teda je aktívne spomínané tlačidlo.



Obrázok 18 Racionálny vzorec 3-hexínu

Ako sme už spomínali neaktívne tlačidlo Násobná väzba na začiatku, spôsobí, že násobná väzba sa vycentruje teda umiestni sa medzi pár uhlíkov, ktorý je v strede reťazca. Tento efekt vidíme na obrázku 18.



Obrázok 19 Racionálny vzorec cyklohexínu

5.5 Štruktúrny vzorec - StrucutralFormula

Štruktúrny vzorec je takmer rovnaký ako racionálny, ale musíme ešte zdôrazniť na väzby medzi samotnými uhlíkmi a atómami vodíku. Opäť budeme tieto väzby znázorňovať čiarami ako v prípade racionálneho vzorca.

Väzby medzi uhlíkmi budeme znázorňovať rovnako ako v prípade racionálneho vzorca, opäť zistíme, či ide o acyklickú alebo cyklickú štruktúru, ak ide o cyklickú, zistíme tvar cyklu, ktorý atómy uhlíku tvoria na základe spôsobu o ktorom sme hovorili.

Ako bolo spomínané v prípade štruktúrneho vzorca zdôrazňujeme aj čiastočné priestorové usporiadanie všetkých atómov tvoriacich molekulu, návod ako vypočítať toto usporiadanie nám dáva teória **VSEPR (Valence Shell Electrion Pair Repulsion),** ktorá nám veľmi zjednodušene povedané, pomáha vypočítať tvar, ktorý väzby atómu vytvárajú na základe toho koľko ich je, v našej aplikácií sa stretávame s nasledujúcimi tvarmi:

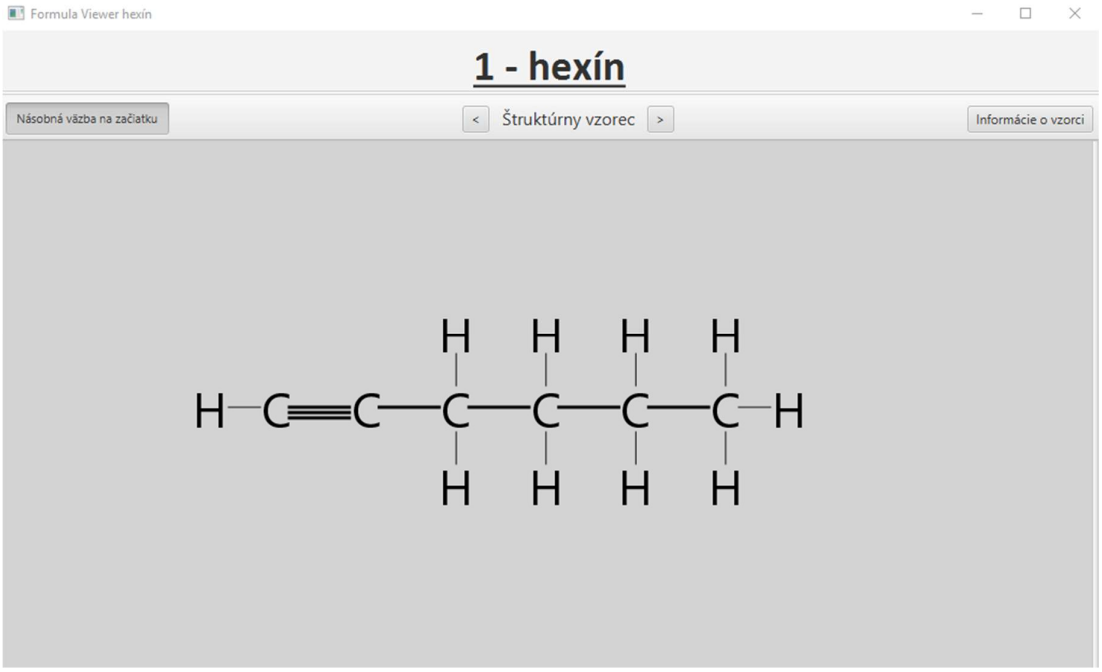
- **Priamka** – ak sú väzby v súčte dve tak tvoria priamku, takže uhol medzi týmito väzbami je 180°
- **Rovnostranný trojuholník** – ak sú väzby tri vytvárajú v jednej rovine trojuholník, uhly medzí väzbami sú tak 120°
- **Pravidelný štvorsten** – ak sú väzby štyri tvoria pravidelný štvorsten, kde uhly medzi väzbami sú 109.5°

Tieto tvary zachytáva enum subMoleculeShape v triede *HydroCarbon*, takže nie je problém zistiť, ktorý tvar vytvárajú spomínané väzby.

Treba však zdôrazniť, že štruktúrny vzorec je 2D štruktúrou, čo znamená, že, pravidelný štvorsten tak nemôžme presne vyjadriť a tak tento útvar v štruktúrnom vzorci reprezentujeme ako jednoduchý štvorec, čo znamená, že uhol medzi väzbami je 90°.

