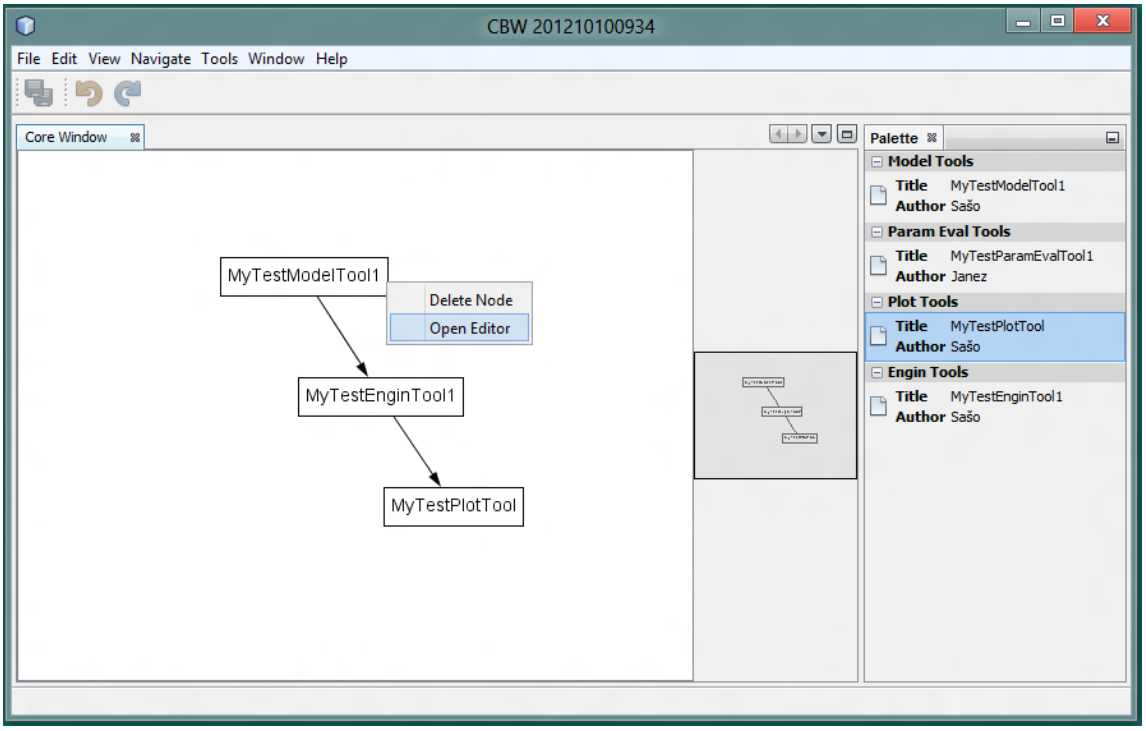
COMPUTATIONAL BIOLOGY WORKSPACE (CBW)
(hitri vodič okolja in kreiranja novega modula)

Okolje

Aplikacija za svoje ogrodje uporablja *NetBeans Platform*. Za najboljšo kompatibilnost razvijanja je potrebno, da imate nameščeno različico **NetBeans IDE 7.2.1**. Aplikacija je spisana za delovanje na Microsoft Windows operacijskem sistemu in zna imeti težave na drugih platformah (zaradi JavaFX knjižnjic, ki se nahajajo v ./release/modules/bin).

Delovanje aplikacije

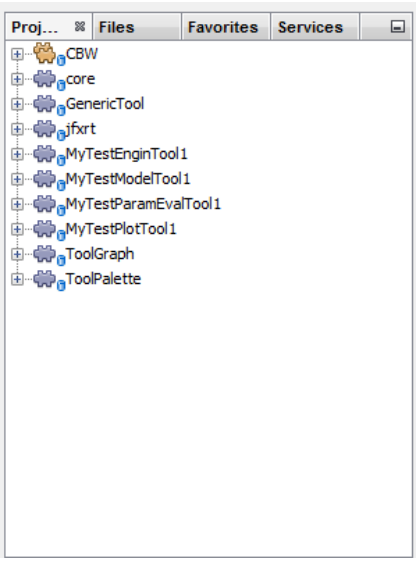
Osnovna dva dela aplikacije sta *Palette*, ki vsebuje razpoložljive gradnike – orodja (*tools*) in *Core window* oziroma delovna površina, v katero razpoložljive gradnike razmeščamo in med seboj povezujemo. Orodja v delovni površini obravnavamo kot vozlišča (*nodes*), ki so medsebojno povezana z usmerjenimi povezavami, pri čemer usmerjenost povezav nakazuje tok podatkov. Orodja lahko med seboj povežemo, tako da držimo tipko *Ctrl* in jih z miško poklikamo.



