**NetBeans Platform Crash Course**

By NetBeans User Group Serbia [www.netbeans-serbia.org](http://www.netbeans-serbia.org/)

In this tutorial we will expand concepts from previous tutorial using new IDE - NetBeans 7.0 Beta 2.

**Overview**

1. Creating NetBeans application for keeping track of music discs – CDCatalogApplication
   1. Creating Netbeans Platform application
   2. Creating a module and adding it to the Platform
   3. Adding new window to the module
   4. Adding new modules
   5. Connecting modules
   6. Adding new action
2. Adding wizard to existing module
   1. Creating singleton class
   2. Creating wizard
3. Adding validator
   1. What is validator
   2. Adding validator
   3. Defining validator through interface
4. Advanced way of showing added CDs
   1. Purpose of advanced view
   2. Purpose of used classes

Requirements:

Java 1.6 or later

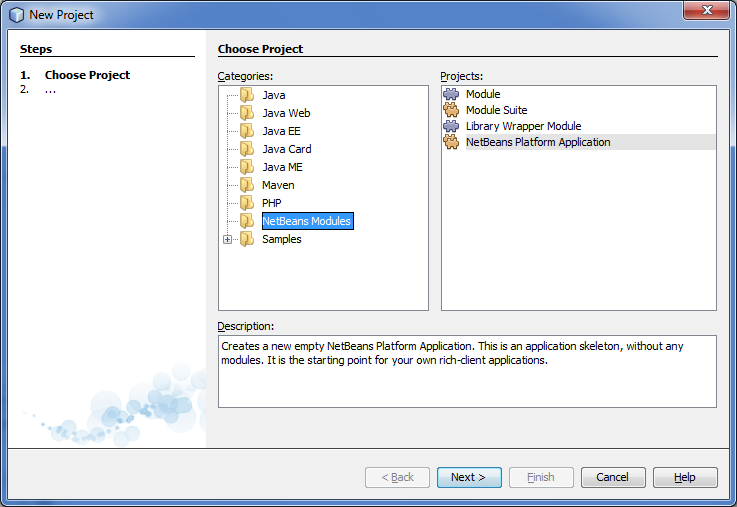
Netbeans 7.0 Beta 2 or later

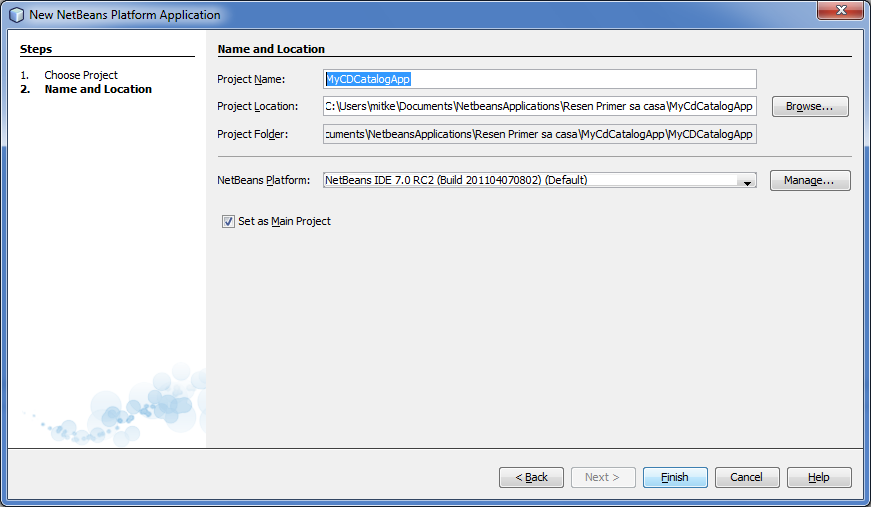
Prerequisites:

Basic Java and Swing programming

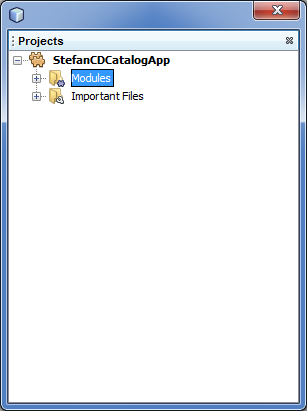
1. Creating NetBeans application for keeping track of music discs – CDCatalogApplication

In main menu go to ***New Project***  and choose ***Netbans Modules*** > ***Netbeans Platform Application*** , as shown on image below:



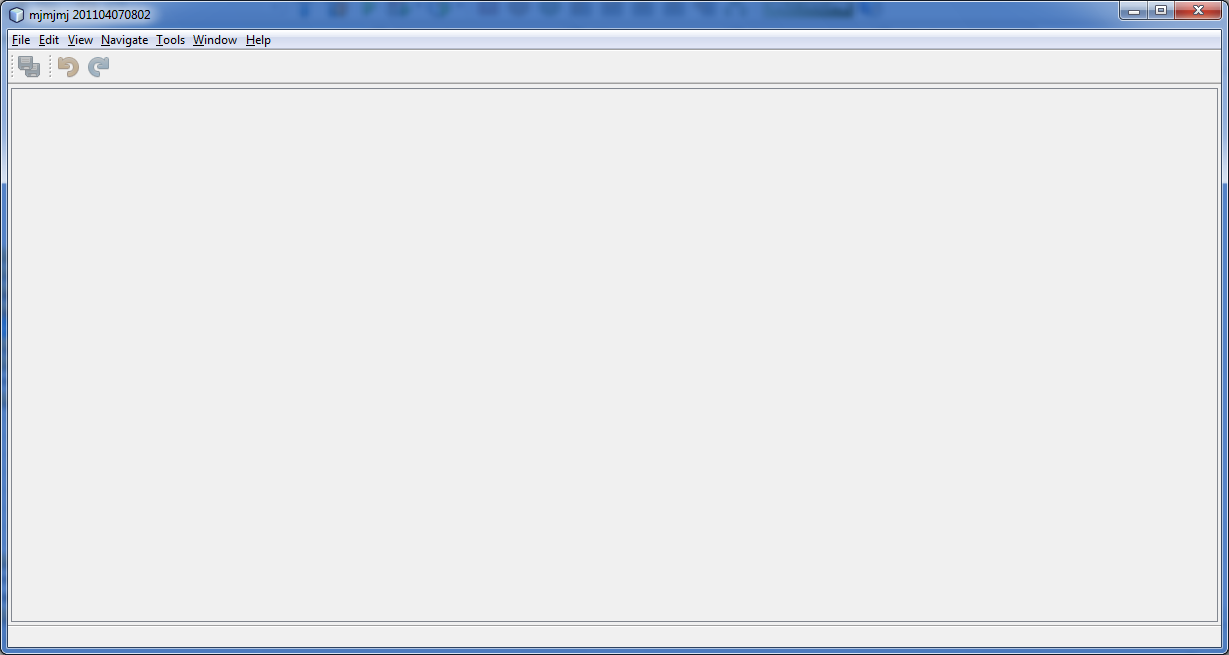
After that, set project name and project location:

Now, your project looks like this:

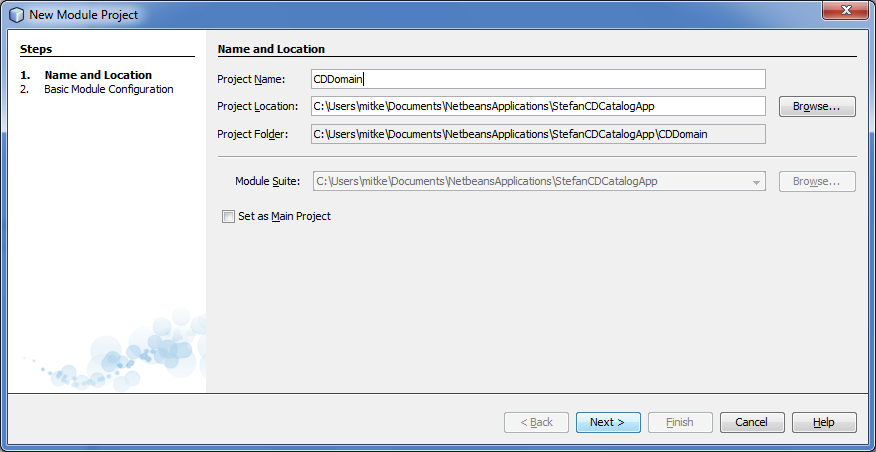


Running the application you will see window like this:

