Kezdeti lépések (Quick start)

A NetBeans Platform

A **NetBeans Platform** egy alkalmazás keretrendszer, elsősorban Java asztali alkalmazásokhoz. A NetBeans Platform fő előnye a meghatározott moduláris architektúra. A további előnyök a NetBeans Platform újrafelhasználható megoldásai, például a dokkoló keretrendszer és a készen kapott összetevők (plug-in) amelyek az SDK, a NetBeans IDE, de különösen a díjnyertes "*Matisse*" *GUI Builder* által biztosított eszközökkel kombinálva, a grafikus felhasználói felület komponenseinek tervezéséhez lehetnek segítségül.

Ebben a gyors bevezetőben egy nagyon egyszerű példa segítségével megismerkedhetsz a moduláris fejlesztés előnyeivel és felhasználási lehetőségeivel a Java asztali alkalmazások terén. A példát Thomas Würthinger, a Linzi Johannes Kepler Egyetem doktorandusza szállította. Amint elsajátítottad a gyors bevezetőben bemutatott fogalmakat, készen állhatsz a NetBeans Platform tanulóösvényre, amely sokféle, a NetBeans Platformmal kapcsolatos különböző forgatókönyvet lefedő tutorialt kínál.

Ha új vagy a NetBeans Platformhoz, nagyon ajánlott beszerezni a '*NetBeans Platform for Beginners*' (2014-ben megjelent) könyvet.

Részek

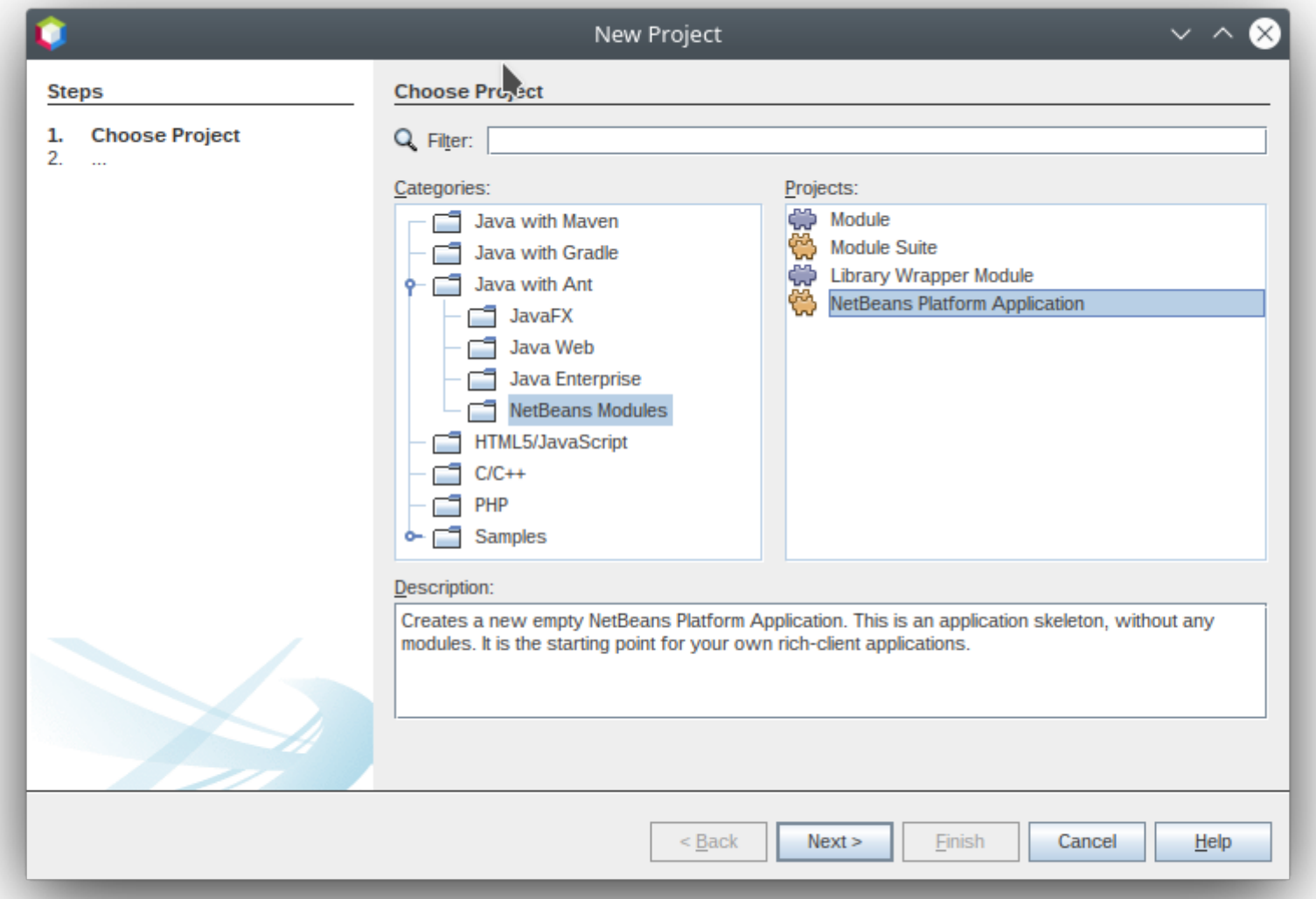
1. rész: Egymodulos NetBeans Platform alkalmazás
2. rész: Moduláris alkalmazás a Lookup használatával
3. rész: Közzététel és feliratkozása a lookupra

Egymodulos NetBeans Platform alkalmazás

Alkalmazás létrehozása

A következőkben létrehozzuk az első NetBeans Alkalmazásod.

1. Fájl → Új projekt - majd ezek után válasszuk a NetBeans Modulokat. Válasszuk a NetBeans Platform Alkalmazást és a következőt kell látnunk:



A fenti 4 sablon közötti különbség a következő:

NetBeans platformalkalmazás. Egy projekt, amely csoportosítja a modulprojekteket és a könyvtári átalakító modulprojekteket, amelyek egymástól függenek, és lehetővé teszi ezek együttes telepítését egy egységként. Automatikusan tartalmazza a NetBeans Platformot alkotó modulok egy részét.