Čeprav aplikacija dopušča povezovanje vseh orodji med seboj je potrebno biti pazljiv, saj so orodja odvisna od predhonikov (predhodnik plot orodja mora na primer biti engine). Z desnim klikom na vozlišče se nam prikaže izpustni meni, ki nam ponudi brisanje ali dodatno urejanje orodja.

Primer pravilnega povezovanja je viden na zgornji sliki. Orodje *PlotTool* dobi podatke iz orodja *EngineTool*, ta pa iz orodja *ModelTool*.

Struktura aplikacije



Aplikacija je razdeljena na glavne module in razširitve. Glavni moduli so:

- ***Core***: glavno okno, ki se odpre ob zagonu aplikacije in je ogrodje za *ToolGraph* in *ToolPalette* module.
- ***ToolGraph***: je implementacija *GraphScene* razreda in omogoča dodajanje vozlišč in povezovanje le teh. Tako imenovana *delovna površina*.
- ***ToolPalette***: je posebno okno v *NetBeans* platformi in se veže na obstoječe okno. V tem primeru se veže na delovno površino in vsebuje vse module, ki so najavljeni *NetBeans Lookup* servisu in so otroci razreda *GenericTool*.
- ***GenericTool***: je paket za vse razširitvene module. Glavni razred je *AbstractGenericTool*, katerega dedujejo vsi ostali abstraktni razredi: *AbstractEngineTool*, *AbstractPlotTool*, *AbstractModelTool* in *AbstractParamEvalTool* (ki deduje *AbstractModelTool*) . Modul ima še razred *ToolTopComponent*, katerega mora uporabiti vsak razširitveni modul, ki bo imel grafični vmesnik ter razred *AbstractReactionType*, ki je namenjen prenašanju informacij reakcije med moduli.
- ***jfxt*:** knjižnica *JavaFX*, za risanje grafov

Ostali moduli so razširitveni demo moduli, ki nakazujejo delovanje in uporabo.

Generična orodja

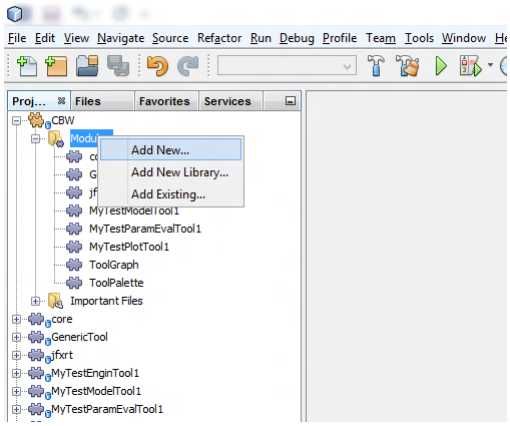
Vsa generična orodja so abstraktne implementacije abstraktnega razreda *AbstractReactionType*. Vsak razred oziroma orodje ima svojo funkcijo. Za bolje razumevanje si bomo tukaj bolj podrobno pogledali 4 osnovne tipe orodji (mogoče je dodati tudi svojega).