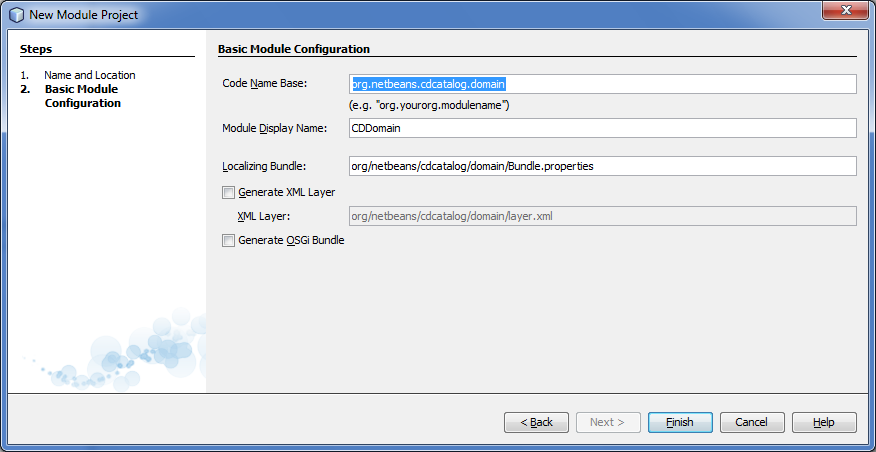




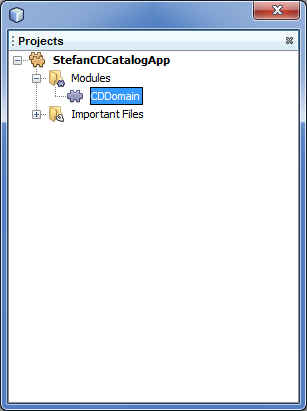
Now we need to add new modules to the project. Right click on ***Modules***  folder of project and choose option ***Add New*** .



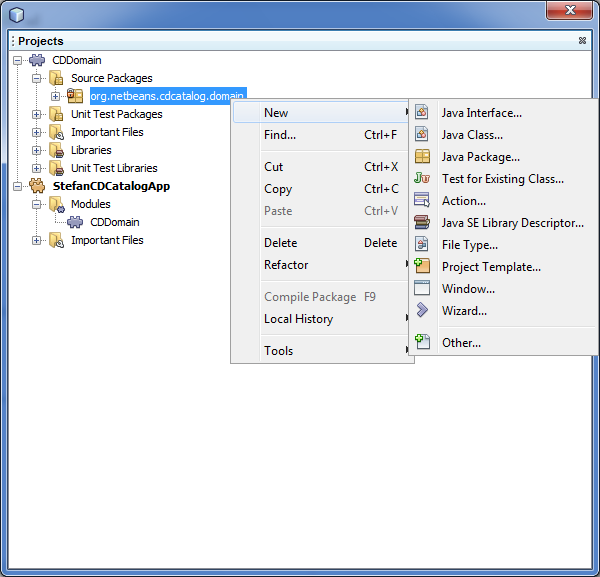
Set name for this module to **CDDomain**.it will contain domain class of project. After that you set the package names in this module. After successful adding click ***Finish***.



On next image you can see project structure:



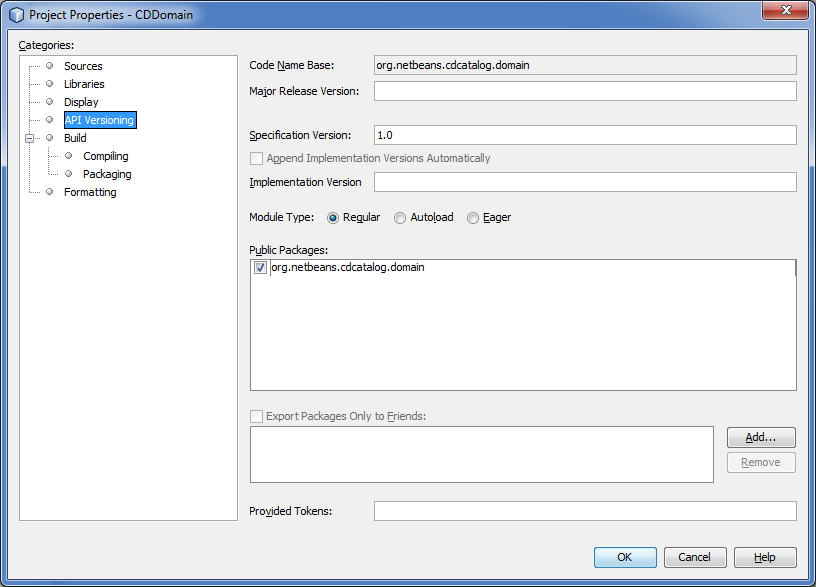
Considering that the module is empty, we need to add classes to it. This module is dedicated to domain classes, so we will add class CD. Double left click on our module in ***Modules*** folder will open our folder, which will give us access to its packages and classes:



Below is code which represents content of CD class:

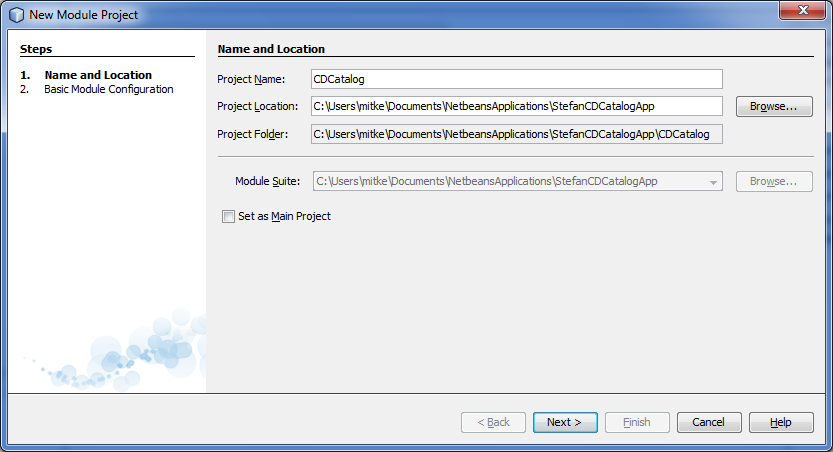
|  |
| --- |
| 1. package org.netbeans.cdcatalog.domain; 2. public class CD { 3. private String title; 4. private String artist; 5. private String songs; 6. public CD(String title, String artist) { 7. this.title = title; 8. this.artist = artist; 9. } 10. public CD(String title, String artist, String songs) { 11. this.title = title; 12. this.artist = artist; 13. this.songs = songs; 14. } 15. public String getArtist() { 16. return artist; 17. } 18. public void setArtist(String artist) { 19. this.artist = artist; 20. } 21. public String getSongs() { 22. return songs; 23. } 24. public void setSongs(String songs) { 25. this.songs = songs; 26. } 27. public String getTitle() { 28. return title; 29. } 30. public void setTitle(String title) { 31. this.title = title; 32. } 33. @Override 34. public String toString() { 35. return "Artist: "+this.artist+ " "+ "Title: "+this.title; 36. } 37. } |

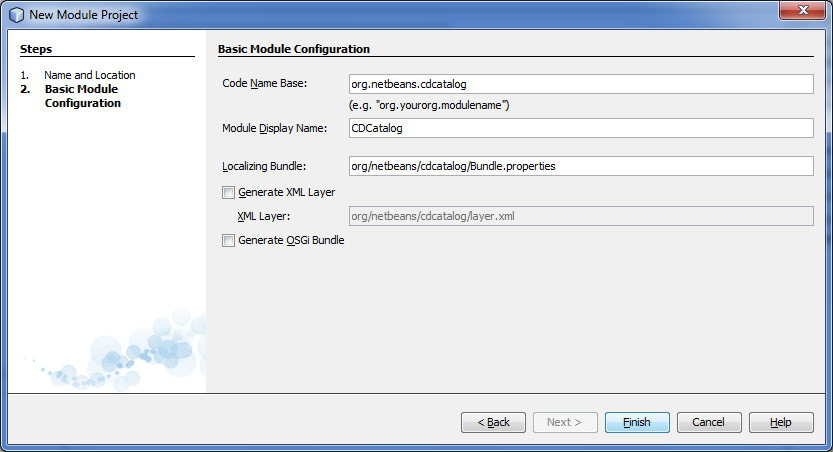
If you take a closer look at project architecture, you will notice small padlock next to package icon in module. That means that content of that package is unavailable to other modules in our application. To make classes in that package public, you need to right click the package and choose option ***Propereties > API Versioning***:



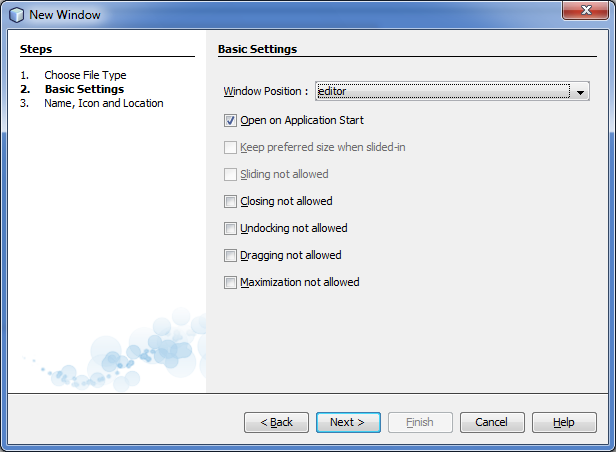
In the field ***Public Packages***  check which packages you wish to make public for other modules in your application.