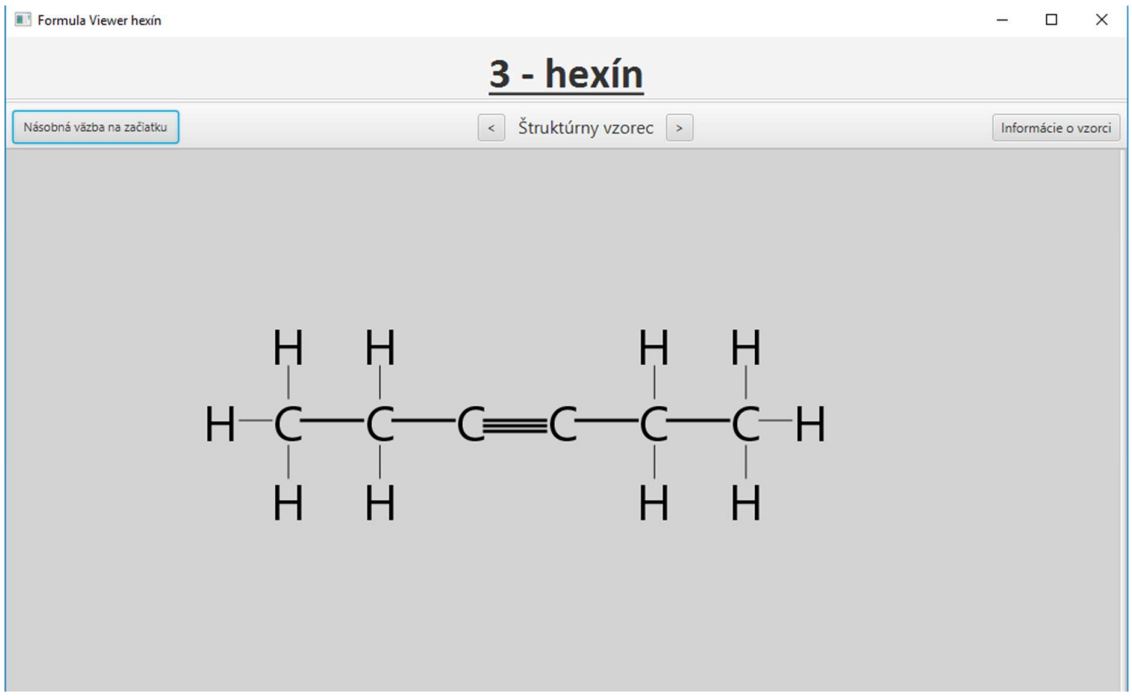
Priamku a rovnostranný trojuholník nie je problém vyjadriť presne, pretože sú to rovinné útvary.

Na počítanie pozícií atómov uhlíku v súradnicovom priestore ResizableCanvas, používame opäť funkciu *calculateCyclicPoints2D* a na počítanie pozícií vodíkov funkciu *calculateHpositions* taktiež v triede *GeometryHelper*.



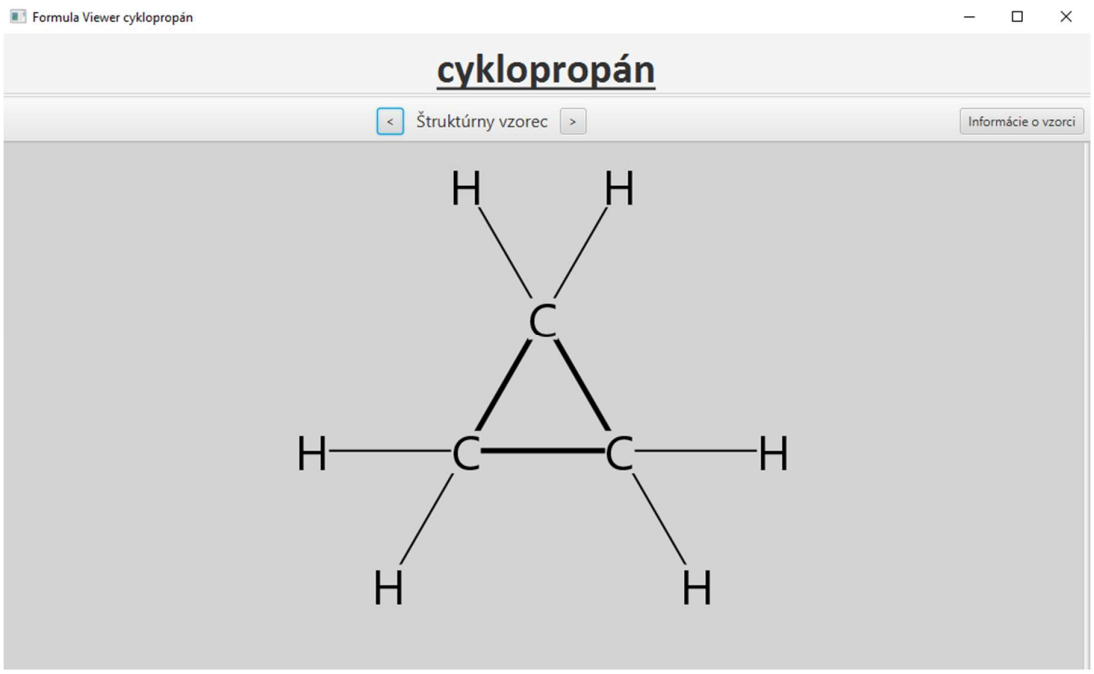
Obrázok 20 Štruktúrny vzorec 1-hexínu

Rovnako ako v prípade racionálneho vzorca má tlačidlo Násobná väzba na začiatku vplyv na vzorec, pre príklad si uvedieme štruktúrny vzorec 3-hexínu.