- ***AbstractModelTool***: primarno orodje kjer definiramo reaktante, produkte, koeficiente… Zato mora model biti prvo orodje v mreži, ki jo kreiramo na delovni površini. Model ima dve spremenljivki, *species* in *reactions*, ter njihove *set* in *get* metode. Obe spremenljivki sta tipa List, le da je spremenljika *species*, seznam tipa String, v katerem hranimo seznam vseh kemijskih zvrsti, spremenljivka *reactions* pa je seznam naše implementacije abstraktnega razreda *AbstractReactionType*. Vsak element v spremenljivki *reactions*, predstavlja reakcijo katero moramo sami definirati (reaktanti, produkti,…). Tako imajo nasledniki orodja (povezani na delovni površini) dostop do podatkov celotnega modela preko *get* metod.
- ***AbstractParamEvalTool***: orodje deduje *AbstractModelTool* ampak ni namenjeno definiranju reakcij, temveč nam omogoča, da že definiran model spremenimo ne da bi vplivali na izvirni model. Tako lahko simuliramo več nastavitev hkrati. Ker deduje *AbstractModelTool* ima tudi ta razred že definirane metode, katere pa mora prepisati, saj ne želimo vrača izvirne podatke temveč spremenjene. Paziti moramo tudi da naredimo svojo instanco podatkov, ter tako ne prepišemo izvornih.
- ***AbstractEngineTool***: ko že imamo definiran model, z *Engien* orodjem definiramo začetno stanje v sistemu ter poženemo simulacijo. Ker želimo podatke vizualizirati mora orodje implementirati metodo *getLineChartData*, ki vrača spremenljivko tipa *ObservableList<XYChart.Series<Double,Double>>*. Ker se spremenljivka neposredno prenese v inicializacijo grafa lahko tako vplivamo kako se bodo podatki izrisali. Orodje ima tudi metodo *calculate*, s katero naj bi sprožili izračun.
- ***AbstractPlotTool***: orodje namenjeno vizualizaciji rezultatov simulacije. Predhodnik orodja na delovni površini mora biti tipa *AbstractEngineTool*, katerega podatke prikaže. Za prikaz naj bi

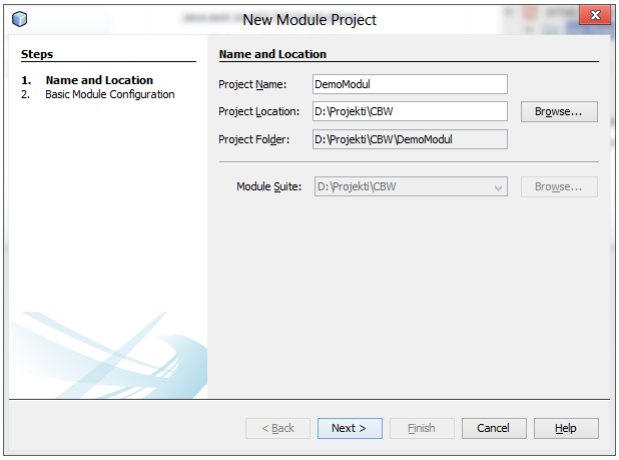
se uporabljala knjižnica JavaFX, ki podpira prikaz grafov različnega tipa in dinamično manipuliranje.

Pisanje novega modula

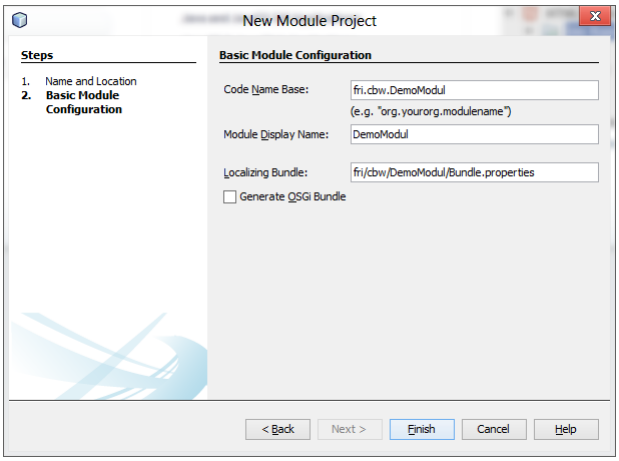
1. Začnemo tako, da s čarovnikom ustvarimo nov *NetBeans module*.



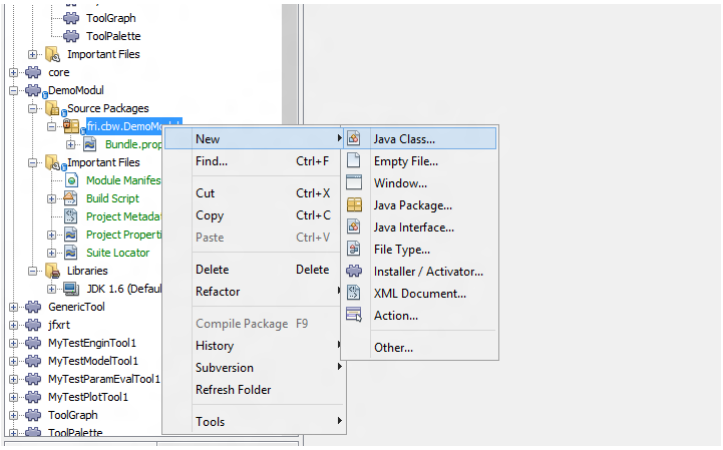
2. Izberemo ime modula in se prepričamo, da je lokacija projekta znotraj glavnega projekta,