It’s time to add new module to the application, which will represent graphical interface of our cd catalog.

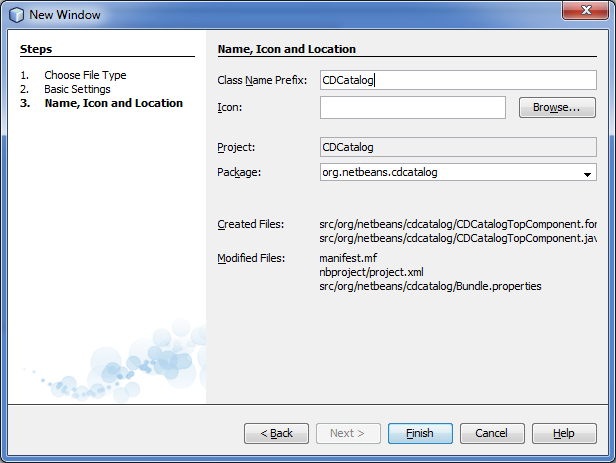




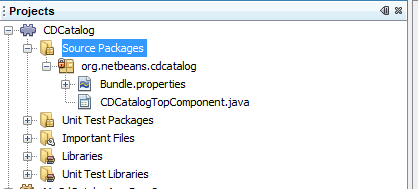
Netbeans platform gives complete system for window management, with some useful predefined options, so we will describe procedure of creating such windows. In **CDCatalog** module right click to package of classes and choose ***New> Window***  and you will see menu like this:



In this menu you can set position of window in main screen of NetBeans application, and you can set when will the window appear. Clicking on ***Next*** will give next window, where you set the name of class that represents the window, and possibility to add icon. You can see files that will IDE create for you.



After these settings, our module looks like this:

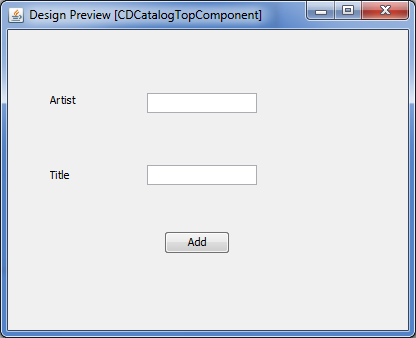


CDCatalogTopComponent looks like regular SWING class, on which we cann add standard SWING components. When you open class in ***Source*** ***View*** you can see annotations which allows it integration in Netbeans windows module, as well as methods ***componentOpened() componentClosed() writeProperties(java.util.Properties p) readProperties(java.util.Properties p)*** .

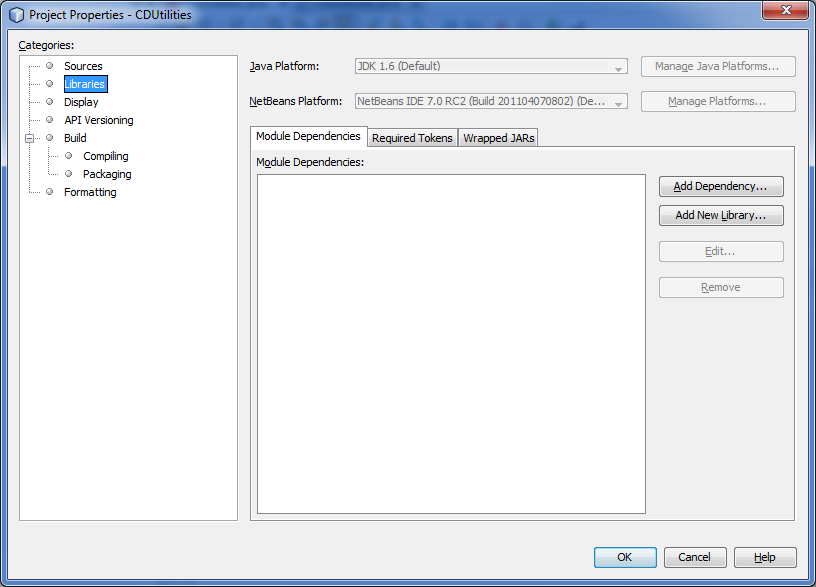
For more information about TopComponent class visit:

<http://bits.netbeans.org/dev/javadoc/org-openide-windows/org/openide/windows/TopComponent.html>

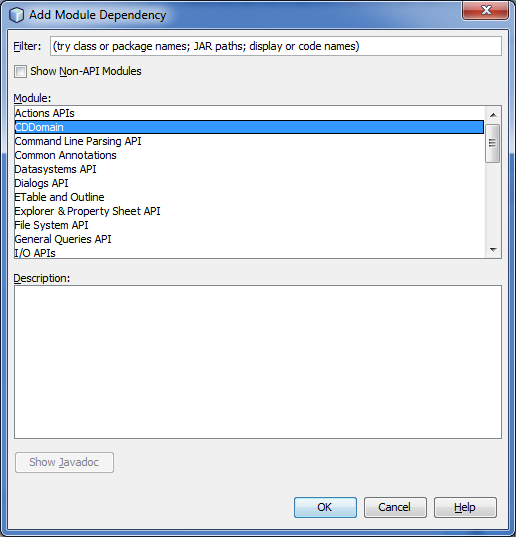
In this step we will set some SWING elements in our window, to make it more functional. Purpose of this window is to enter new albums with songs (new discs), and for that we need two text fields, two labels and one button. Window should look like this:



Double click on ***Add***  button we can set handler that will define what is happening when button is clicked. We will set that click on button adds album to our discs database. Our database will be represented by list, concrete class list will be put in different module which will be called ***CDUtilities,*** and its package ***org.netbeans.cdcatalog.cdutilities***. To be able to use class ***CD***, which is in different module and made public to other modules(procedure for this is explained on page 7), we have to set dependency for module that contains class ***CD***, and that is module ***CDDomain***. Right click on module ***CDUtilities***, and choose option ***Properties>Libraries.*** You will see window like this:



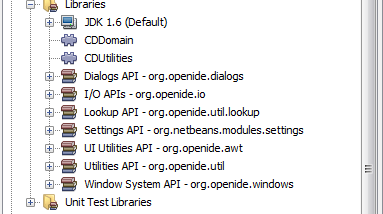
Click on ***Add Dependency,***  and choose ***CDDomain***



Now we can use class ***CD***  *in* ***CDUtilities,***  it’s enough to import class. Code required for **CDUtilities** class is below:

|  |
| --- |
| package org.netbeans.cdcatalog.cdutilities;  import java.util.ArrayList;  import org.netbeans.cdcatalog.domain.CD;  public class CDUtilities {  private static CDUtilities instance;  private ArrayList<CD> database = new ArrayList<CD>();  private CDUtilities() {  }  public static CDUtilities getInstance() {  if (instance != null) {  instance = new CDUtilities();  }  return instance;  }  public ArrayList<CD> getCDs() {  return database;  }  public void addCDs(CD newCD) {  database.add(newCD);  }  } |

Tip–> If you wish to check which dependencies you added to your module it’s enough to look in ***Libraries*** folder of your module:



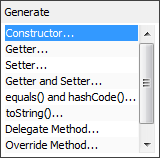
In definition of ***CDUtilities*** class we used one of the most used design patterns: Singlton pattern, which guarantees existence of only one instance of ***CDUtilities,*** for one user(one user – one connection to database). Singlton is relatively simple pattern: it’s based on existence of private constructor and public static method which calls the constructor if no instance of the given class exists already.

Now we need to repeat the procudure for module ***CDCatalaog***, by adding dependencies to ***CDUtilities***, ***CDDomain*** and ***I/O API*** modules.