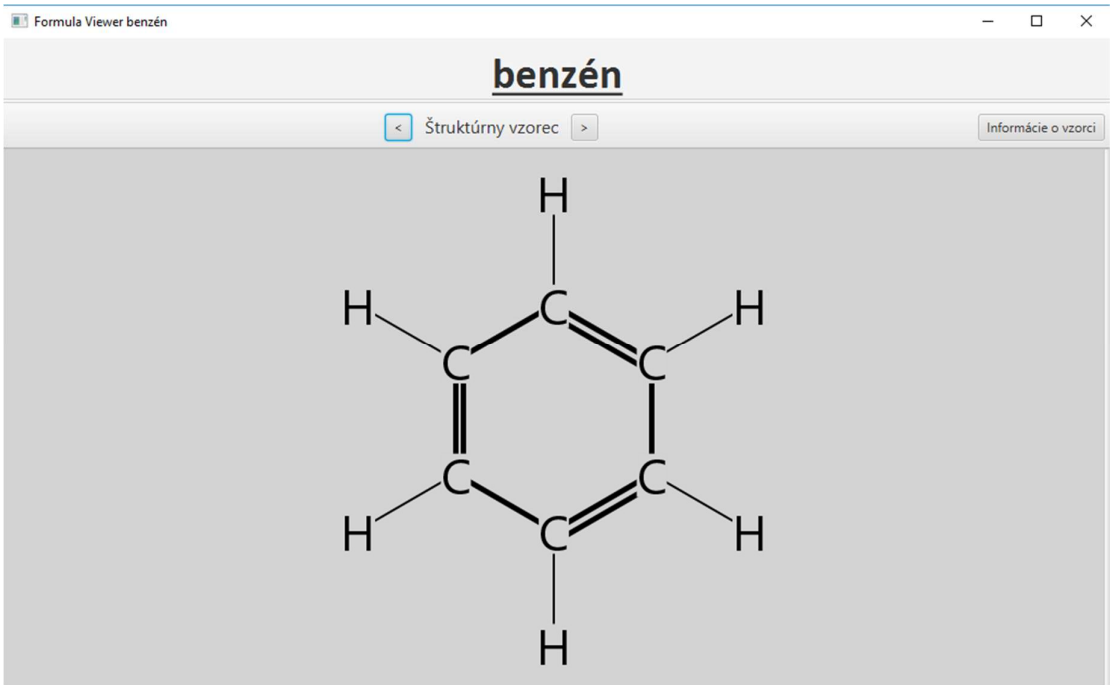


Obrázok 21 Štruktúrny vzorec 3-hexínu

Pre príklad štruktúrneho vzorca cyklického uhľovodíku uvádzame cyklopropán a benzén.



Obrázok 22 Štruktúrny vzorec cyklopropánu



Obrázok 23 Štruktúrny vzorec benzénu

6 Zobrazovanie modelov

Vo všeobecnosti majú modely v chémií poukázať na rôzne vlastnosti, ktoré sú momentálne záujmom skúmania danej chemickej zlúčeniny, z čoho vyplýva, že druhov modelov je veľmi veľa a neustále pribúdajú nové, preto sme modelom v našej aplikácií pridelili nasledujúcu úlohu.

Úlohou modelov v našej aplikácií je grafické znázornenie priestorového usporiadania atómov konkrétneho uhľovodíku na základe tejto požiadavky sme sa rozhodli do aplikácie zakomponovať tri druhy modelov a to konkrétne:

- **Kalótový model**
- **Guľôčkový model**
- **Trubičkový model**

O uvedených modeloch a spôsobu ich implementácie a prípadne o ich výhodách/nevýhodách si povieme v ďalších podkapitolách.

Samotné zobrazovanie týchto vzorcov sme sa taktiež ako pri zobrazovaní vzorcov rozhodli realizovať v samostatnom okne po stlačení tlačidla Zobrazenie modelov, ktoré je zobrazené pri prezeraní základných informácií o konkrétnom uhľovodíku. Najpodstatnejším implementačným detailom okrem samotného modelovania uhľovodíku je použitie JavaFX objektu s názvom *SubScene*, ktorý nám v skratke umožňuje kombinovanie 2D a 3D obsahu, vďaka čomu získame prostriedok na tvorenia komplexných aplikácií.

Na modelovanie uhľovodíka v našej aplikácií používame nasledovné natívne JavaFX 3D objekty:

- ***Sphere*** – guľa, v aplikácií guľou prezentujeme atóm
- ***Cylinder*** – valec, v aplikácií valcom prezentujeme väzbu medzi atómami.

Ďalším užitočným mechanizmom v 3D zobrazovaní pomocou JavaFX je kamera. Kamera predstavuje používateľove zorné pole v 3D scéne. Kamerou ale aj inými objektami v 3D scéne môžeme hýbať pomocou priestorových transformácií z ktorých v aplikácií využívame nasledujúce dve základné:

- **Posunutie (*Translate*)** – predstavuje zmenu pozície objektu v priestore

- **Rotácia (*Rotate*)** – predstavuje otočenie v priestore

Uvedené dve transformácie nám nielenže umožňujú samotné modelovanie uhľovodíkov ale ich vhodnými aplikáciami na spomínanú kameru vieme umožniť používateľovi pohybovať sa v priestore a tak si môže pozrieť model z rôznych uhľov a vzdialeností.

Ovládanie spomínaného pohybu je realizované pomocou myši:

- Podržaním ľavého tlačidla a následným pohybom myši realizujeme rotáciu kamery
- Koliečkom meníme vzdialenosť kamery
- Stlačením a následným držaním koliečka a pohybom myši realizujeme pohyb v priestore

Toto ovládanie je zaužívané v rôznych 3D softvéroch čo je dôvodom voľby tohto rozloženia.

Atómy v modeloch prezentujeme ich všeobecne zaužívanými farbami a to nasledovne:

- **Uhlík** – čierna farba
- **Vodík** – biela farba

, keďže znázornenie väzieb v modeloch sa líši od druhu modelu, budeme o nich hovoriť až neskôr.