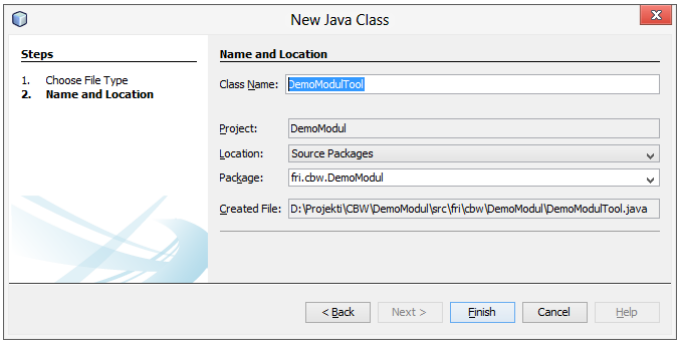
3. Izberemo ime paketa in tako zaključimo s čarovnikom.



4. Ko imamo nov modul, lahko začnemo s programiranjem našega orodja. Najprej dodamo nov razred, ki bo dedoval enega izmed abstraktnih razredov. S tem se bo prikazoval tudi na *ToolPalette* in ga bo moč prenesti na *ToolGraph*.



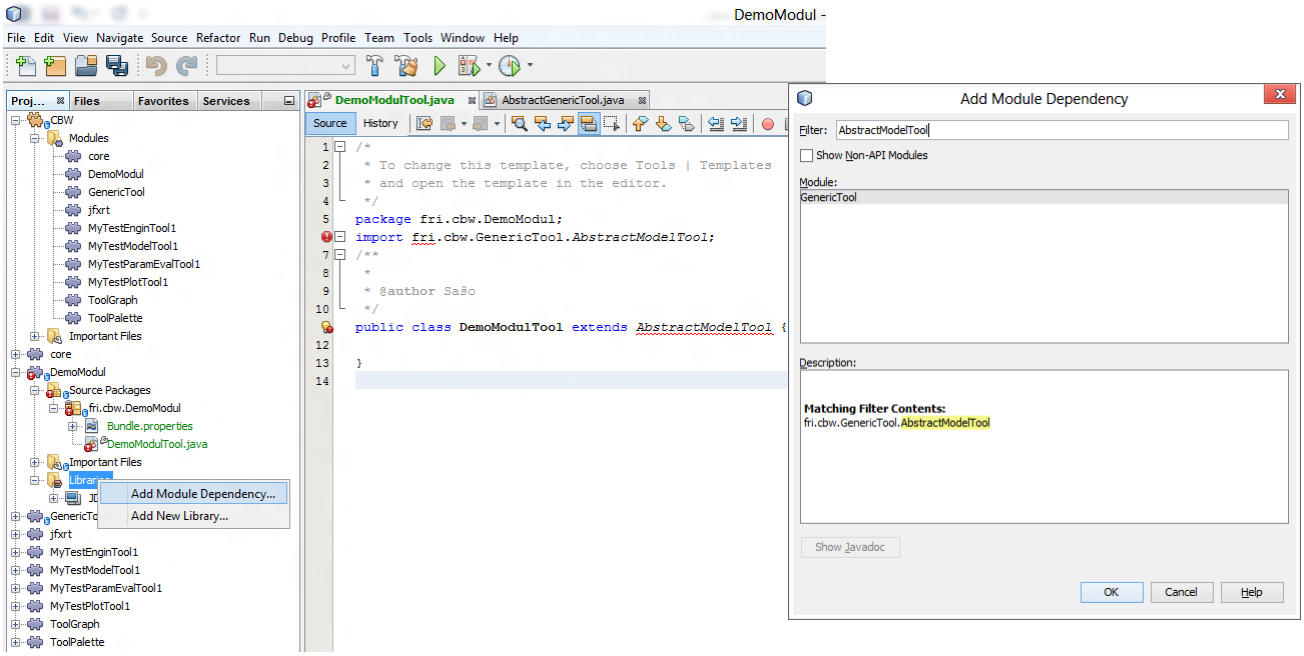
V našem primeru bomo implementirali orodje za modeliranje,



zato bo razred dedoval `AbstractModelTool`.

```
package fri.cbw.DemoModul;
import fri.cbw.GenericTool.AbstractModelTool;

public class DemoModulTool extends AbstractModelTool {

}
```