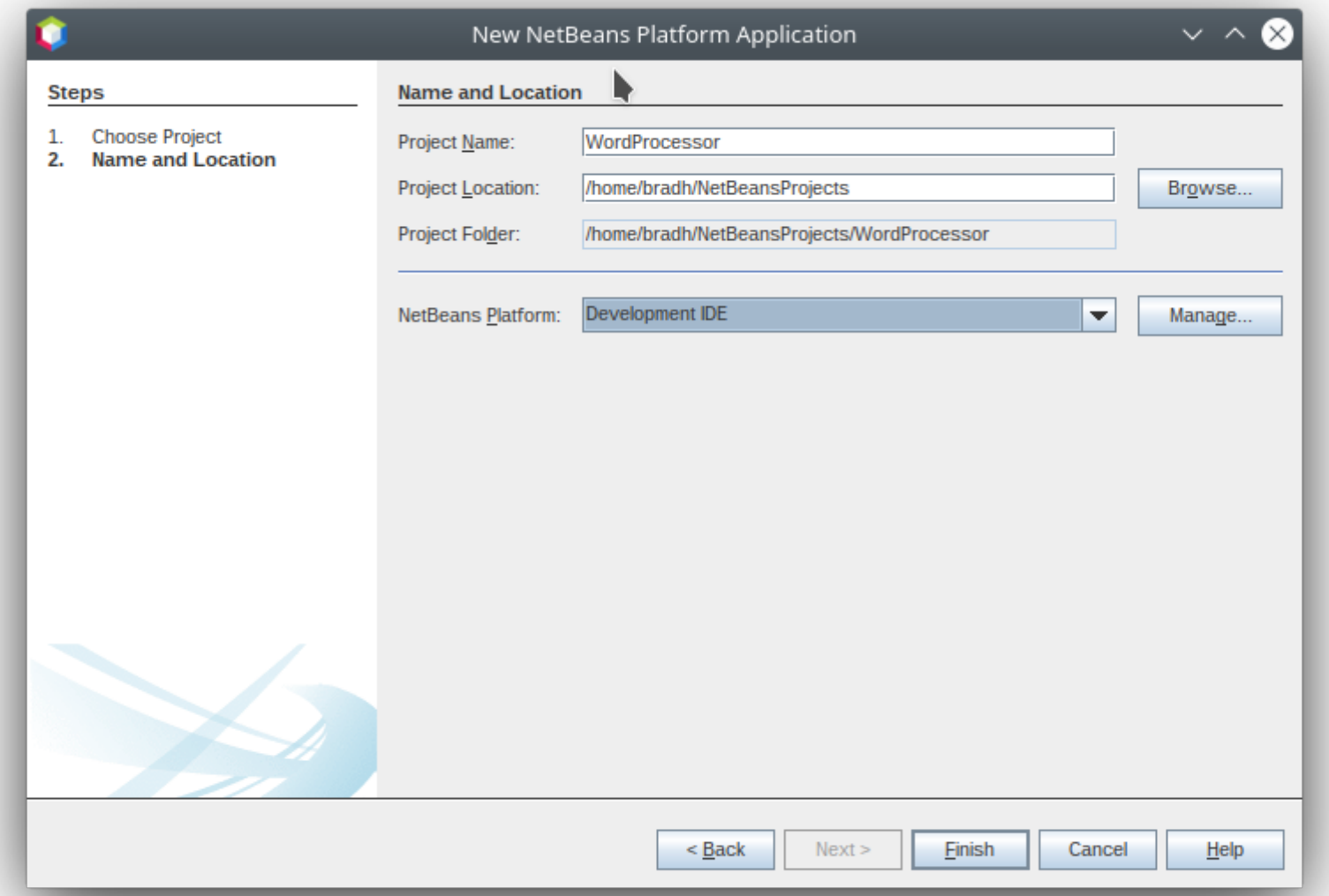
Modul Suite. Ugyanaz, mint fent, azzal a különbséggel, hogy az előre beépített modulok nem csak a NetBeans Platformhoz kapcsolódnak – ebben az esetben a NetBeans IDE-t alkotó összes modul is benne van.

Library Wrapper modul. Olyan projekt, amely egy könyvtár JAR-fájlt helyez el az osztályútvonalára, és nyilvános csomagként exportálja a JAR-fájl néhány vagy összes csomagját a modulból.