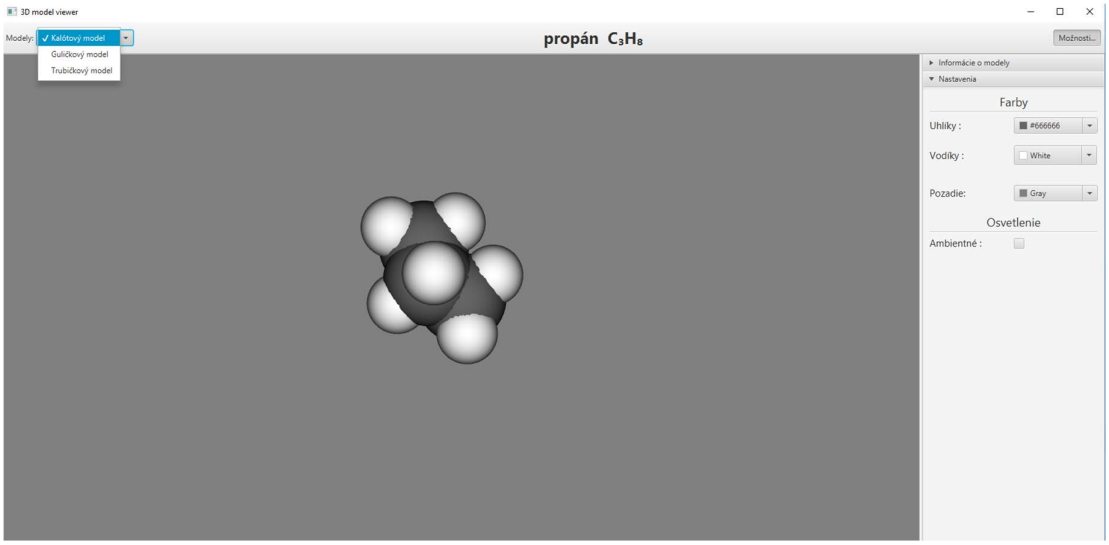
V samotnom zobrazovacom okne sme sa rozhodli zakomponovať aj panel Možnosti, ktorý obsahuje informácie o danom druhu modelu a taktiež obsahuje aj zopár jednoduchých nastavení, ktorými si používateľ môže meniť:

- **Farby atómov**
- **Farba pozadia v scéne**
- **Typ osvetlenia v scéne**

Tieto nastavenia sú opäť realizované pomocou mechanizmu bindingov, ktorý je vysvetlený v kapitole 4 Zobrazovanie základných informácií.

Prepínanie medzi druhmi modelov, používateľ realizuje pomocou jednoduchého menu v ľavej hornej časti okna.

Ako každé okno v aplikácií je aj okno pre zobrazovanie modelov plne responzívne.



Obrázok 24 Okno pre modely so zobrazeným kalótovým modelom propánu

Okno taktiež ako pri vzorcoch obsahuje tlačidlo Násobná väzba na začiatku, ktoré zmení pozíciu násobnej väzby, tlačidlo sa zobrazuje iba ak má význam ho zobraziť.

Najdôležitejšou triedou celého zobrazovanie modelov je trieda *ContentModel3D*, ktorá je zodpovedná za celé okno, teda má na starosti spomínanú kameru a taktiež prepína, ktorý model má byť práve zobrazený.

Všetko čo sa modelovania týka je umiestnené v balíku view3D.

6.1 Princíp tvorenia modelov v aplikácií

Ako sme už spomínali, možné tvary, ktoré vytvárajú väzby atómu uhlíku sú v našej aplikácií nasledovné:

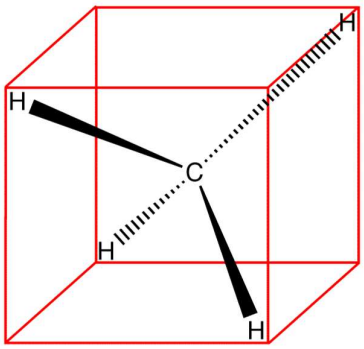
- **Priamka** – ak sú väzby v súčte dve tak tvoria priamku, takže uhol medzi týmito väzbami je 180°
- **Rovnostranný trojuholník** – ak sú väzby tri vytvárajú v jednej rovine trojuholník. Uhly medzí väzbami sú tak 120°

- **Pravidelný štvorsten** – ak sú väzby štyri tvoria pravidelný štvorsten, kde uhly medzi väzbami sú 109.5°

Ak chceme aby model čo najviac zodpovedal skutočnému usporiadaniu atómov v priestore, musíme zakomponovať do 3D zobrazovania uvedené uhly presne, čo je ale komplikované ak by sme chceli dané uhly tvoriť spomínanými rotáciami, pretože nájsť os okolo ktorej treba spraviť danú rotáciu je zložitá úloha. Takže, dané uhly nebudeme tvoriť rotáciami ale takými posunutiami(*translate*) po ktorých bude zvieraný uhol aj tak presný.

Základnou úvahou je uvedomenie si, že každý z daných útvarov sa dá umiestniť do vnútra kocky.

Ako príklad uvádzame uhľovodík metán, ktorého väzby tvoria pravidelný štvorsten.

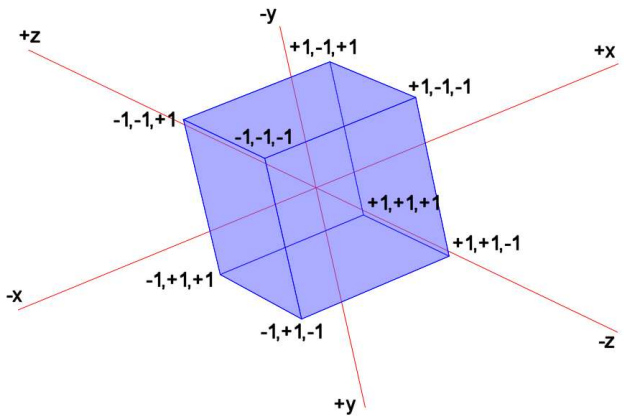


Obrázok 25 pravidelný štvorsten tvorený väzbami metánu

Je očividné, že rovnostranný trojuholník a priamka sa taktiež dajú umiestniť do kocky tak, aby centrálny atóm(uhlík) bol v strede tejto kocky, podobne ako je to na obrázku č.25.