After this we need to add code in ***CDCatalogTopComponent*** class (code that needs to be added is marked in red):

|  |
| --- |
| package org.netbeans.cdcatalog;  import java.util.ArrayList;  import java.util.List;  import java.util.logging.Logger;  import org.openide.util.NbBundle;  import org.openide.windows.TopComponent;  import org.openide.windows.WindowManager;  import org.netbeans.api.settings.ConvertAsProperties;  import org.netbeans.cdcatalog.domain.CD;  import org.openide.awt.ActionID;  import org.openide.awt.ActionReference;  /\*\*  \* Top component which displays something.  \*/  @ConvertAsProperties(dtd = "-//org.netbeans.cdcatalog//CDCatalog//EN",  autostore = false)  @TopComponent.Description(preferredID = "CDCatalogTopComponent",  //iconBase="SET/PATH/TO/ICON/HERE",  persistenceType = TopComponent.PERSISTENCE\_ALWAYS)  @TopComponent.Registration(mode = "editor", openAtStartup = true)  @ActionID(category = "Window", id = "org.netbeans.cdcatalog.CDCatalogTopComponent")  @ActionReference(path = "Menu/Window" /\*, position = 333 \*/)  @TopComponent.OpenActionRegistration(displayName = "#CTL\_CDCatalogAction",  preferredID = "CDCatalogTopComponent")  public final class CDCatalogTopComponent extends TopComponent {  private static CDCatalogTopComponent instance;  private List<CD> cdDatabase = new ArrayList<CD>();  private static final String PREFERRED\_ID = "CDCatalogTopComponent";  public static synchronized CDCatalogTopComponent getDefault() {  if (instance != null) {  return instance;  }  return instance = new CDCatalogTopComponent();  }  public CDCatalogTopComponent() {  initComponents();  setName(NbBundle.getMessage(CDCatalogTopComponent.class, "CTL\_CDCatalogTopComponent"));  setToolTipText(NbBundle.getMessage(CDCatalogTopComponent.class, "HINT\_CDCatalogTopComponent"));  }  <GeneratedCode>    public static synchronized CDCatalogTopComponent findInstance(){  TopComponent wind = WindowManager.getDefault().findTopComponent(PREFERRED\_ID);  if (wind == null) {  Logger.getLogger(CDCatalogTopComponent.class.getName()).warning(  "Cannot find " + PREFERRED\_ID + " component. It will not be located properly in the window system.");  return getDefault();  }  if (wind instanceof CDCatalogTopComponent) {  return (CDCatalogTopComponent) wind;  }  Logger.getLogger(CDCatalogTopComponent.class.getName()).warning(  "There seem to be multiple components with the '" + PREFERRED\_ID  + "' ID. That is a potential source of errors and unexpected behavior.");  return getDefault();  }  @Override  public int getPersistenceType() {  return TopComponent.PERSISTENCE\_ALWAYS;  }        @Override  public void componentOpened() {  // TODO add custom code on component opening  }  @Override  public void componentClosed() {  // TODO add custom code on component closing  }  void writeProperties(java.util.Properties p) {  // better to version settings since initial version as advocated at  // http://wiki.apidesign.org/wiki/PropertyFiles  p.setProperty("version", "1.0");  // TODO store your settings  }  Object readProperties(java.util.Properties p) {  if (instance == null) {  instance = this;  }  instance.readPropertiesImpl(p);  return instance;  }  private void readPropertiesImpl(java.util.Properties p) {  String version = p.getProperty("version");  // TODO read your settings according to their version  }  public List<CD> getCdDatabase() {  return cdDatabase;  }  @Override  protected String preferredID() {  return PREFERRED\_ID;  }  } |
|  |

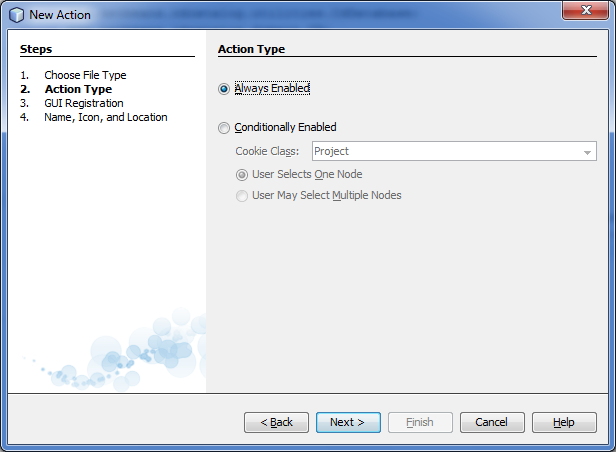
Tip-> you can get code of methods that have annotations ***@Override***  by clicking ***Alt + Insert*** on your keyboard or ***right click>Insert Code*** which gives you menu like on image below, and choose option **Override Method...** choose methods ***preferedID()*** and ***getPersistance()*** , and change their code, so it matches with previously given code.



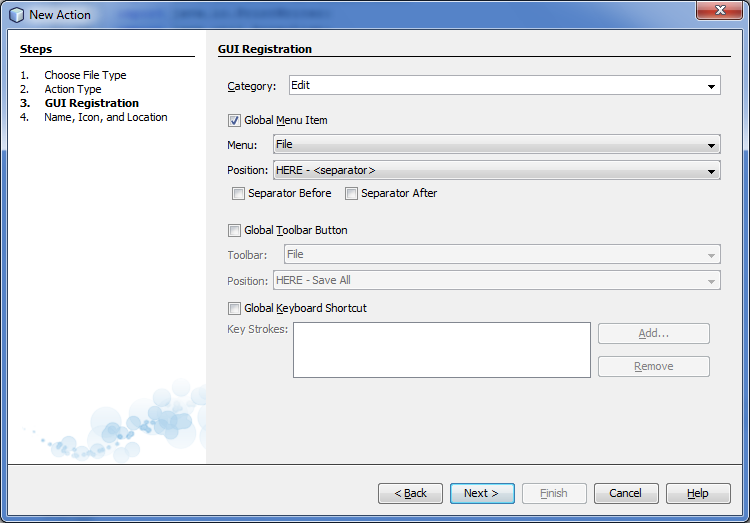
Add event handler for ***Add***  button on **CDCatalogTopComponent** form(double click button on form):

|  |
| --- |
| private void dbAddActionPerformed(java.awt.event.ActionEvent evt) {  CD cd = new CD(null, null);  cd.setArtist(txtArtist.getText().trim());  cd.setTitle(txtTitle.getText().trim());    cdDatabase.add(cd);    txtArtist.setText("");  txtTitle.setText("");  } |

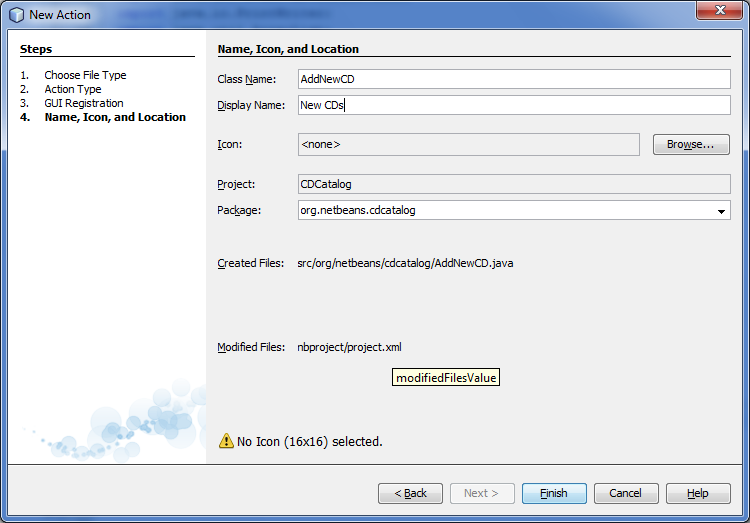
Now we will define action which will show (for start in console) all discs that we inserted. In ***CDCatalog*** module right click and choose option ***New>Action*** which will give you next window***:***

<

In this window you set availability of action, set ***Always Enabled***, and click Next.



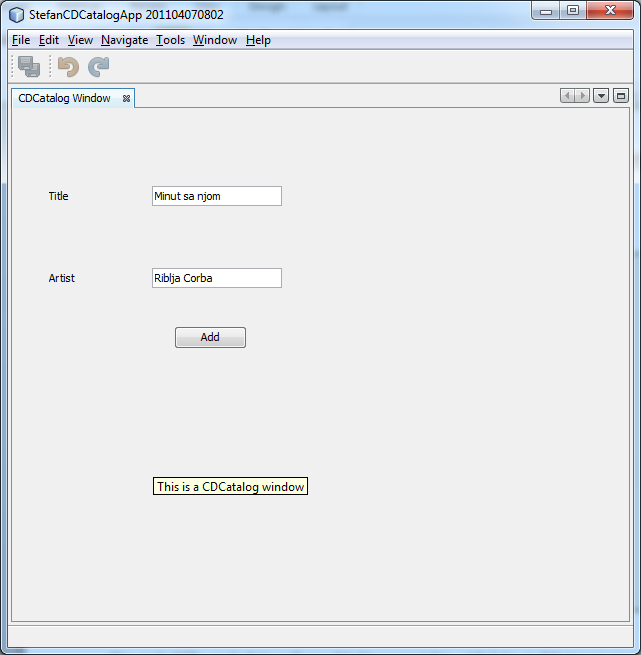
Set class name, that represents the action, and name that will be shown in main menu.