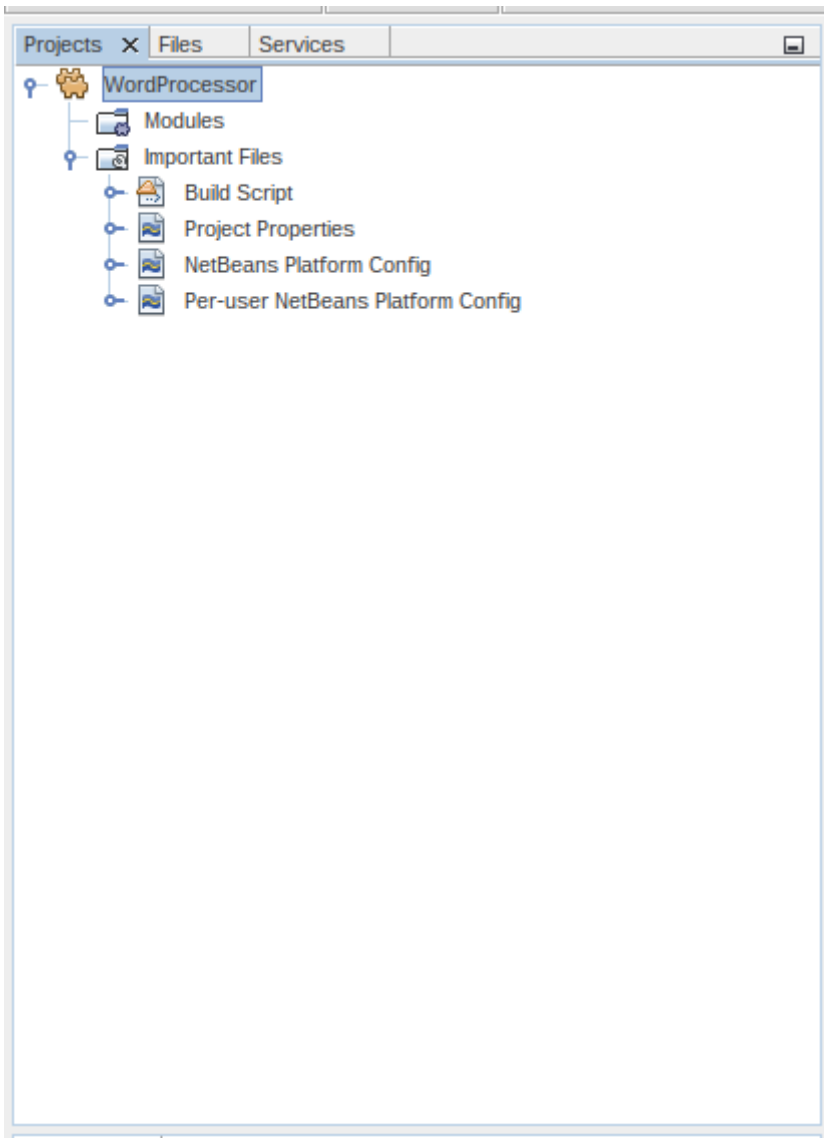
Modul. Projekt egy NetBeans Platformra épített modul vagy alkalmazás funkcionalitásának, üzleti logikájának és felhasználói felületének megvalósítására.

Miután választottunk kattintsunk a **következőre**.

2. Nevezd el az alkamazásod és válassz neki mappát a tároláshoz.



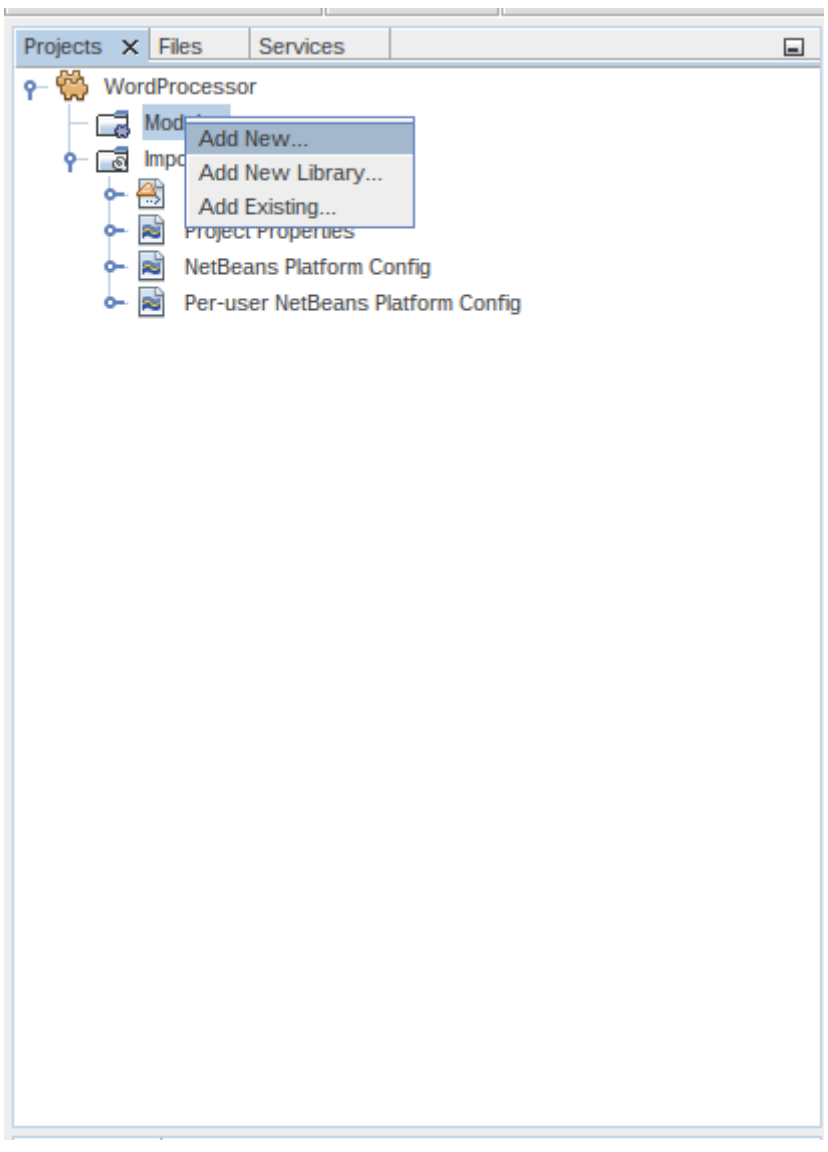
Kattinsunk a Készre, és már készül is a projektünk.