Z týchto skutočností vyplýva to, že na namodelovanie akejkoľvek štruktúry skladajúcej sa z uvedených útvarov stačí postupne počítať súradnice rohov do ktorých umiestnime susediace atómy práve počítaného uhlíku, postupným spájaním takýchto útvarov, získame celkovú štruktúru uhľovodíku.

Vypočítanie súradníc týchto rohov je veľmi triviálne, ak si predstavíme kocku s dĺžkou strany 2, ktorej stred je umiestnený v bode [0,0,0] potom súradnice rohov takejto kocky sú nasledovné:



Obrázok 26 Kocka umiestnená v bode [0,0,0]

Podľa rovnakej konštrukcie ako na obrázku č.26 potom jednoducho vieme odvodiť rohy kocky s určitou dĺžkou strany s tým, že stred kocky je v ľubovoľnom bode priestoru.

Tento základný princíp modelovania funguje ako na acyklické tak aj na cyklické štruktúry s niekoľkými rozdielmi:

- **Acyklická štruktúra** – rohy do ktorých umiestnime susediace atómy konkrétneho uhlíku, musia byť opačné ako boli rohy predchádzajúceho uhlíku, inak by susediace atómy ležali na jednej priamke, čo ale nie je realitou.
- **Cyklická štruktúra** - v prípade, že atómy uhlíku tvoria cyklus, musíme rohy počítať tak aby v konečnom dôsledku vznikol tento cyklus.

Práve kvôli týmto dvom rozdielom sme sa rozhodli vytvoriť dve samostatné triedy, kde každá bude modelovať modeli cyklické/acyklické, týmito triedami sú:

- ***LinearStructureModeler*** – modeluje acyklické(lineárne) štruktúry
- ***CyclicStructureModeler*** – modeluje cyklické štruktúry

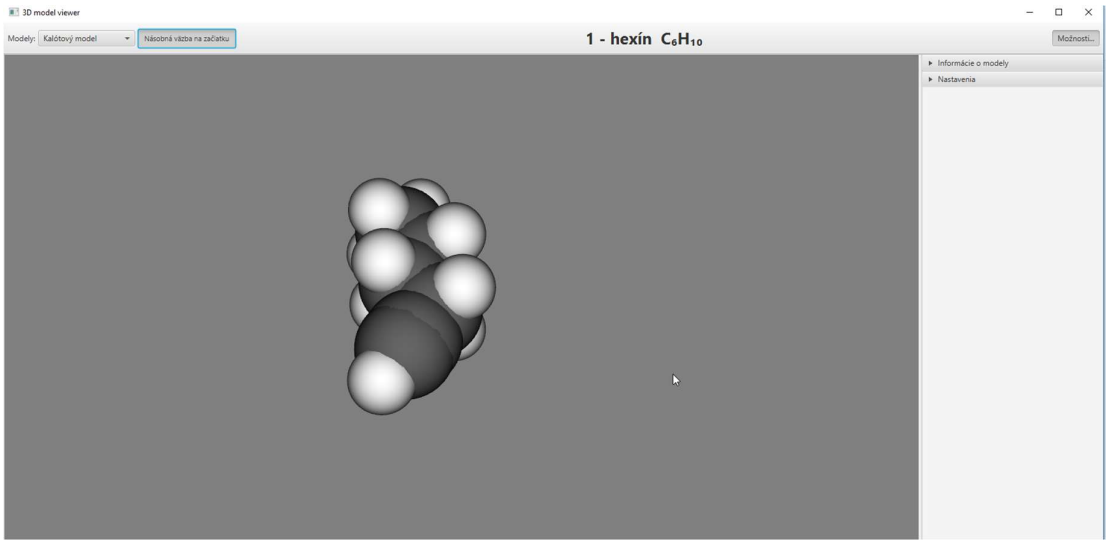
Na kontrolu či výsledná štruktúra modelu uhľovodíku zodpovedá realite sme použili známy portál zaoberajúci sa chémiou [PubChem].

6.2 Kalótový model

Kalótový model znázorňuje väzby medzi atómami tak, že atómy tvoriace väzbu sú umiestnené v sebe. Násobnosť väzby sa v modeli znázorňuje veľkosťou častí, ktoré sú umiestnené v sebe.

Tento model sa pomocou spomínaného princípu rohov kocky modeluje najjednoduchšie, stačí určiť vzdialenosť rohov, tak aby sa v prípade väzby rohy prekrývali.

Tento spôsob znázorňovania väzieb nie je príliš vhodný čo si ukážeme na príklade hexínu.