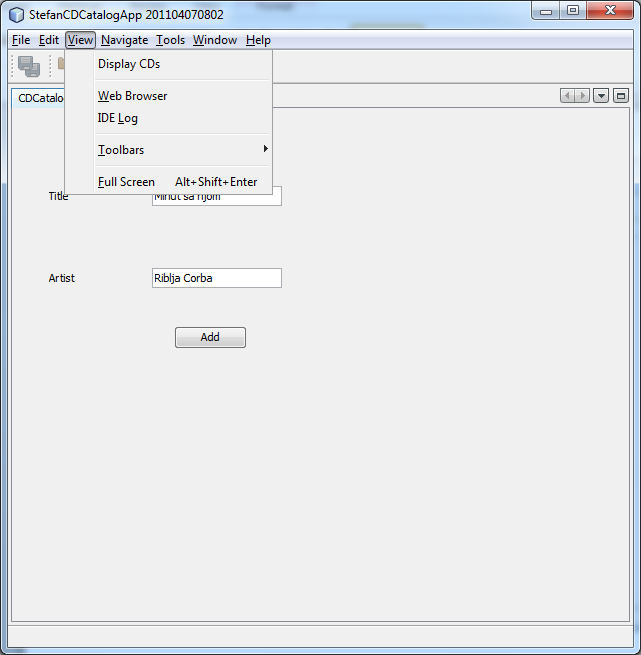
This is class code:

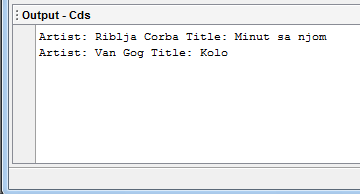
|  |
| --- |
| package org.netbeans.cdcatalog;  import java.awt.event.ActionEvent;  import java.awt.event.ActionListener;  import java.io.PrintWriter;  import java.util.ArrayList;  import org.netbeans.cdcatalog.cdutilities.CDUtilities;  import org.netbeans.cdcatalog.domain.CD;  import org.openide.awt.ActionRegistration;  import org.openide.awt.ActionReference;  import org.openide.awt.ActionReferences;  import org.openide.awt.ActionID;  import org.openide.util.NbBundle.Messages;  import org.openide.windows.IOProvider;  @ActionID(category = "Edit",  id = "org.netbeans.cdcatalog.AddCDsAction")  @ActionRegistration(displayName = "#CTL\_AddCDsAction")  @ActionReferences({  @ActionReference(path = "Menu/View", position = 0)  })  @Messages("CTL\_AddCDsAction=New CD")  public final class AddNewCDAction implements ActionListener {  @Override  public void actionPerformed(ActionEvent e) {  PrintWriter out = IOProvider.getDefault().getIO("Cds", false).getOut();  ArrayList<CD> cdDatabase = CDCatalogTopComponent.findInstance().getCdDatabase();  for (CD cd : cdDatabase) {  out.println(cd);  }  } |

After that, you can run your application (don’t forget to clean and build).



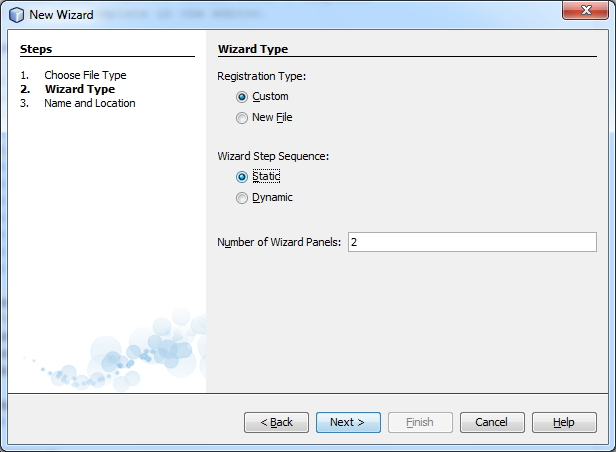
After inserting CDs, we can see that in output window (Ctrl+4):





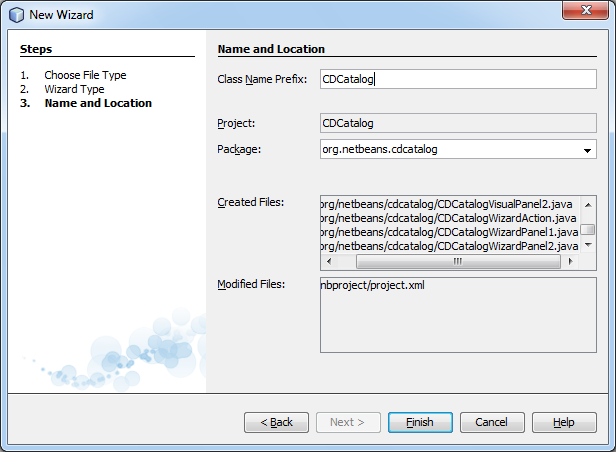
2. Adding wizard to existing module

Through this tutorial we worked with different wizards without noticing that (wizards for new actions, new windows), and now we will create wizard which will allow user to insert new CD in two steps. Right click on module ***CDCatalog*** and choose ***New > Wizard,*** that will give you next window, which is beginning of wizard for creating wizards:

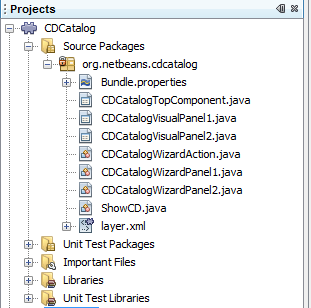


In first step you choose way of registering wizard actions and number of panels (windows) required.

After that you enter the name of class that will represent wizard, and below you can see document that will Netbeans create based on our settings.



After that, module looks like this:

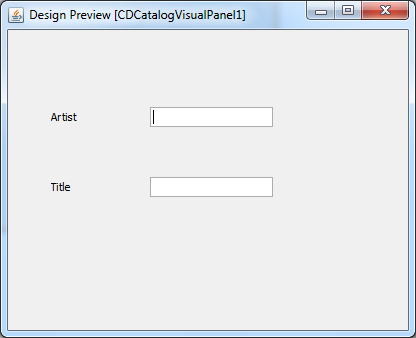


***CDCatalogVisualPanel*** represents class of type Jpanel, on which we will add SWING elements that we will use to gather data about CDs. It represents fornt page of our wizard.

***CDCatalogWizardPanel*** keeps data that user entered in the given step of wizard, before going to the next step.

***CDCAtalogWizardAction*** represents event (action) that will be triggered after successful finish of wizard.

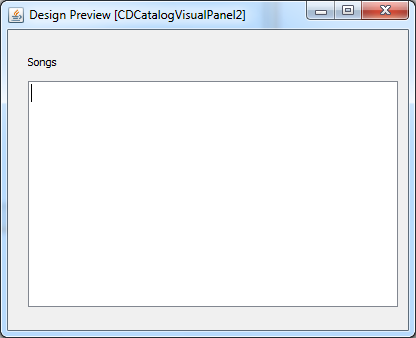
Set the form of our first step to look like this:



We need to change source code of form:

|  |
| --- |
| public final class CDCatalogVisualPanel1 extends Jpanel {  public CDCatalogVisualPanel1() {  initComponents();  }  @Override  public String getName() {  return „Enter CD’s title and artist“;  }  public javax.swing.JtextField getTxtArtist() {  return txtArtist;  }    public javax.swing.JtextField getTxtTitle() {  return txtTitle;  } |

Similar changes are required on second step:



|  |
| --- |
| public final class CDCatalogVisualPanel2 extends JPanel {  /\*\* Creates new form CDCatalogVisualPanel2 \*/  public CDCatalogVisualPanel2() {  initComponents();  }  @Override  public String getName() {  return "Enter list of songs";  }  public javax.swing.JTextArea gettxtSongs() {  return txtSongs;  } |

Now we will define action that will be activated after successful finish of wizard. Code that needs to be changed is marked in red:

|  |
| --- |
| @ActionID(category="Edit", id="org.netbeans.cdcatalog.CDCatalogWizardAction")  @ActionRegistration(displayName="Open CDCatalog Wizard")  @ActionReference(path="Menu/Tools", position=0)  public final class CDCatalogWizardAction implements ActionListener {  private WizardDescriptor.Panel[] panels;  public @Override  void actionPerformed(ActionEvent e) {  WizardDescriptor wizardDescriptor = new WizardDescriptor(getPanels());  // {0} will be replaced by WizardDesriptor.Panel.getComponent().getName()  wizardDescriptor.setTitleFormat(new MessageFormat("{0}"));  wizardDescriptor.setTitle("Your wizard dialog title here");  Dialog dialog = DialogDisplayer.getDefault().createDialog(wizardDescriptor);  dialog.setVisible(true);  dialog.toFront();  boolean cancelled = wizardDescriptor.getValue() != WizardDescriptor.FINISH\_OPTION;  if (!cancelled) {  String artist = (String) wizardDescriptor.getProperty("artist");  String title = (String) wizardDescriptor.getProperty("title");  String songs = (String) wizardDescriptor.getProperty("songs");    CD cd = new CD(title, artist, songs);    CDUtilities.getInstance().addCDs(cd);    NbPreferences.forModule(CD.class).putBoolean("refresh", true);    }  } |

