NetBeans and NetBeans Platform
Overview

• History
  > originally MFF student project (Xelfi)
• IDE
  > Java, C/C++, PHP, Python,...
• Platform
  > rich clients development
  > Swing
Sources

• NetBeans source code
  > http://www.netbeans.org/downloads/zip.html

• API Javadoc
  > http://bits.netbeans.org/dev/javadoc/index.html

• Planet NetBeans
  > http://planetnetbeans.org/

• Numerous NetBeans bloggers
  > e.g. https://blogs.oracle.com/geertjan/
Getting Started with the NetBeans Platform
Extending NetBeans

• Possibilities
  > single module
  > suite of modules
  > standalone application
    > like a suite of modules
  > wrapper module of an existing JAR
Extending NetBeans

• Possibilities
  > single module
  > suite of modules
  > standalone application
    > like a suite
  > wrapper module of an existing JAR
Single module creation

**Steps**
1. Choose Project
2. **Name and Location**
3. Basic Module Configuration

**Name and Location**
- **Project Name:** ExampleModule
- **Project Location:** /home/hnetynka/devel/NB2014
- **Project Folder:** /home/hnetynka/devel/NB2014/ExampleModule

**Standalone Module**
- **NetBeans Platform:** Development IDE

**Add to Module Suite**
- **Module Suite:**

**Buttons:**
- < Back
- Next >
- Finish
- Cancel
- Help
Suite & standalone application

- **Suite**
  - set of modules that have to be loaded together
- **Standalone application**
  - same as the suite
  - configured to be run as a standalone application
## Dependencies

![Project Properties - StandaloneApp1](image)

**Categories:**
- Sources
- Libraries
- Application
- Installer
- Hints

### Java Platform:
- JDK 1.8 (Default)

### NetBeans Platform:
- Development IDE

#### Platform Modules

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Included</th>
<th>Origin</th>
</tr>
</thead>
<tbody>
<tr>
<td>apisupport</td>
<td></td>
<td>Platform</td>
</tr>
<tr>
<td>enterprise</td>
<td></td>
<td>Platform</td>
</tr>
<tr>
<td>extide</td>
<td></td>
<td>Platform</td>
</tr>
<tr>
<td>harness</td>
<td></td>
<td>Platform</td>
</tr>
<tr>
<td>ide</td>
<td></td>
<td>Platform</td>
</tr>
<tr>
<td>java</td>
<td></td>
<td>Platform</td>
</tr>
<tr>
<td>javafx</td>
<td></td>
<td>Platform</td>
</tr>
<tr>
<td>mobility</td>
<td></td>
<td>Platform</td>
</tr>
<tr>
<td>nb</td>
<td></td>
<td>Platform</td>
</tr>
<tr>
<td>php</td>
<td></td>
<td>Platform</td>
</tr>
<tr>
<td>platform</td>
<td></td>
<td>Platform</td>
</tr>
<tr>
<td>profiler</td>
<td></td>
<td>Platform</td>
</tr>
<tr>
<td>webcommon</td>
<td></td>
<td>Platform</td>
</tr>
<tr>
<td>websvccommon</td>
<td></td>
<td>Platform</td>
</tr>
</tbody>
</table>

**Warning:** Module JavaFX wrapper in platform requests the token org.openide.modules.jre.JavaFX but there are no known providers.

[NetBeans](image)
Branding application
Executing application/module

- **Run**
  - executes new instance of IDE with installed modules
- **Install /Reload in Development IDE**
  - runs module in the development instance of IDE
    - no new instance is executed
  - available for standalone modules only
Distribution

- Modules ~ NBM files
  > common JAR file
  > with extra info in its manifest
- Standalone apps
  > ZIP files or
  > JNLP application
Converting an existing applications
Generic process

• “Library” without UI => library wrapper

• Application with UI
  > converting the application by parts
    > Swing panel => TopComponent
    > Actions => CallableSystemAction, CallbackSystemAction
    > Menu => NB menu via layer
    > ...