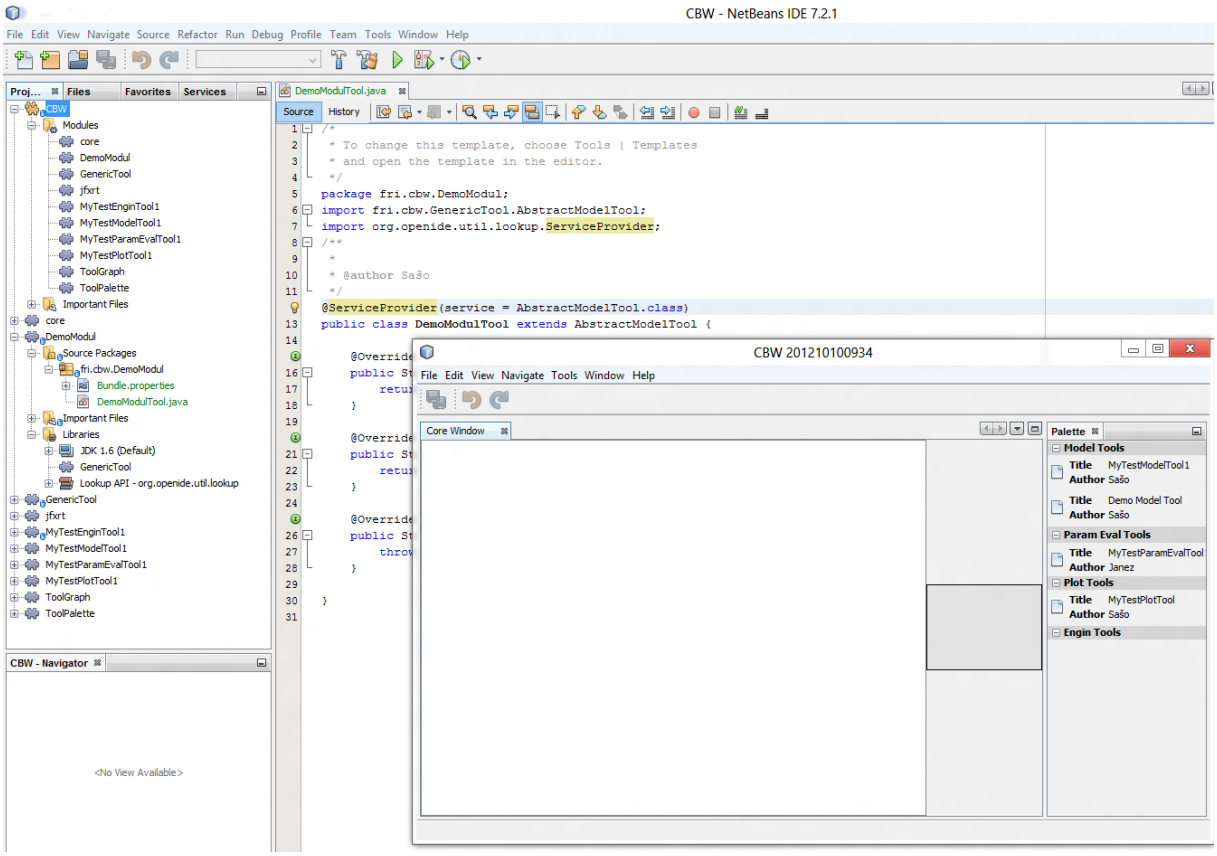
Tu že vidimo prve probleme modularnega razvijanja, saj se moduli med seboj ne vidijo dokler ne določimo odvisnosti med njimi. V *NetBeans*-ih to preprosto naredimo preko iskalnika *Add Module Dependency*…