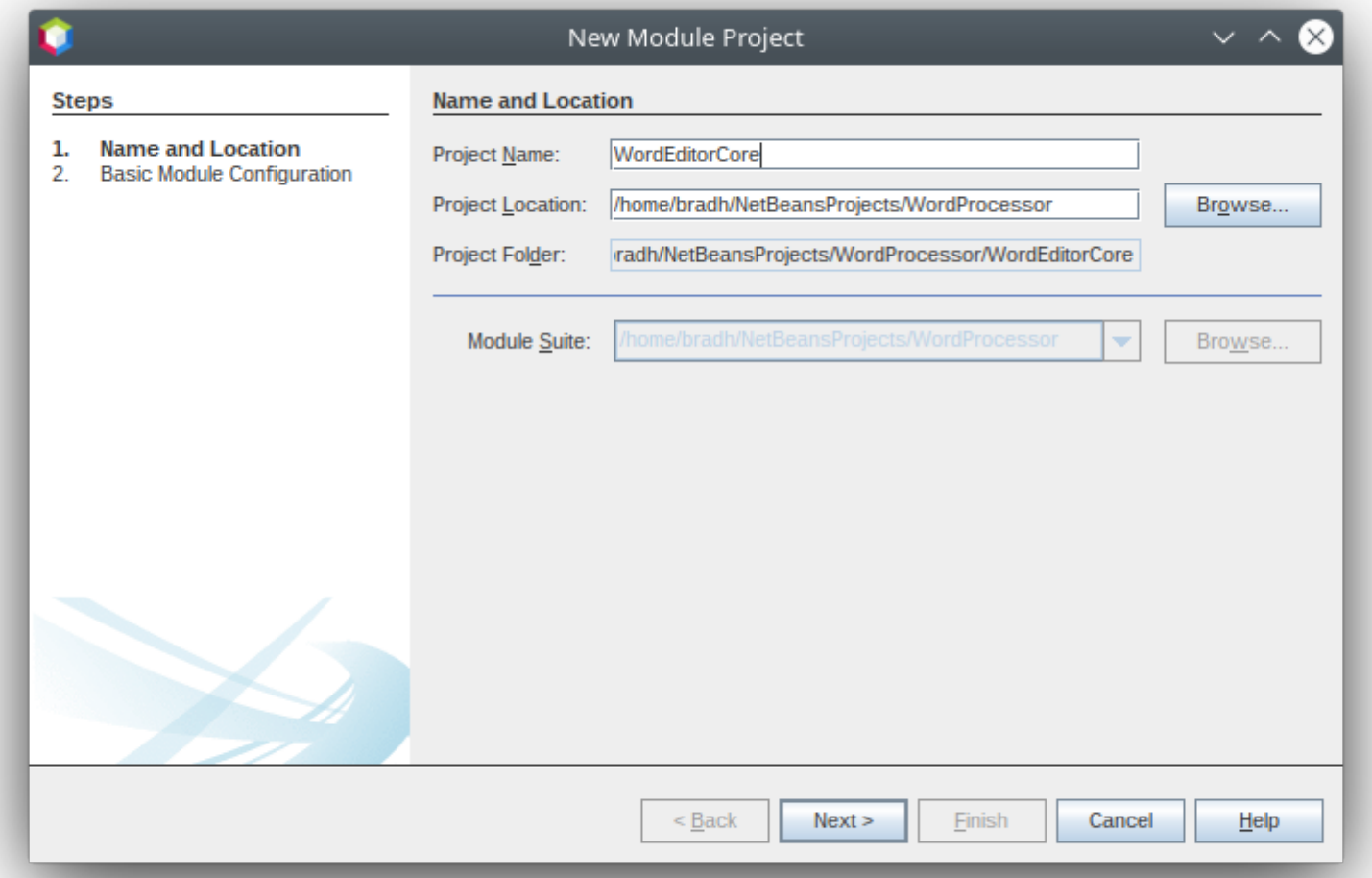
Modul létrehozása

Ebben a szakaszban hozzuk létre az első NetBeans Platform modult.