After that we need to configure way of saving data that user entered in wizard forms. Below are given methods that need to be changed:

|  |
| --- |
| public class CDCatalogWizardPanel1 implements WizardDescriptor.Panel {  ..........  @Override  public void storeSettings(Object settings) {  CDCatalogVisualPanel1 panel = (CDCatalogVisualPanel1) getComponent();  ((WizardDescriptor)settings).putProperty("title", panel.getTxtTitle().getText().trim());  ((WizardDescriptor)settings).putProperty("artist", panel.getTxtArtist().getText().trim());  }  } |

|  |
| --- |
| public class CDCatalogWizardPanel2 implements WizardDescriptor.Panel {  ..........  @Override  public void storeSettings(Object settings) {  CDCatalogVisualPanel1 panel = (CDCatalogVisualPanel1) getComponent();  ((WizardDescriptor)settings).putProperty("songs", panel.gettxtSongs().getText().trim());  }  } |

1. Adding validators

Purpose of validators is to check if content of some object, concrete data in object, is in harmony with rules that application constructor set. If all rules are followed, application will continue with its normal flow, otherwise, we can see message about breaking the rules and application won’t allow change of state. We will create validators that won’t allow bad singers to add CDs. Create new module, and name it ***CDValidatorAPI (***package name: ***org.netbeans.cdcatalog.validator)***, it represents abstract interface for all validators that are considered with blocking of CD insertion for bad singers. In it, create one interface, with one method:

|  |
| --- |
| public interface ValidatorAPI {    public boolean validateCDs(CD cd);  } |

Considering that this module represents general pattern for all validators, we need to make it public for other modules (concrete validators), you need to set package of this module to public. Objects of ***CD*** class are subjects of validation, so we need to make required dependancies.

Now we can create module that will represent concrete validator for one singer. Call it ***KarleusaValidator*** (package name: ***org.netbeans.validator.karleusa***), and add dependancies towards modules ***ValidatorAPI,CDDomain*** and ***LookupAPI*** to it. Add one class with name ***KarleusaValidator***, implement interface ***ValidatorAPI*** this way:

|  |
| --- |
| import org.netbeans.cdcatalog.domain.CD;  import org.netbeans.cdcatalog.validator.api.CDValidatorInterface;  import org.openide.util.lookup.ServiceProvider;  @ServiceProvider(service=CDValidatorAPI.class)  public class ZeljkoValidator implements CDValidatorAPI {  @Override  public boolean validate(CD cd) {  if (cd.getArtist().equals("Jelena Karleusa") ){  return false;  } else {  return true;  }  }  } |

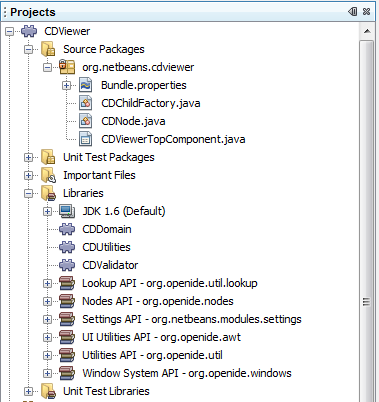
Now that our application is safe form bad authors and bad music, we can devote ourselves to more beautiful view of CDs.

1. Advanced way of showing CDs

Considering that showing CDs in console (output window) doesn’t give us much options for improvement, and isn’t pretty, we will create module that is specially intended to show new CDs. For that purpose we will use special package of classes org.openide.viewer, which is located in module ***NodesAPI.*** It allows us to show object by using Node elements. There are different ways of representation of nodes. We will show IconView and BeanTreeView (CDs will be represented through icons or tree). For more information about it, visit:

<http://bits.netbeans.org/dev/javadoc/org-openide-explorer/org/openide/explorer/view/package-summary.html>

Create module ***CDViewer*** (package name: **org.netbeans.cdviewer**), and add dependencies toward these modules: ***CDDomain, CDUtilities, Lookup API, NodesAPI, CDValidator*** and ***Utilities API***. Create these classes: ***CDNode, CDChildFactory***, and after that window ***CDViewer*** (for window position choose ***explorer***)***.***



Code of ***CDNode*** class:

|  |
| --- |
| public class CDNode extends BeanNode {  public CDNode(CD bean) throws IntrospectionException {  super(bean);  setDisplayName(bean.getTitle());  setShortDescription(bean.getArtist());  } |

Code of ***CDChildFactory*** class:

|  |
| --- |
| public class CDChildFactory extends ChildFactory<CD> implements PreferenceChangeListener {  public CDChildFactory() {  Preferences refPreferences = NbPreferences.forModule(CD.class);  refPreferences.addPreferenceChangeListener(this);  }  @Override  protected boolean createKeys(List<CD> list) {  ArrayList<CD> cdDatabase = CDUtilities.getInstance().getCDs();  for (CD cd : cdDatabase) {  boolean allOK = true;  Collection<? extends ValidatorAPI> allValidators = Lookup.getDefault().lookupAll(ValidatorAPI.class);  for (ValidatorAPI validator : allValidators) {  if (!validator.validateCDs(cd)) {  allOK = false;  }  }  if (allOK) {  list.add(cd);  } else {  StatusDisplayer.getDefault().setStatusText("Failed: " + cd.getArtist());  }  }  return true;  }  @Override  protected Node createNodeForKey(CD key) {  Node cdNode=null;  try {  cdNode=new CDNode(key);  } catch (IntrospectionException ex) {  Exceptions.printStackTrace(ex);  }  return cdNode;    }  @Override  public void preferenceChange(PreferenceChangeEvent evt) {  refresh(true);  NbPreferences.forModule(CD.class).putBoolean("refresh", false);  }  } |

Code of ***CDViewerTopCOmponent*** class:

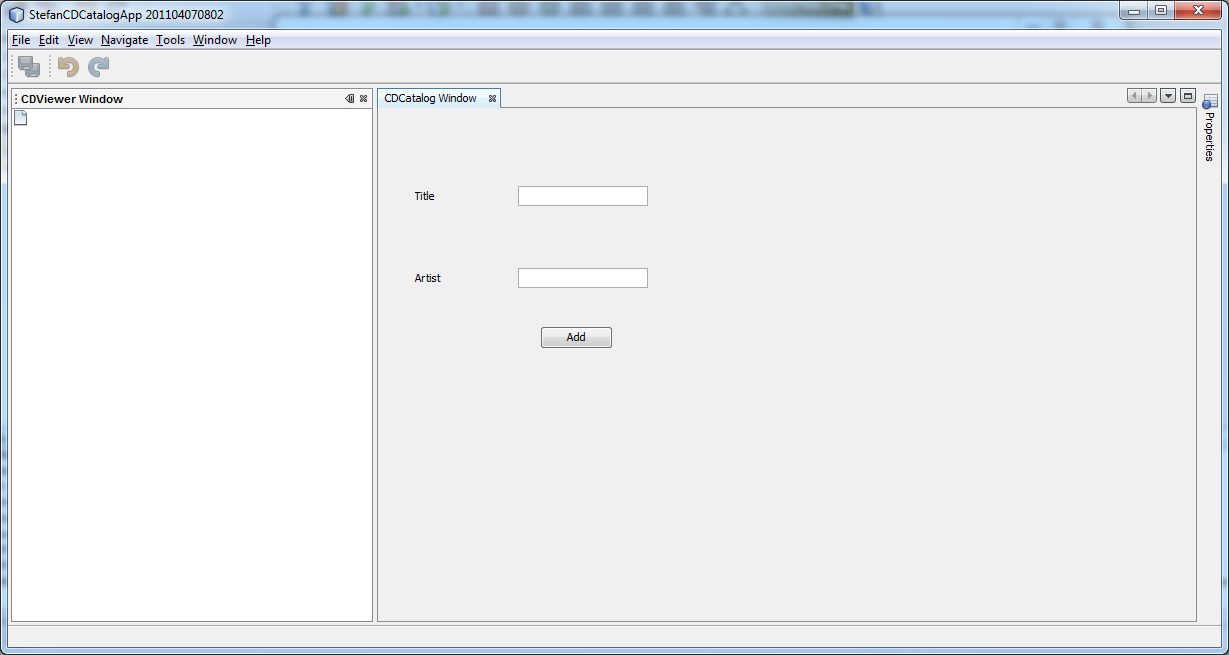
|  |
| --- |
| import java.awt.BorderLayout;  import org.openide.util.NbBundle;  import org.openide.windows.TopComponent;  import org.netbeans.api.settings.ConvertAsProperties;  import org.openide.awt.ActionID;  import org.openide.awt.ActionReference;  import org.openide.explorer.ExplorerManager;  import org.openide.explorer.ExplorerUtils;  import org.openide.explorer.view.BeanTreeView;  import org.openide.explorer.view.IconView;  import org.openide.nodes.AbstractNode;  import org.openide.nodes.Children;  import org.openide.nodes.Node;  @ConvertAsProperties(dtd = "-//org.netbeans.cdviewer//CDViewer//EN",  autostore = false)  @TopComponent.Description(preferredID = "CDViewerTopComponent",  //iconBase="SET/PATH/TO/ICON/HERE",  persistenceType = TopComponent.PERSISTENCE\_ALWAYS)  @TopComponent.Registration(mode = "explorer", openAtStartup = true)  @ActionID(category = "Window", id = "org.netbeans.cdviewer.CDViewerTopComponent")  @ActionReference(path = "Menu/Window" /\*, position = 333 \*/)  @TopComponent.OpenActionRegistration(displayName = "#CTL\_CDViewerAction",  preferredID = "CDViewerTopComponent")  public final class CDViewerTopComponent extends TopComponent implements ExplorerManager.Provider{  ExplorerManager controler = new ExplorerManager();    public CDViewerTopComponent() {  initComponents();  setName(NbBundle.getMessage(CDViewerTopComponent.class, "CTL\_CDViewerTopComponent"));  setToolTipText(NbBundle.getMessage(CDViewerTopComponent.class, "HINT\_CDViewerTopComponent"));  setLayout(new BorderLayout());  BeanTreeView btv = new BeanTreeView();  //IconView icv = new IconView();  add(btv, BorderLayout.CENTER);    Children child = Children.create(new CDChildFactory(), true);  Node root = new AbstractNode(child);    controler.setRootContext(root);    associateLookup(ExplorerUtils.createLookup(controler, getActionMap()));  }  @Override  public ExplorerManager getExplorerManager() {  return controler;  }  .............. |