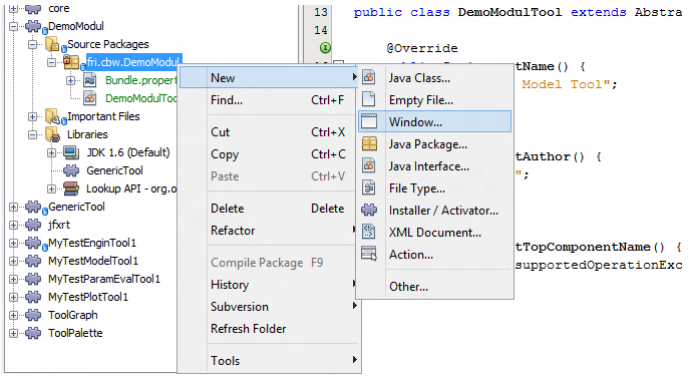
5. Ko implementiramo še abstraktne metode nam ostane samo še da ga najavimo *NetBeans service lookup*-u. Modul *ToolPalette* namreč naredi seznam s pomočjo *Lookup API*, ki pa preišče vse module ki imajo najavljene servise določenega razreda.

```
package fri.cbw.DemoModul;
import fri.cbw.GenericTool.AbstractModelTool;

@ServiceProvider(service = AbstractModelTool.class)
public class DemoModulTool extends AbstractModelTool {

    @Override
    public String getName() {
        return "Demo Model Tool";
    }

    @Override
    public String getAuthor() {
        return "Sašo";
    }

    @Override
    public Class getTopComponentClass() {
          //Metodo bomo implementirali šele ko bomo naredili GUI za modul
        throw new UnsupportedOperationException("Not supported yet.");
    }

}
```

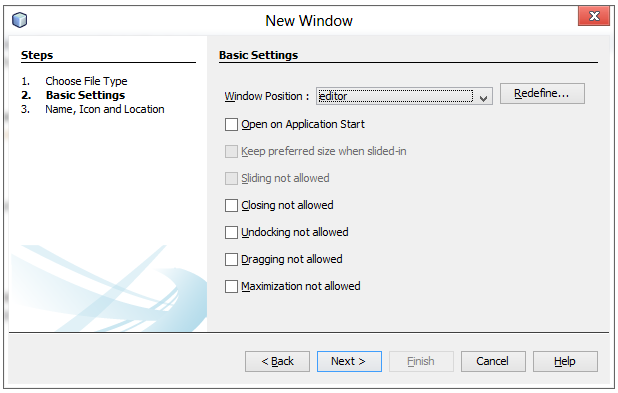
6. Registracijo naredimo na 2 načina. Lahko naredimo datoteko z imenom servisa v paktu *META-INF.services* (kot je to narejeno v *MyTestModelTool1*) ali pa preko notacij (kot je to narejeno v *MyTestEngineTool1*). V tem primeru ga bomo registrirali preko notacije in to z `@ServiceProvider(service = AbstractModelTool.class)`, ki pa bo javljala napako dokler modulu ne dodamo odvisnosti za *Lookup API*. Če zdaj lahko zaženemo aplikacijo, je naš *DemoModul* že na paleti.