1. Kattintson jobb gombbal a fenti képernyőképen látható "Modulok"-ra, és válassza az "Új hozzáadása..." lehetőséget:

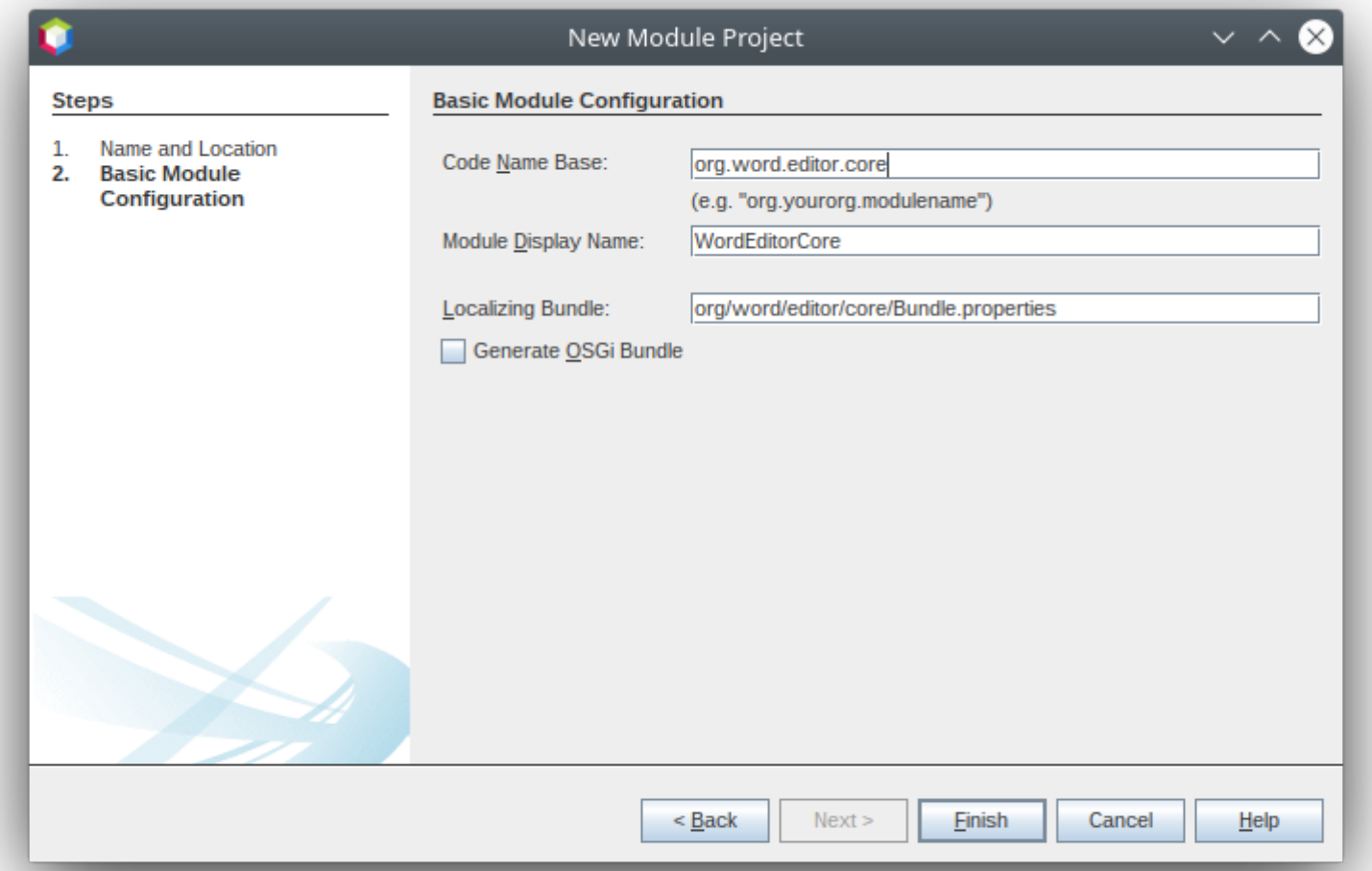


Megjelenik egy párbeszédpanel az új modulprojekthez.

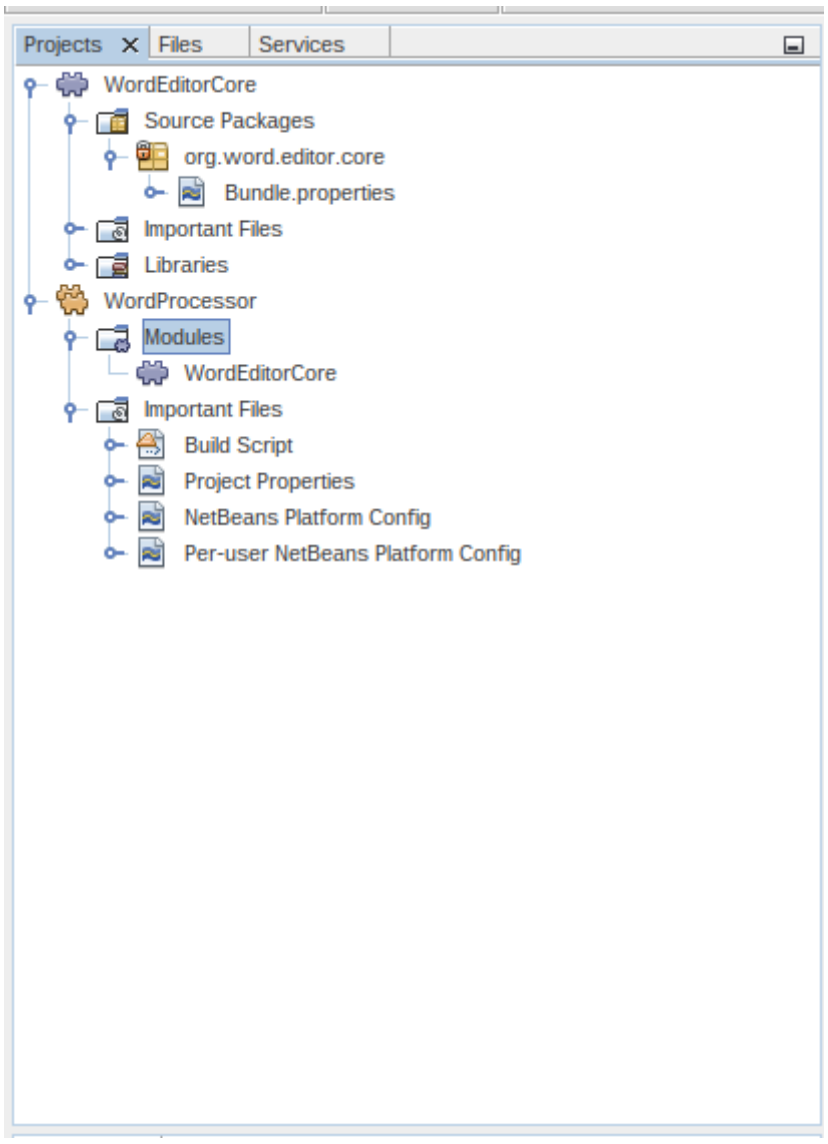


Miután elneveztók a modult kattintsunk a **következőre**.

2. Adja meg a kódnévbázist, amely a modult azonosító egyedi karakterlánc. A Modul megjelenített neve a modul címkeként szolgál a Projektek ablakban.