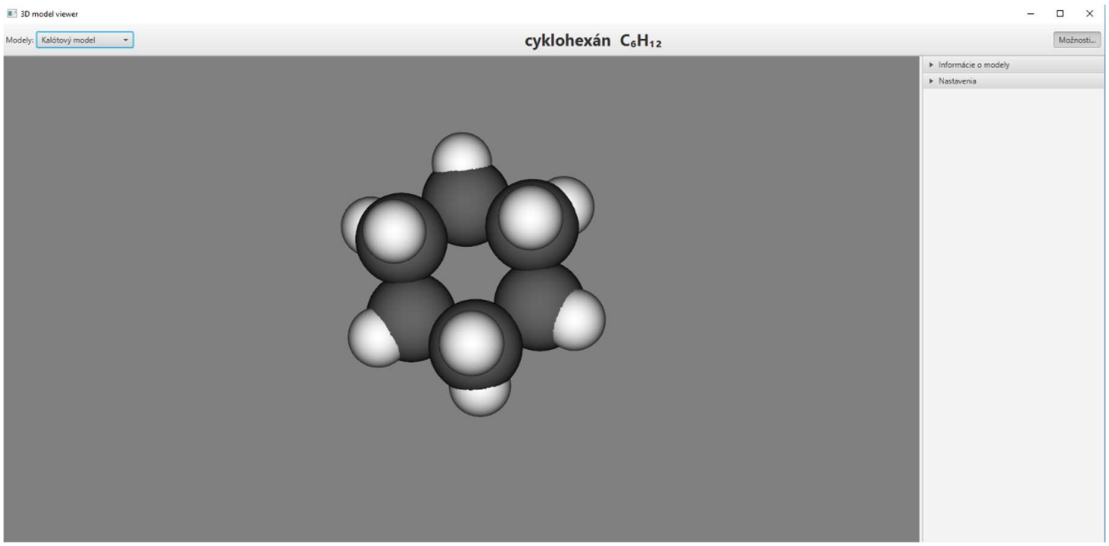


Obrázok 27 Kalótový model 1-hexínu

Na obrázku č.27 si môžeme všimnúť ako vyzerajú väzby kalótového modelu, ak sa bližšie pozrieme na druhý atóm uhlíku, na prvý pohľad to vyzerá tak , ako keby tvoril trojitú väzbu s prvým (čo je pravda) ale aj s tretím atómom uhlíku (čo nie je pravda).

Vyzerá to tak preto, lebo jeho značná časť je umiestnená v prvom atóme uhlíku. Práve toto je spomínanou nevýhodou kalótového modelu.

Cyklický kalótový model ukážeme na príklade cyklohexánu.

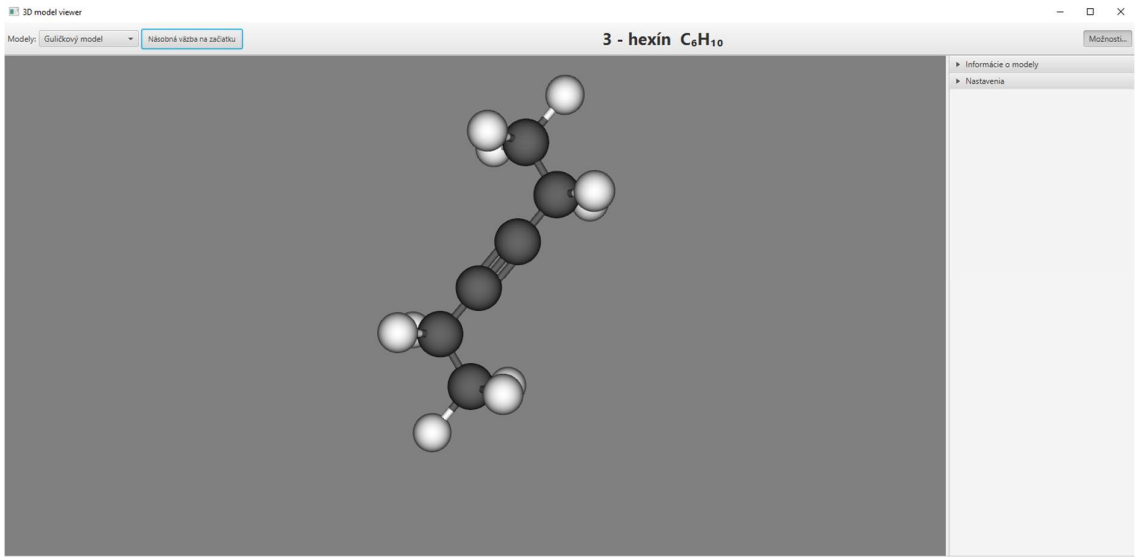


Obrázok 28 Kalótový model cyklohexánu

6.3 Guľôčkový model

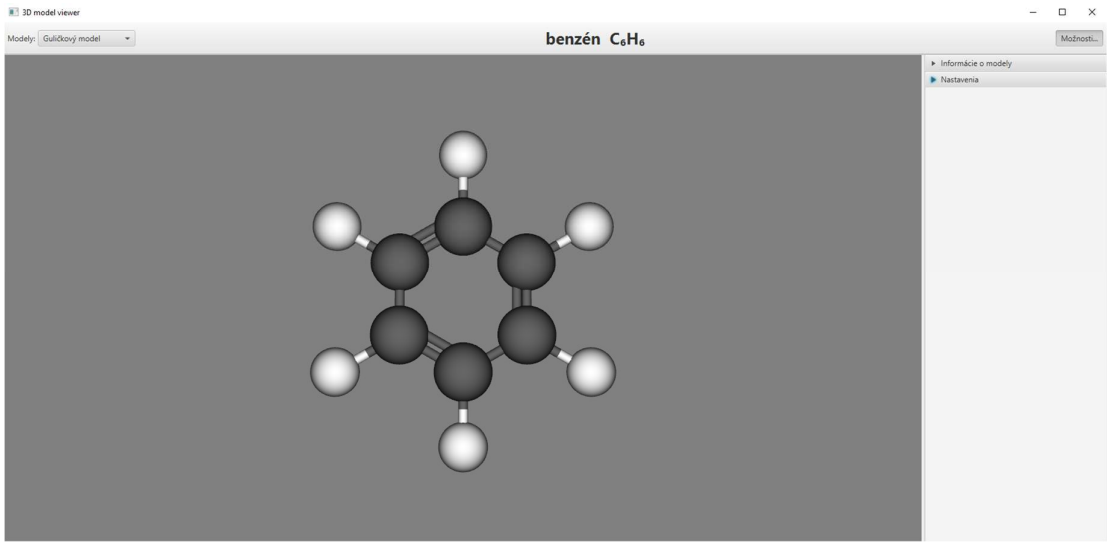
Podobne ako v kalótovom modeli, reprezentujeme jednotlivé atómy guličkami. Rozdiel je jedine v reprezentácií samotných väzieb, väzby reprezentujeme valcami (*Cylinder*), ich počet závisí od násobnosti väzby.