7. Orodje potrebuje še grafični urejevalnik. *NetBeans* nam tu zopet olajša zadevo s čarovnikom *New Window*.



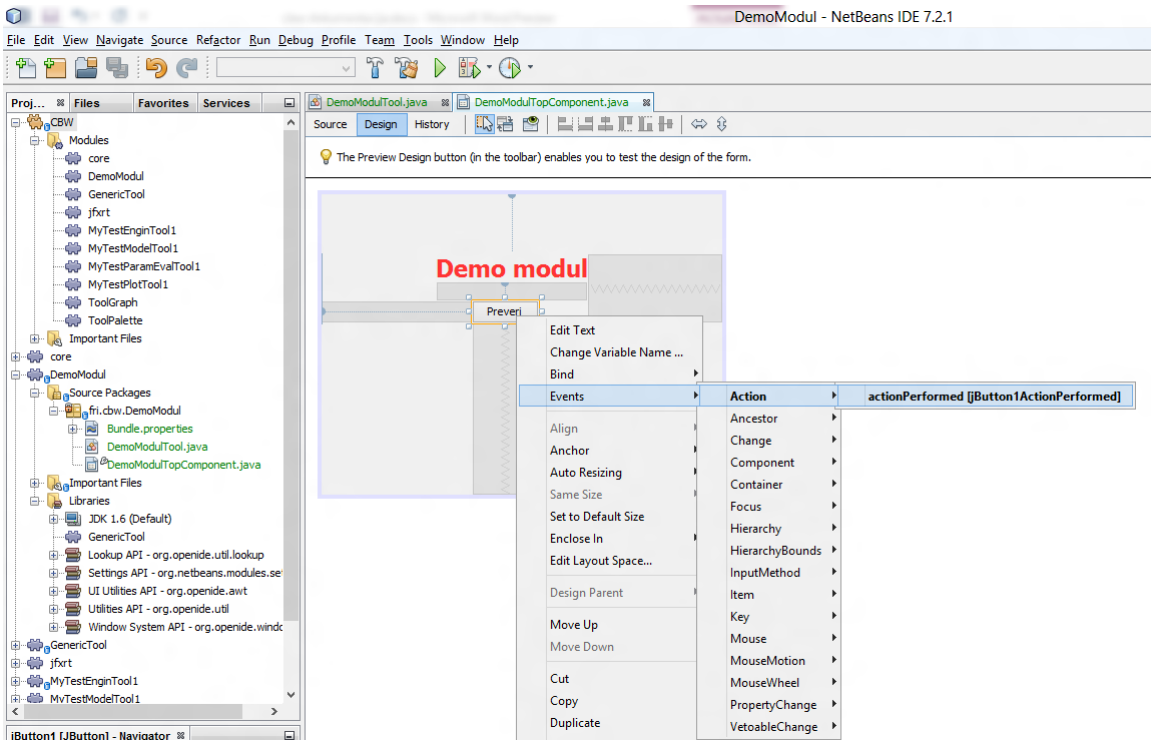
8. V našem primeru želimo, da se okno odpre na poziciji *Editor* (osrednji del, o razporeditvi si lahko preberete več na http://ui.netbeans.org/docs/ui/ws/ws_spec-netbeans_ide.html). Poleg tega ne želimo da se okno odpre že ob zagonu applikacije (odpreti ga želimo šele ob sprožitvi urejanja orodja). Na naslednjem koraku dodamo še predpono in s čarovnikom smo zaključili. *NetBeans* nam zdaj odpre *Swing* urejevalnik preko katerega lahko z *drag and drop* komponentami ustvarimo naš vmesnik. Opazimo lahko, da nam je *NetBeans* čarovnik dodal tudi vse potrebne odvisnosti do knjižnjižnjic in se nam s tem ni potrebno s tem ubadati (npr. *Window System API*).



9. Preden se lotimo izdelave vmesnika je dobro, da ga najprej naredimo kompatibilnega s *CBW* okoljem. Naš `DemoModulTopComponent` razred trenutno deduje `TopComponent`. To je potrebo spremeniti v `ToolTopComponent`, saj nam ta omogoča povezavo do delovne površine in tako povezav med tem in ostalimi moduli. Spremenjeno dedovanje javi napako konstruktorja, saj ne obstaja *super* konstr brez parametrov. Zato našemu konstruktorju `DemoModulTopComponent()` dodamo parametre `GraphScene scene` in `IconNodeWidget toolNode` in v njem kličemo metodo `super(scene, toolNode)`.

```
public DemoModulTopComponent(GraphScene scene, IconNodeWidget toolNode) {
    super(scene, toolNode);
    initComponents();
    setName(Bundle.CTL_DemoModulTopComponent());
    setToolTipText(Bundle.HINT_DemoModulTopComponent());
}
```

Spremenimo še `TopComponent` anotacijo `TopComponent.PERSISTENCE_ALWAYS` v `TopComponent.PERSISTENCE_NEVER`, da se okno ponastavi ob izhodu aplikacije in zbrišemo anotacije `@ActionReference`, @TopComponent.OpenActionRegistration, @ConvertAsProperties ki doda akcijo v meni *Window*. S tem onemogočimo prikaz urejevalnika ne da bi imeli povezavo na delovno površino, kar bi povrzočalo napake pri delovanju aplikacije.
Popravimo še naš *DemoModulTool* razred in implementiramo še metodo `getTopComponentClass()`. Dodamo ji samo *return* izjavo z vrednostjo `DemoModulTopComponent.class`. Zdaj lahko zaženemo aplikacijo. Dodamo naše orodje na delovno površino in z desnim klikom odpremo urejevalnik, ki je trenutno še prazen.