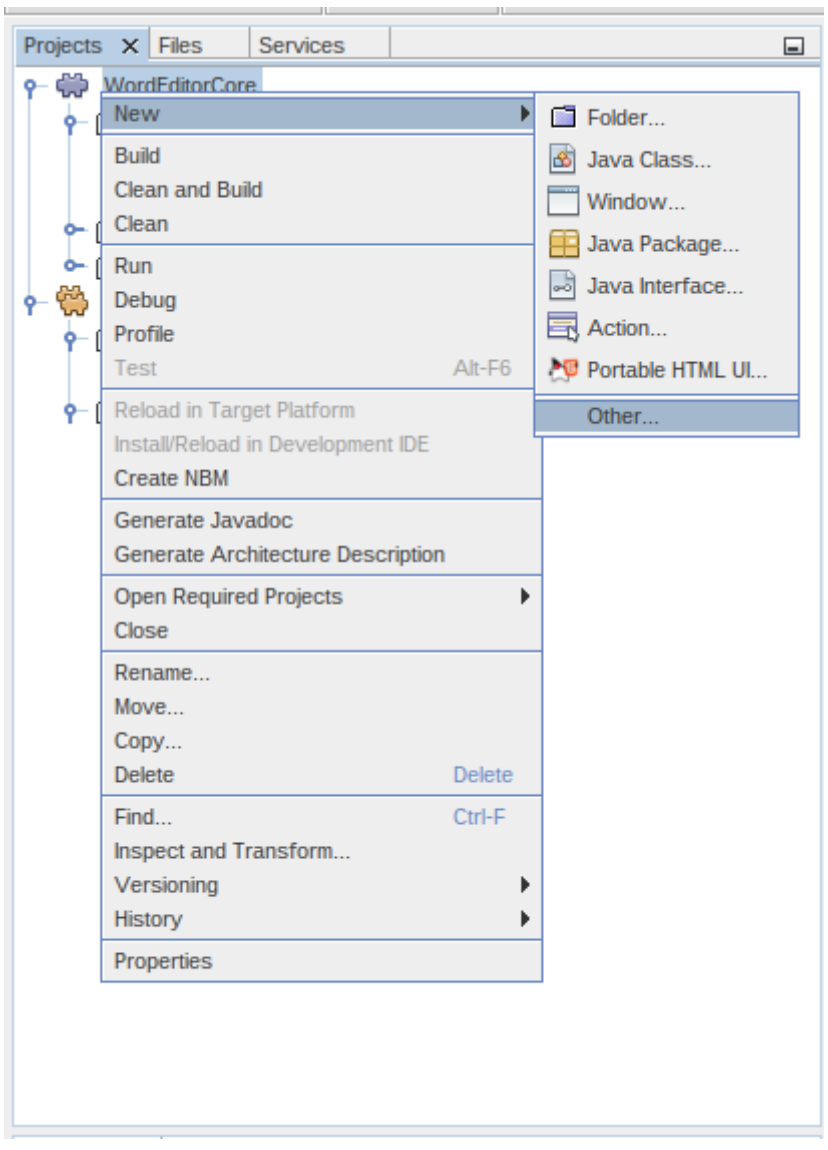
Kattinsunk a **Készre**.



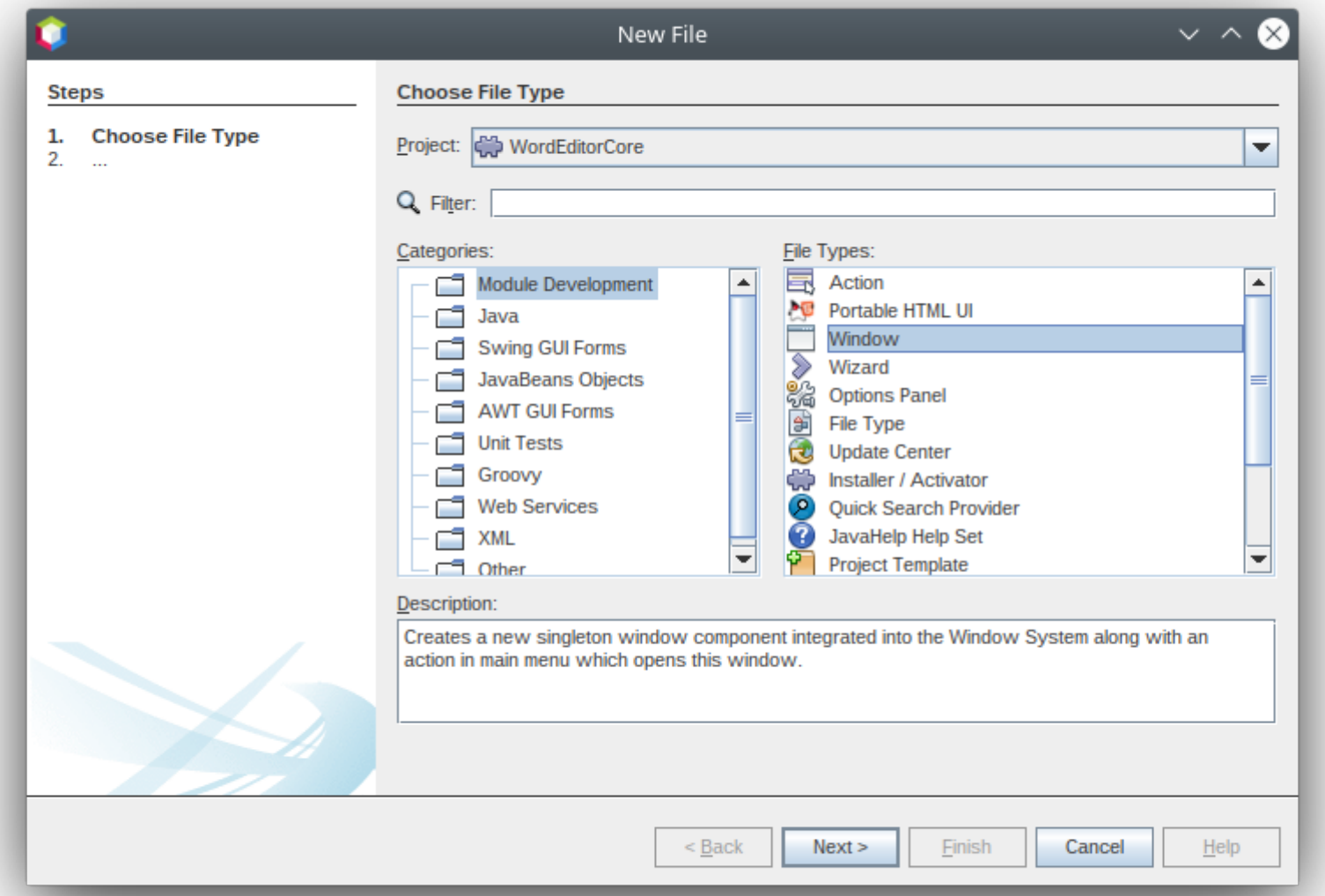
Ablak létrehozása

Létrehoztuk az első modulunkat, most hozzuk létre az első ablakot.