Modelovanie guľôčkového modelu je vďaka princípu, ktorý sme opísali v kapitole 6.1 veľmi jednoduchý, vzdialenosť spomínaných rohov kocky určíme tak aby sa neprekrývali a zostal medzi nimi priestor na znázornenie väzby. Potom stačí medzi tieto rohy umiestniť spomínaný valec/valce, predstavujúci väzbu/väzby. Farba valca je do polovice rovnaká ako atóm z ktorého vychádza a od polovice je jeho farba rovnaká ako farba atómu do ktorého vchádza.



Obrázok 29 Guľôčkový model 3-hexínu

Na obrázku č.29 môžeme vidieť neaktívne tlačidlo Násobná väzba na začiatku, rovnako ako v prípade zobrazovania vzorcov, inaktivita tlačidla spôsobí, že sa násobná väzba umiestni do stredu reťazca, čím v tomto prípade vznikol 3-hexín.



Obrázok 30 Guľôčkový model benzénu

Z obrázkov jasne vyplýva, že na znázornenie väzieb medzi atómami, je lepšie uprednostniť guľôčkový model pred modelom kalótovým.

6.4 Trubičkový model

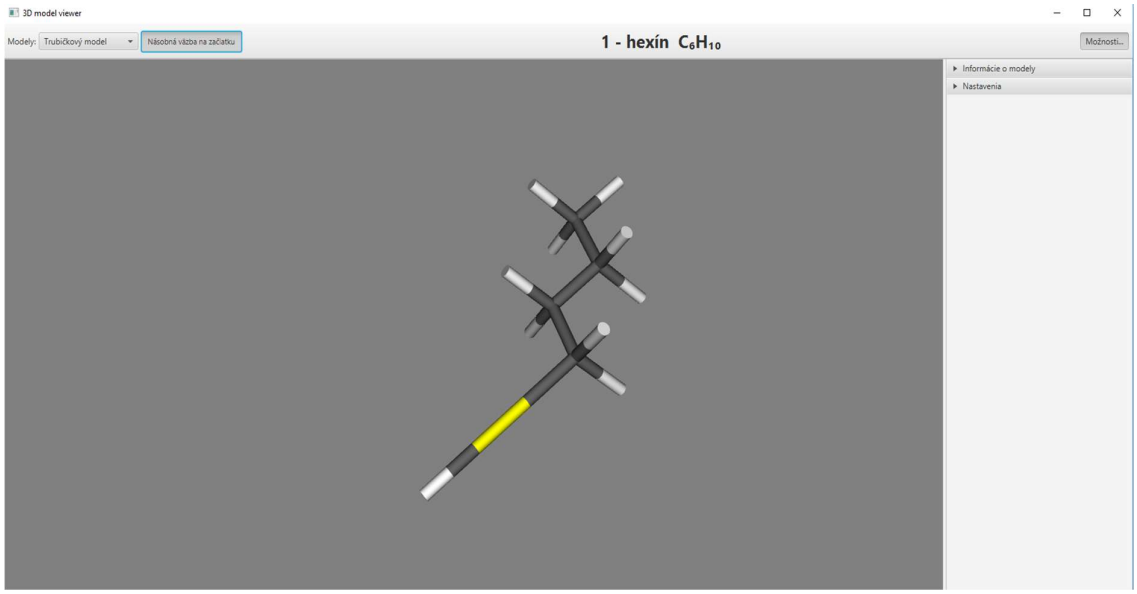
V trubičkovom modeli konkrétne atómy neznázorňujeme, znázorňujeme len väzby medzi nimi. Tieto väzby znázorňujeme podobne ako v guľôčkovom modeli s výnimkou násobných väzieb.

Násobné väzby znázorňujeme taktiež ako jednoduché, jedným valcom, rozdiel je v ich farbách. Farby v trubičkovom modeli nie sú vo všeobecnosti určené, tak sme sa zhodli na uvedených farbách:

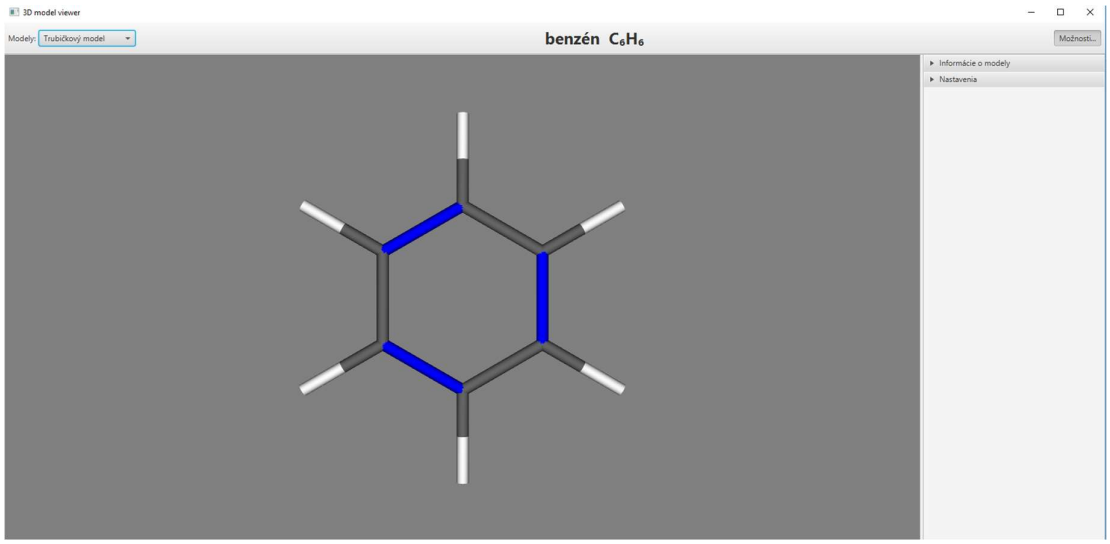
- **Dvojitá väzba** – valec má modrú farbu
- **Trojita väzba** – valec má žltú farbu

Trubičkový model je skôr určený pre komplexné zlúčeniny, v ktorých vďaka tomuto modelu jednoduchšie nájdeme určité symetrie alebo podobné časti molekuly.

Modelovanie trubičkového modelu v našej aplikácií je rovnaké ako modelovanie guľôčkového modelu s rozdielom, že v trubičkovom modeli nevykreslíme guličky reprezentujúce atómy a zmeníme reprezentáciu násobných väzieb.



Obrázok 31 Trubičkový model 1-hexínu



Obrázok 32 Trubičkový model benzénu

Záver

Cieľom práce bolo vytvorenie aplikácie na vzdelávacie účely, konkrétne mala aplikácia slúžiť na zjednotenie informácií o vybraných uhľovodíkoch.

Tento účel sa podarilo splniť v plnom rozsahu zadania. Všetky informácie pochádzajú z overených zdrojov, ktoré slúžili aj na kontrolu namodelovaných vzorcov a modelov, ktoré naša aplikácia používateľovi prezentuje.

V práci sme si najskôr vysvetlili základné vlastnosti uhľovodíkov, ďalej sme si predstavili použité softvérové platformy a v hlavnej časti sme si predstavili dátový model aplikácie a vysvetlili sme si základné princípy implementácie našej aplikácie.

Keďže zdrojové kódy samotného modelovania uhľovodíkov v našej aplikácie sú príliš rozsiahle, prezentovali sme čitateľovi len základné princípy podľa ktorých samotné modelovanie realizujeme.

Aplikácia bola vyvinutá v programovacom jazyku JAVA, čo znamená, že aplikácia nie je obmedzená len na jeden operačný systém, ale pobeží na všetkých systémoch s podporou JVM (Java Virtual Machine).

Ako sme spomínali, aplikácia v súčasnom stave prezentuje len vybraných zástupcov uhľovodíkov, z čoho vyplýva jedno z jej možných plánovaných rozšírení v budúcnosti:

- Pridanie podpory pre viacero skupín uhľovodíkov a zástupcov týchto skupín
- Rozšírenie základných informácií o uhľovodíkoch, napríklad pomocou implementácie modulu, ktorý by tieto informácie získaval z internetu.
- Implementovanie testovacieho modulu v ktorom si používateľ môže vytvárať rôzne molekulárne štruktúry

Práca bola pre nás veľkým prínosom, umožnila nám využiť doterajšie znalosti z oblasti chémie a programovania, zlepšili sme si znalosti o možnostiach grafiky v jazyku JAVA, naučili sme sa pracovať s momentálne najpoužívanejšou grafickou knižnicou JavaFX, ale hlavne sme používateľovi uľahčili pátranie po informáciách.

7 Zoznam použitej literatúry

JAVOROVÁ, K., LISÁ, V.: ***Chémia - pracovný zošit pre 9. ročník ZŠ a 4. ročník gymnázií s osemročným štúdiom***. Raabe, Bratislava, 2012, ISBN 978-80-8140-039-1

J.KMEŤOVÁ - M.SKORŠEPA - P.MÄČKO, ***Chémia pre 2.ročník gymnázia so štvorročným štúdiom a 6. ročník gymnázia s osemročným štúdiom***,Bratislava, 2012, ISBN 978-80-8091-271-0

J.KOVÁČ – Š.KOVÁČ – Ľ.FIŠERA **–** ***Organická chémia 1.diel*** ,Bratislava, 1995 , ISBN 80-227-0791-0

R.H. PETRUCCI - W.S. HARWOOD - F.G. HERRING, ***General Chemistry*** (8th ed., Prentice-Hall 2002), ISBN 0-13-014329-4

J.VOS – W.GAO – S.CHIN – D.IVERSON – J.WEAVER ***Pro JavaFX 8, A Definitive Guide to Building Desktop, Mobile, and Embedded Java Clients***, New York, 2014, ISBN-13 (electronic): 978-1-4302-6575-7 , ISBN-13 (pbk): 978-1-4302-6574-0

[PubChem] ***The PubChem Projetct***, WWW.PUBCHEM.NCBI.NLM.NIH.GOV

Planetá vedomostí WWW.PLANETAVEDOMOSTI.IEDU.SK/

Zoznam príloh

Príloha A DVD

Príloha A: Obsah DVD

Priložené DVD obsahuje:

Práca v elektronickej podobe (formát PDF)

Zdrojové kódy aplikácie

Spustiteľná aplikácie vo formáte JAR