NetBeans
Converting application

- Levels of conformance
  - Level 0:Launchable
    - enhancing the manifest file with NetBeans entries
    - adding dependencies to other modules
    - adding menu item to “launch” the application
  - Level 1: Integration
    - using NetBeans Window system and Dialog API
    - initialization via ModuleInstAl or META-INF/services
  - Level 2: Use case support
    - follow NetBeans paradigms
  - Level 3: Aligned
    - reusing as much as possible, cooperating with other modules
Example

• Converting the Anagram Game
  > available as a std example
    > New Project → Samples → Java → Anagram game

• Step 1 – create new module
Step 1

1. Choose Project
2. **Name and Location**
3. Basic Module Configuration

**Name and Location**

- **Project Name:** NBAagram
- **Project Location:** /home/hnetynka/devel/NB2014
- **Project Folder:** /home/hnetynka/devel/NB2014/NBAagram

**Standalone Module**

- **NetBeans Platform:** Development IDE

**Add to Module Suite**

- **Module Suite:**
Step 1

New Module

Steps
1. Choose Project
2. Name and Location
3. Basic Module Configuration

Basic Module Configuration

- Code Name Base: cz.cuni.mff.nb.anagram
  (e.g. "org.yourorg.modulename")
- Module Display Name: NBAngram
- Localizing Bundle: cz/cuni/mff/nb/anagram/Bundle.properties
  [Generate OSGi Bundle]

< Back  Next >  Finish  Cancel  Help
Step 1

- NBackagram
  - Source Packages
    - cz.cuni.mff.nb.anagram
      - Bundle.properties
  - Important Files
  - Libraries
Getting Level 0

- Copy classes of the anagram game to our module
- Add new action
- Implement the action to show the anagram game window
Copying game classes
Adding an action

Steps
1. Choose File Type
2. Action Type
3. GUI Registration
4. Name, Icon, and Location

GUI Registration
- Category: Window
- Menu: Window
- Position: HERE - Projects
- Global Menu Item
- Global Toolbar Button
- Global Keyboard Shortcut
- Key Strokes:

Steps
1. Choose File Type
2. Action Type
3. GUI Registration
4. Name, Icon, and Location

Name, Icon, and Location
- Class Name: AnagramGameShowAction
- Display Name: Show Anagram Game
- Icon: <none>
- Icon: Browse...
- Project: NDAagrams
- Package: cz.cuni.mff.nb.anagram
- Source Files: src/cz/cuni/mff/nb/anagram/AnagramGameShowAction.java
- Modified Files: nbproject/project.xml

No Icon (16x16) selected.
package cz.cuni.mff.nb.anagram;

import com.toy.anagrams.ui.Anagrams;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

// ...annotations are here...
public final class AnagramGameShowAction implements ActionListener {

    public void actionPerformed(ActionEvent e) {
        new Anagrams().setVisible(true);
    }

}
• Execute new IDE with out module
  > “Run” in the right-click menu

• Pack module as NBM file

• Distribute the module ;(-)
Converting an existing applications

Obtaining Level 1
Converting to Level 1

- Using TopComponent and Dialog API
  > JFrame → TopComponent
Process

- Create new TopComponent
  > “Window component”
- Copy Anagram panel to the created window
- Copy local variables
Creating new TopComponent

• Choosing position
  > in which are the component has to appear
Creating new TopComponent

- Name prefix for created classes etc.
Creating new TopComponent

NBAnagram

Source Packages
- com.toy.anagrams.lib
- com.toy.anagrams.ui
- cz.cuni.mff.nb.anagram
  - AnagramGameShowAction.java
  - AnagramGameTopComponent.java
  - Bundle.properties

Important Files

Libraries
Copying panel to the window

- Select JPanel in the Anagram class
- Copy it
- Paste it to the TopComponent class