1. Kattintson jobb gombbal a modulra, és válassza az Új → Egyéb...



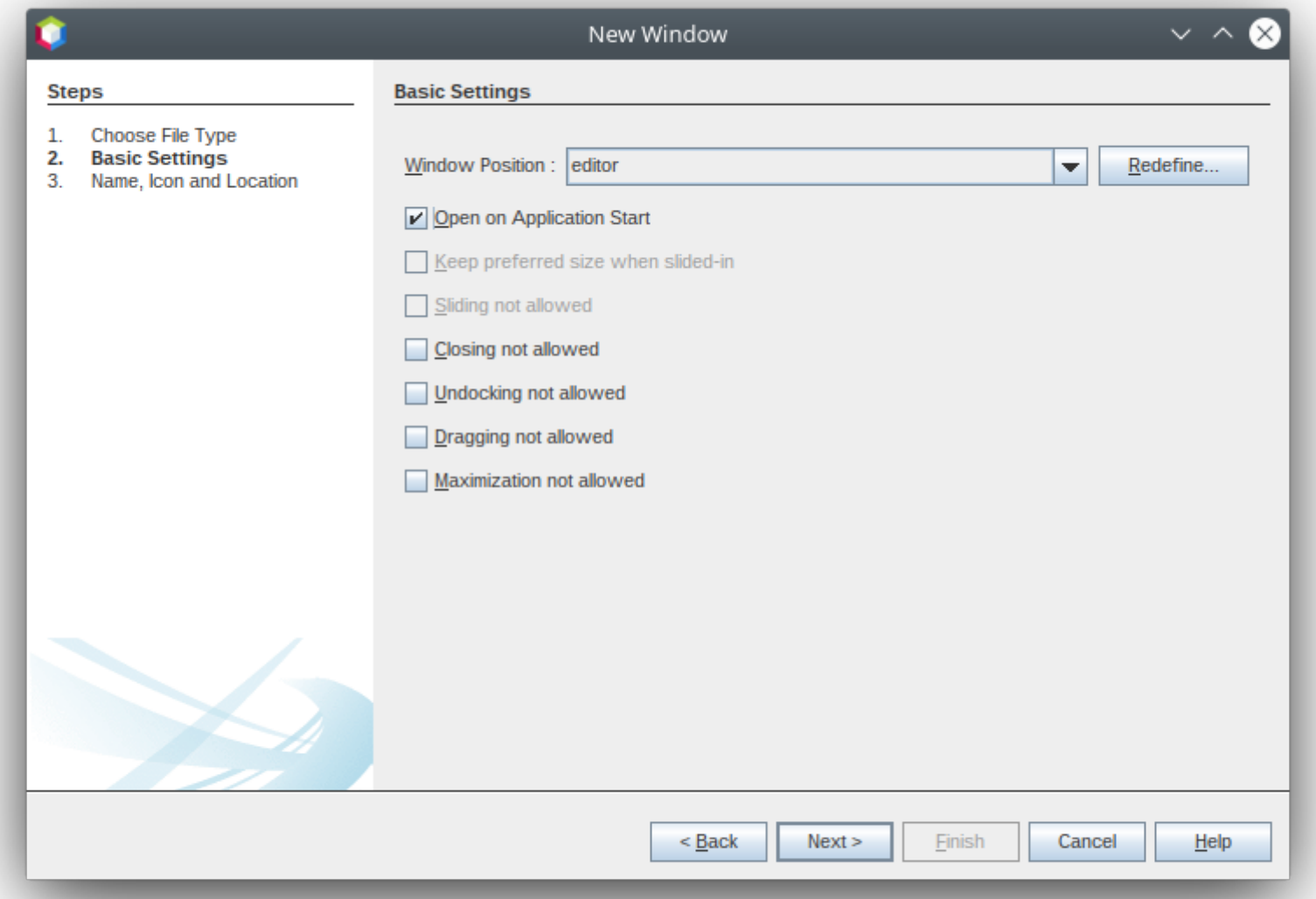
Megjelenik az Új fájl párbeszédpanel. A Modulfejlesztés kategóriában válassza az "Ablak" lehetőséget:



Kattintsunk a **következőre**.