***CDNode*** ->In this class we set which type will be represented through node elements.

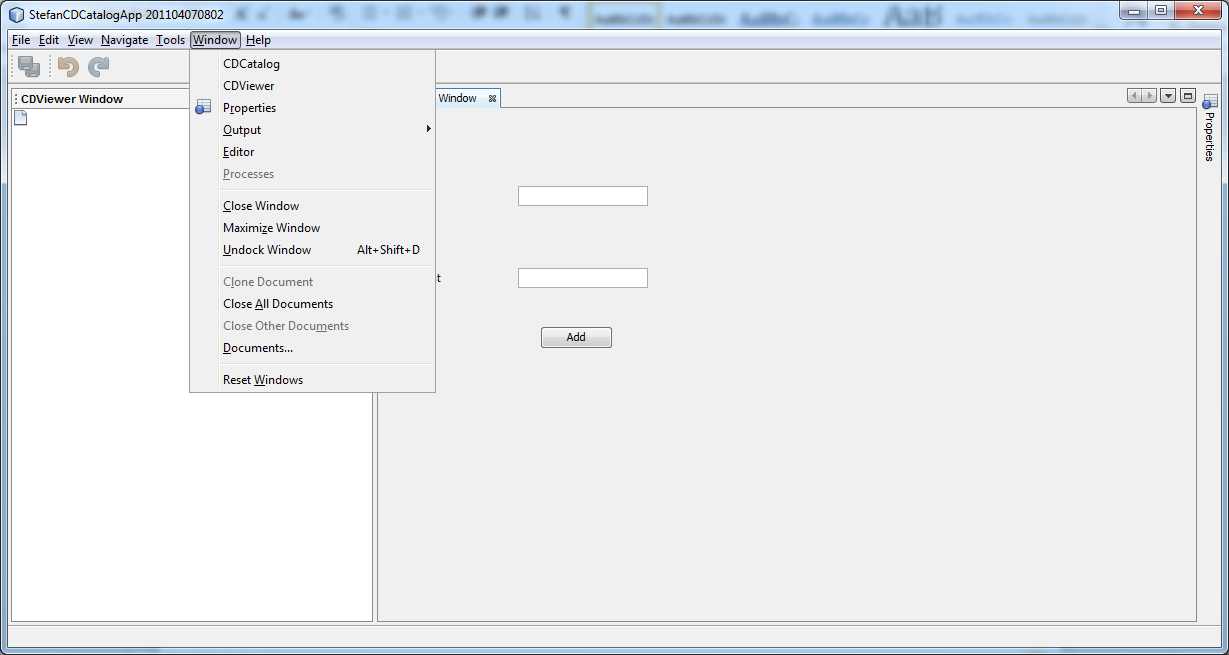
***CDChildFactory*** -> In this class we set way of importing concrete CDs (or other persistent entities), that user will add through wizard, to showing window. This is where validators are implemented.

***CDViewerTopCOmponent ->*** In this class we define concrete way of showing CDs(using icons or tree).

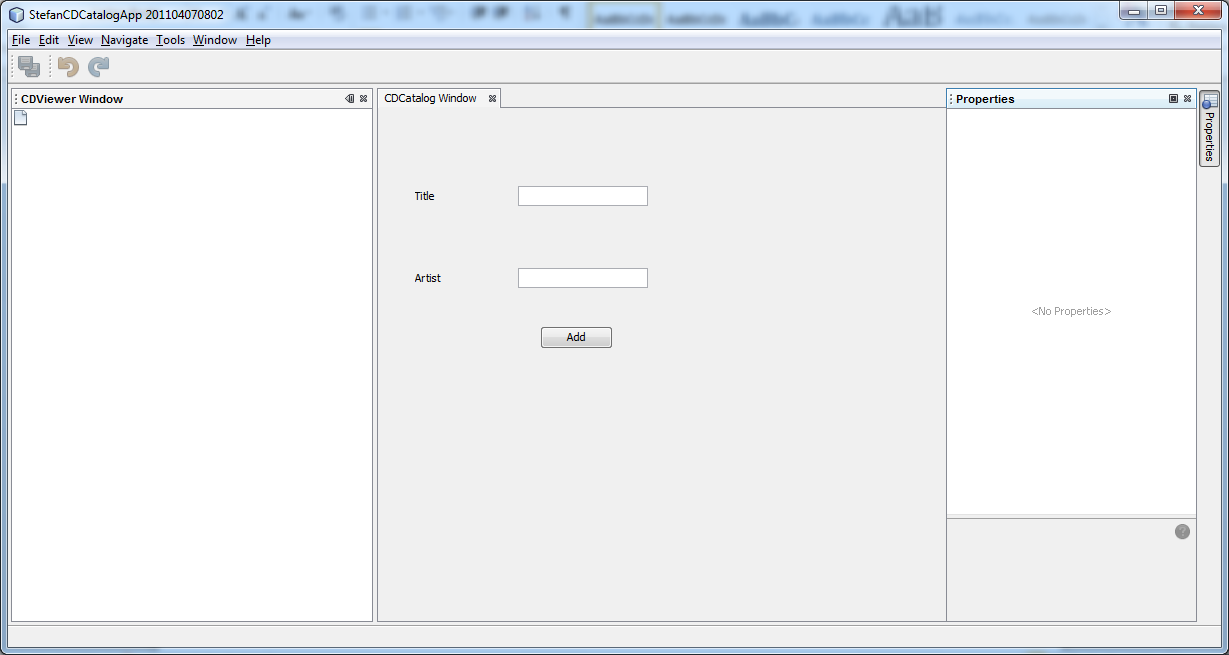
After successful implementation of given code, run application. Result is next array of windows:



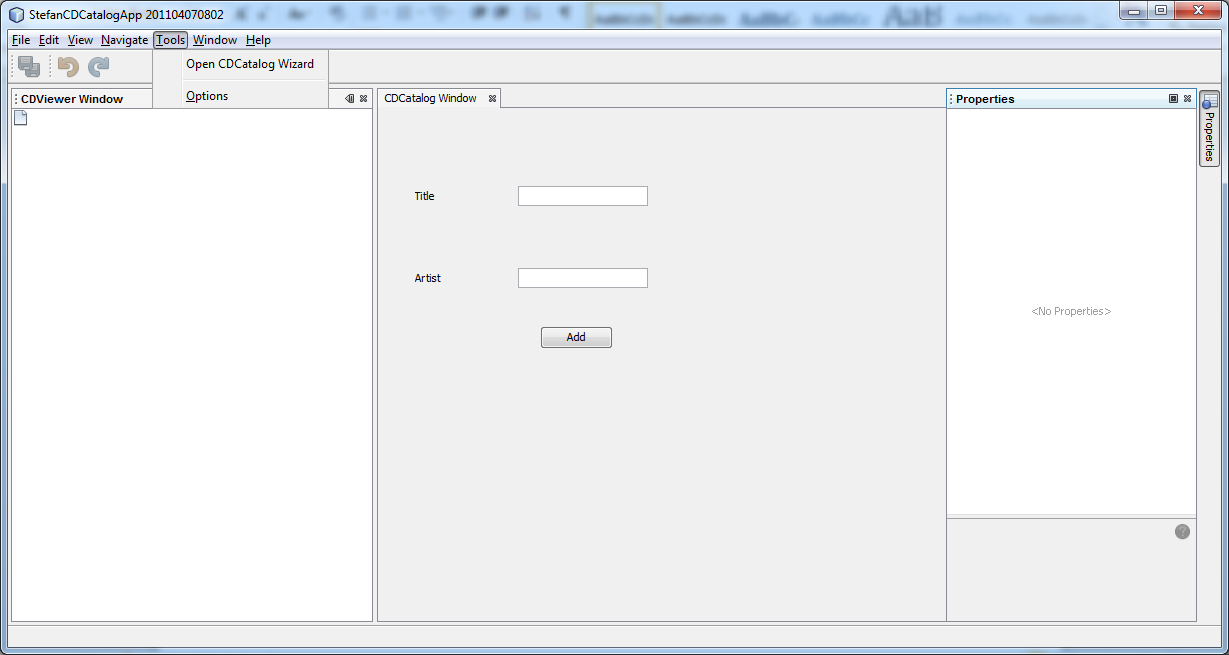
Show ***Properties***  menu as on image below:



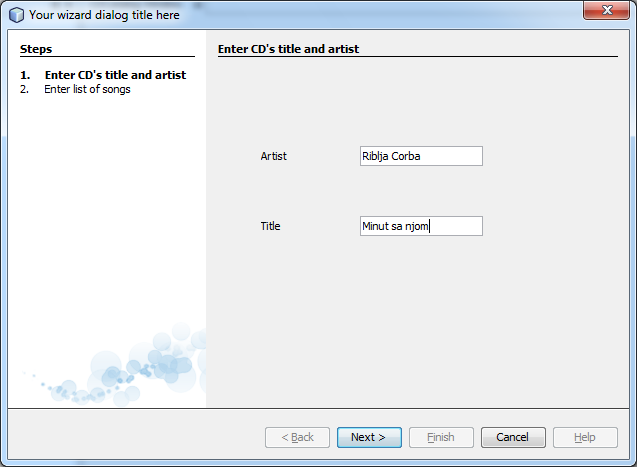
Then „dock“ it (***Pin*** option in upper right corner of window):

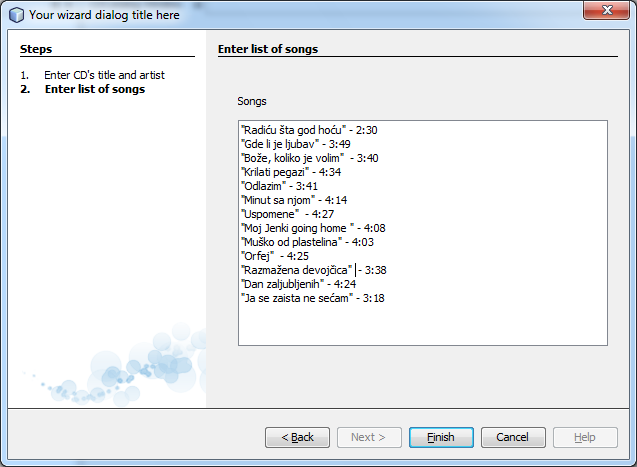


After that start our wizard for adding new CDs:

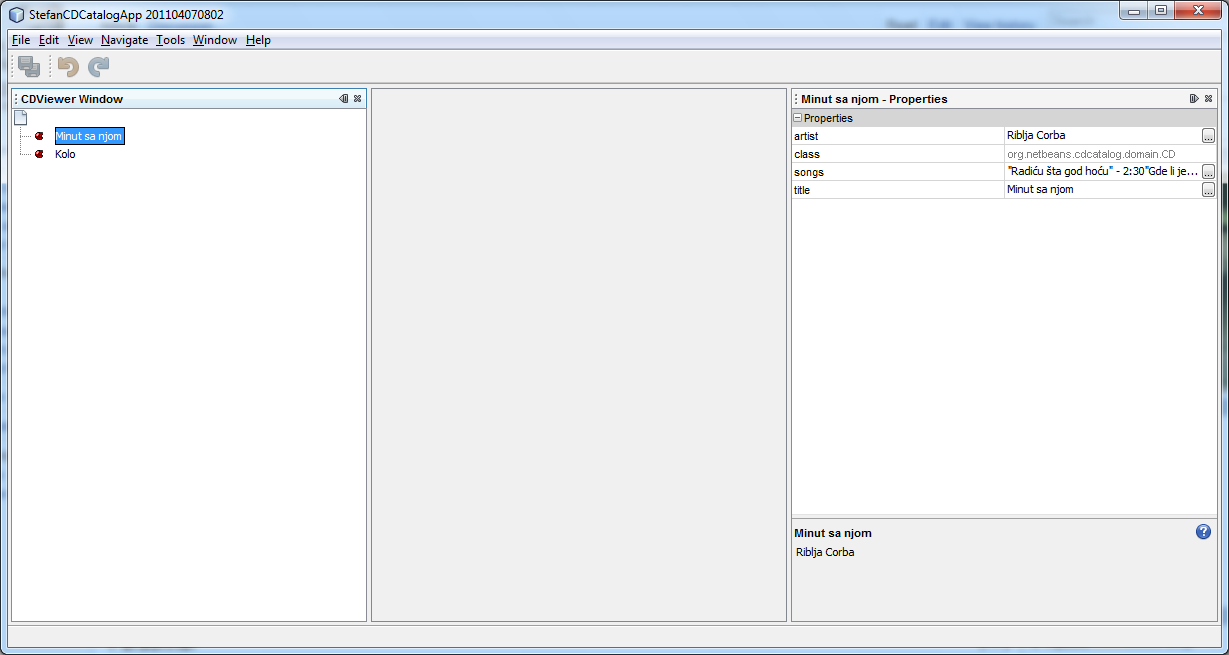


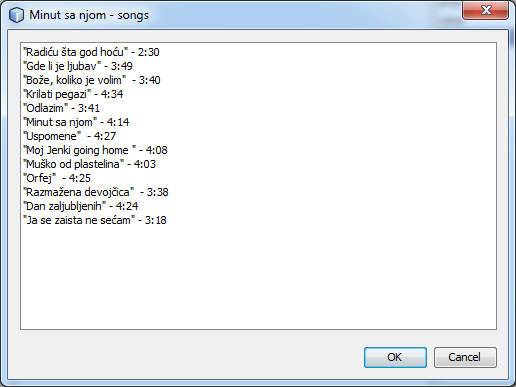
Add few CDs, and few songs:





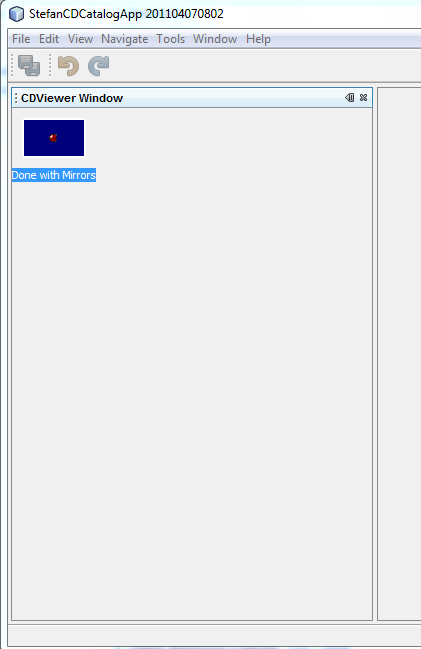
Click on one album and notice that Properties window lists data about your album. You can access them by pressing button „...“.





To change way of representing albums, it is enough to uncomment part of code that considers ***IconView,*** and to change next line of code:

|  |
| --- |
| ...  //BeanTreeView btv = new BeanTreeView();  IconView icv = new IconView();  add(icv, BorderLayout.CENTER);  ... |



Try to add some author that doesn’t know to sing (ofcourse under condition that you created appropriate validator).

For more tutorials visit:

<http://netbeans.org/features/platform/all-docs.html>