Ctrl + C

Ctrl + V
Copying variable

- Copy variables from the Anagrams class
- Paste them to the TopComponent

```java
public class Anagrams extends JFrame {
    public static void main(String[] args) {
        new Anagrams().setVisible(true);
    }

    private int wordIdx = 0;
}
```

```java
final class AnagramTopComponentTopComponent extends TopComponent {
    private int wordIdx = 0;

    private static AnagramTopComponentTopComponent instance;
}
```
Overview

• Window system
  > management of windows (panels) in the NetBeans

• Basic Elements
  > TopComponent
    > JPanel with additional methods
  > Mode
    > in which are the component has to be placed
      - i.e. docking mode
  > WindowManager
    > managing state of UI
  > TopComponentGroup
    > set of windows that should be activated together
  > Roles (Perspectives)
    > switching between window layouts (new in 7.1)

• UI = Swing
TopComponent

- open()
- close()
- requestVisible()
- requestActive()
- componentHidden()
- componentShowing()
- componentDeactivated()
- componentActivated()
- componentClosed()
- componentOpened()
TopComponent

• Persisting session across sessions
  > TopComponent implements Externalizable

• Persistence modes
  > PERSISTENCE_ALWAYS
  > PERSISTENCE_NEVER
  > PERSISTENCE_OPENED
• Persistence
  - @ConvertAsProperties
    - defines public ID of a DTD for the storing file
      - identification of the file
  - methods
    readProperties(Properties p)
    writeProperties(Properties p)
    - reading/saving via them
Mode

- Position in application
- Many predefined
  > editor, navigator, output,....
- Own one can be defined
  > defined by XML
Mode

- Position
- Many predefined: editor, navigator, output, ...
- Own one can be defined by XML
• Opening a component in a particular mode programmatically

```java
public void open() {
    Mode mode = WindowManager.getDefault().findMode("mode");
    if (mode != null) {
        mode.dockInto(this);
    }
    super.open();
}
```
TopComponent groups

• Set of windows that should be activated together
• Defined by file descriptors
  > wsgrp
  > wstcgrp
TopComponent groups

```xml
<group version="2.0">
  <module name="org.netbeans.modules.windowgroupsample" spec="1.0" />
  <name unique="MyGroup" />
  <state opened="false" />
</group>

<tc-group version="2.0">
  <module name="org.netbeans.modules.windowgroupsample" spec="1.0"/>
  <tc-id id="OneTopComponent" />
  <open-close-behavior open="true" close="true" />
</tc-group>
```
Roles (Perspectives)

- New in 7.1
- Easy switching between window layouts

- `@TopComponent.Registration(mode = "editor", openAtStartup = true, role="admin")`
- `WindowManager.getDefault().setRole("admin");`
Modular Applications
Architecture

NetBeans IDE

NetBeans Platform

NetBeans Application

Swing / JDK

Java VM
Module System

**Module Suite** (= Deployment Unit)

**Module A** (= JAR File)

- META-INF/manifest.mf
- layer.xml
- META-INF/services/*
- *.class
- Bundle.properties

**Module B**

- ...

**Module C**

- ...

- Well-defined module dependencies
- Lazy loading / Unloading
- layer.xml for declarative registrations (file system)
- Bundle.properties for internationalization
Information Hiding

Public packages are explicitly defined in manifest.mf (project.xml).
Linkage Dependencies

- Each module loaded by own classloader
- Link to classes of modules they explicitly depend on.
- No cyclic dependencies
Runtime Dependencies

- Modules can require presence of others during runtime
- Useful for modular libraries
- to require implementations

Diagram:

- Module A
- Module B
- Module C

May require presence
Types of library

- Simple library
  > one impl, put it on classpath and use
- Reference Impl + Vendor Impl
  > You trust that the Vendor impl conforms to the spec
- Modular Library – the API is separate from the implementation
  > Multiple implementations possible
  > Spec conformance is enforced by design
  > API must find its implementation
  > You need a registry of things
The Java Extension Mechanism