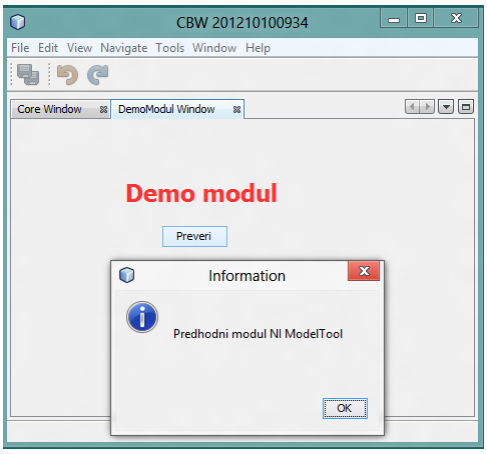


10. Za naš Demo primer sem na urejevalni površini dodal labelo in gumb, ki bo preveril ali je predhodno orodje tudi tipa `AbstractModelTool`.

    Koda za gumb je sledeča:
    (ne pozabite dodati odvisnosti: *Visual Library API, ToolGraph, Dialogs API, Utilities API*)

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        NotifyDescriptor nd;

        try{
            ToolWrapper tool = (ToolWrapper)getScene().findObject (getToolNode());
            ToolWrapper prevTool = tool.getPrevNode(getScene());
            AbstractModelTool gt = (AbstractModelTool) prevTool.getNodeGenericTool();

            nd = new NotifyDescriptor.Message("Predhodni modul JE ModelTool");

        }catch(ClassCastException e){
            Logger.getAnonymousLogger().severe(e.getMessage());
            nd = new NotifyDescriptor.Message("Predhodni modul NI ModelTool");
        }catch(NullPointerException e){
            Logger.getAnonymousLogger().severe(e.getMessage());
            nd = new NotifyDescriptor.Message("Predhodni modul ne obstaja");
        }finally {
            DialogDisplayer.getDefault().notify(nd);
        }
    }
```

    Funkciji `getScene()` in `getToolNode()` se morata nahajati v *ToolTopComponent* razredu in sta referenci na delovno površino in vozlišče orodja, ki ga trenutno urejamo. S tem lahko dobimo *ToolWrapper*, na delovni površini, ki ima metodo `getPrevNode()`. Ta vrne prvo predhodno orodje (ali `null`, če predhodno orodje ne bostaja). V *ToolWrapper*-ju pa se nahaja tudi samo orodje ki ga dobimo z metodo `getNodeGenericTool`.



11. Aplikacija omogoča shranjevanje in nalaganje delovne površine z vsemi nastavitvami. Akcije se izvedejo v modulu *ToolGraph* in je kombinacija XML strukture in serializacije objektov. Shrani se samo delovna površina (vozlišča in povezave) ter orodja vezana na njo (objekti ki dedujejo *AbstractGenericTool*). Grafični urejevalnik posameznega orodja se ne shrani, zato moramo poskrbeti da se vsebina na maski prenese na objekt. To lahko naredimo na dva načina. Lahko poslušamo na spremembe na maski ter podatke istočasno zapisujemo na objekt lahko pa prepišemo (`@Override`) metodo `doSave()`, ki se nahaja v razredu *ToolTopComponent*.
    Za demonstracijo bomo na našo masko dodali še komponento *JTextField*, katere vrednost se bo shranila v naš *DemoModulTool*. V *DemoModulTool* dodamo novo spremenljivko `savedText` tipa `String` ter kreiramo `set` in `get` metodi. *JTextField* dodamo *focusLost event* kjer dodamo kodo za shranjevanje teksta.

```
private void jTextField1FocusLost(java.awt.event.FocusEvent evt) {
        ((DemoModulTool)getGenericTool()).setSavedText(jTextField1.getText());
    }
```

    Metoda `getGenericTool()` vrne naš *DemoModulTool,* od koder lahko dostopamo do `saveText` spremenljivke. Tekst bo tako ob shranjevanju serializiran, še vedno pa se ne prikaže če naložimo shranjeno delo. Pri inicializaciji *DemoModulTopComponent* moramo ovrednotiti vsa polja ki smo jih shranili. V našem primeru bi na koncu konstruktorja dodali še:

```
setToolTipText(Bundle.HINT_DemoModulTopComponent());
```

    Če imamo v našem orodju spremenljivke, ki ne želimo da se shranijo (npr. izračun podatkov) jim moramo ob deklaraciji predpisati tip *transient* (npr. `private transient String s;`).