- ServiceLoader
Lookup – NetBeans solution

- Small, NetBeans independent library
  > org-openide-util.jar
- Works with any version of Java
- A Lookup is dynamic
  > Can fire changes
- A Lookup is instantiable
- Lookups are composable
Lookup – service discovery

- Global Lookup Patterns
  - Pseudo-singletons:
    ```java
    Lookup.getDefault().lookup ( SomeClass.class )
    ```
  - Global services
    ```java
    Lookup.Result<SomeClass> r = Lookup.getDefault().lookupResult(SomeClass.class);
    Collection <SomeClass> c = r.allInstances();
    ```

- Simple registration
  - Using @ServiceProvider annotation
    ```java
    @ServiceProvider(service=SomeClass.class)
    public class SomeClassImpl extends SomeClass {
    }
    ```
Listening for changes

Lookup l = Lookup.getDefault();
Lookup.Result<SomeClass> r =
    l.lookupResult ( SomeClass.class );

r.addLookupListener (new LookupListener() {
    public void resultChanged (LookupEvent e) {
        //handler code here
    }
});
Clean Unloading/Reloading

- Get a Lookup.Result from the META-INF/services
  
  - Lookup (Lookup.getDefault() +/-)
  
- If a module is uninstalled, it will fire changes
Lookup

• Used “almost” everywhere in NetBeans
  > solution to “almost” everything
Example: Project API

- Associates a directory on disk with a Lookup
- Defines interfaces that may be in that Lookup

```java
public interface Project extends Lookup.Provider {
    FileObject getProjectDirectory();
    Lookup getLookup();
}
```
Example: Selection in NetBeans

- Each main window tab has its own Lookup
  - Some tabs show Nodes, which also have Lookups, and proxy the selected Node's Lookup
- A utility Lookup proxies the Lookup of whatever window tab has focus

```java
Lookup lkp = Utilities.actionsGlobalContext();
```
Lookup is a place

• A space objects swim into and out of
• You can observe when specific types of object appear and disappear
• You can get a collection all of the instances of a type in a Lookup
Useful utility implementations

- AbstractLookup + InstanceContent
  > Lookup whose contents you can manage
- Lookups.singleton( Object )
  > one item Lookup
- Lookups.fixed( Object[] )
  > unchanging Lookup
- Lookups.exclude ( Lookup, Class[] );
- ProxyLookup ( Lookup[] otherLookups )
  > compose multiple lookups
Defining extension point

- interface to be implemented
  interface Foo {...}

- extension point
  Lookup.Result<Foo> rslt =
    Lookup.getDefault().lookupResult(Foo.class);
  for (Foo foo : rslt.allInstances()) {...}

- in third-party modules
  @ServiceProvider(service=Foo.class)
  class FooImpl1 implements Foo {...}
Context sensitive actions

- Action observes presence of a type in the global context
  - xxxCookie
  - Utilities.actionsGlobalContext();
SaveCookie vs. Savable

- Saving of documents in an editor, etc
  - by adding SaveCookie implementation to the global context
    - std Save button listens for the SaveCookie
- From 7.1
  - Savable instead of SaveCookie
    - SaveCookie still exists and implement Savable
  - AbstractSavable abstract implementation of Savable
Communication between windows

- A window puts an object to its lookup
- Another window listens on the global context

```java
InstanceContent ic = new ...
associateLookup(
    new AbstractLookup(this));
ic.set(...)

public void componentOpened() {
    result = Utilities.
        actionsGlobalContext().
        lookupResult(Foo.class);
    result.addLookupListener(this);
    result.allItems();
}

public void resultChanged(LookupEvent ev) {
    Collection<? extends Foo> words =
        result.allInstances();
    if (!words.isEmpty()) {
        ....
    }
}