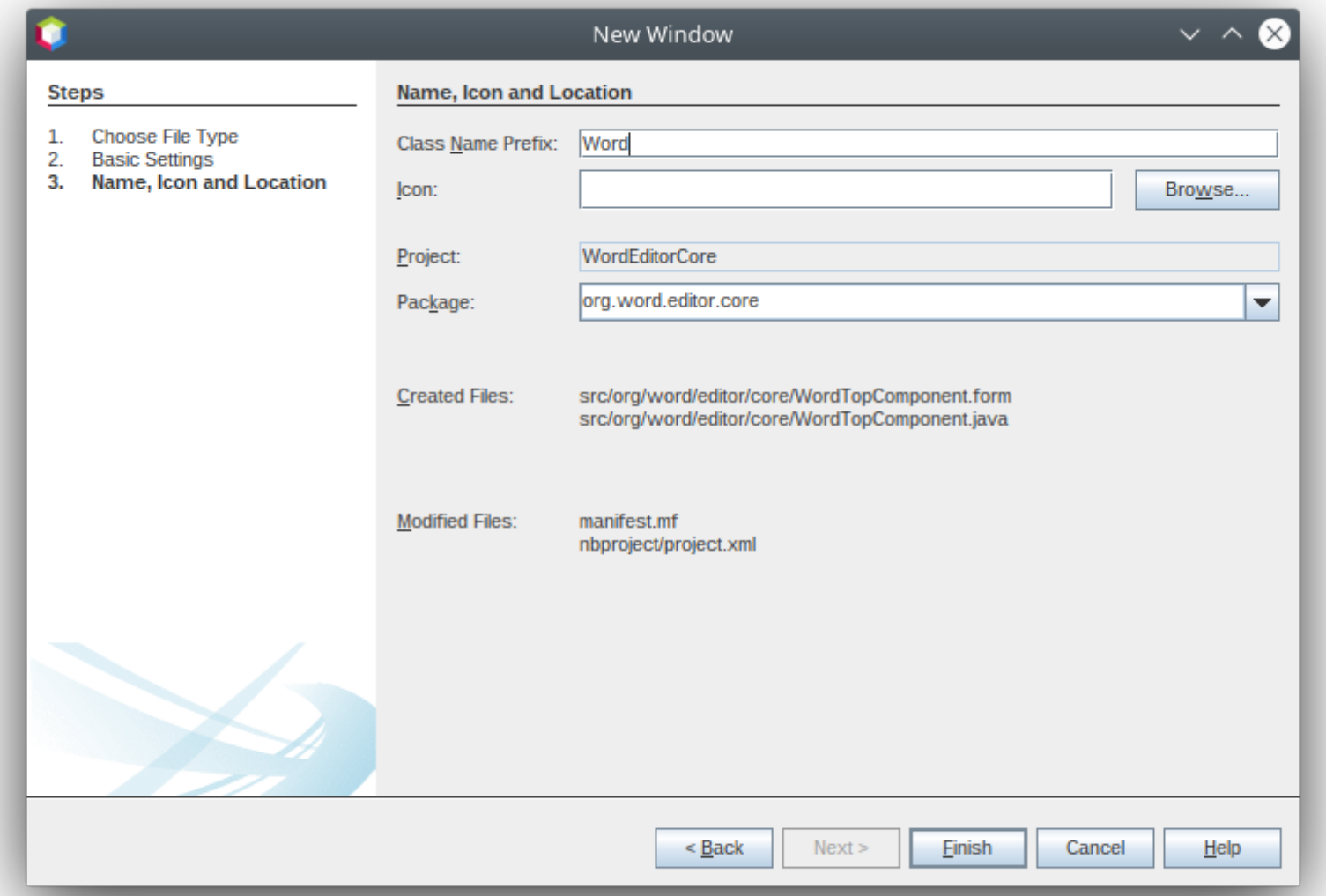
2. Most megjelenik egy párbeszédpanel, amelyben többek között megadhatja, hogy az új ablak hol jelenjen meg a képernyőn, valamint azt, hogy az alkalmazás indításakor automatikusan megnyíljon-e az ablak.

Állítsa be az Ablak pozícióját editor-nak, ami az alapértelmezett központi pozíció az ablakon belül, és válassza a "Megnyitás az alkalmazás indításakor" lehetőséget.



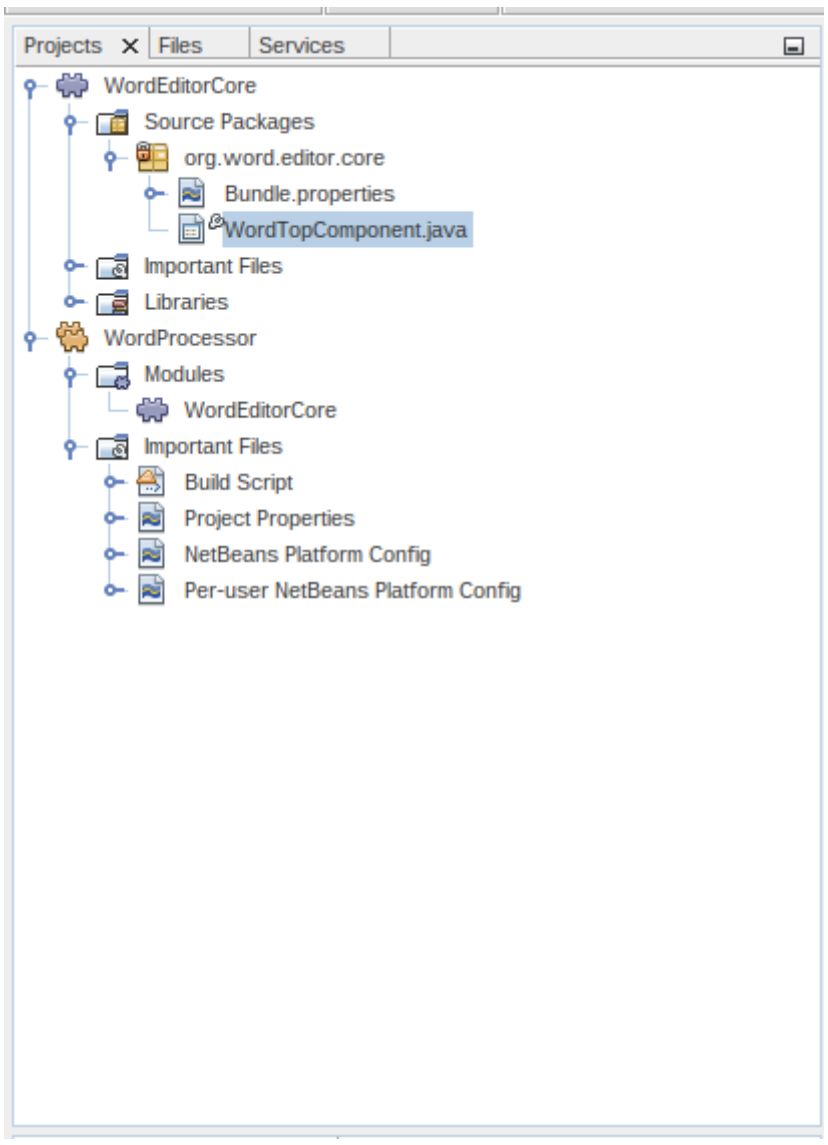
Kattintsunk a **következőre**.

3. Állítsa az osztálynév előtagját Word-re, a package nevét pedig **org.word.editor.core**-ra:

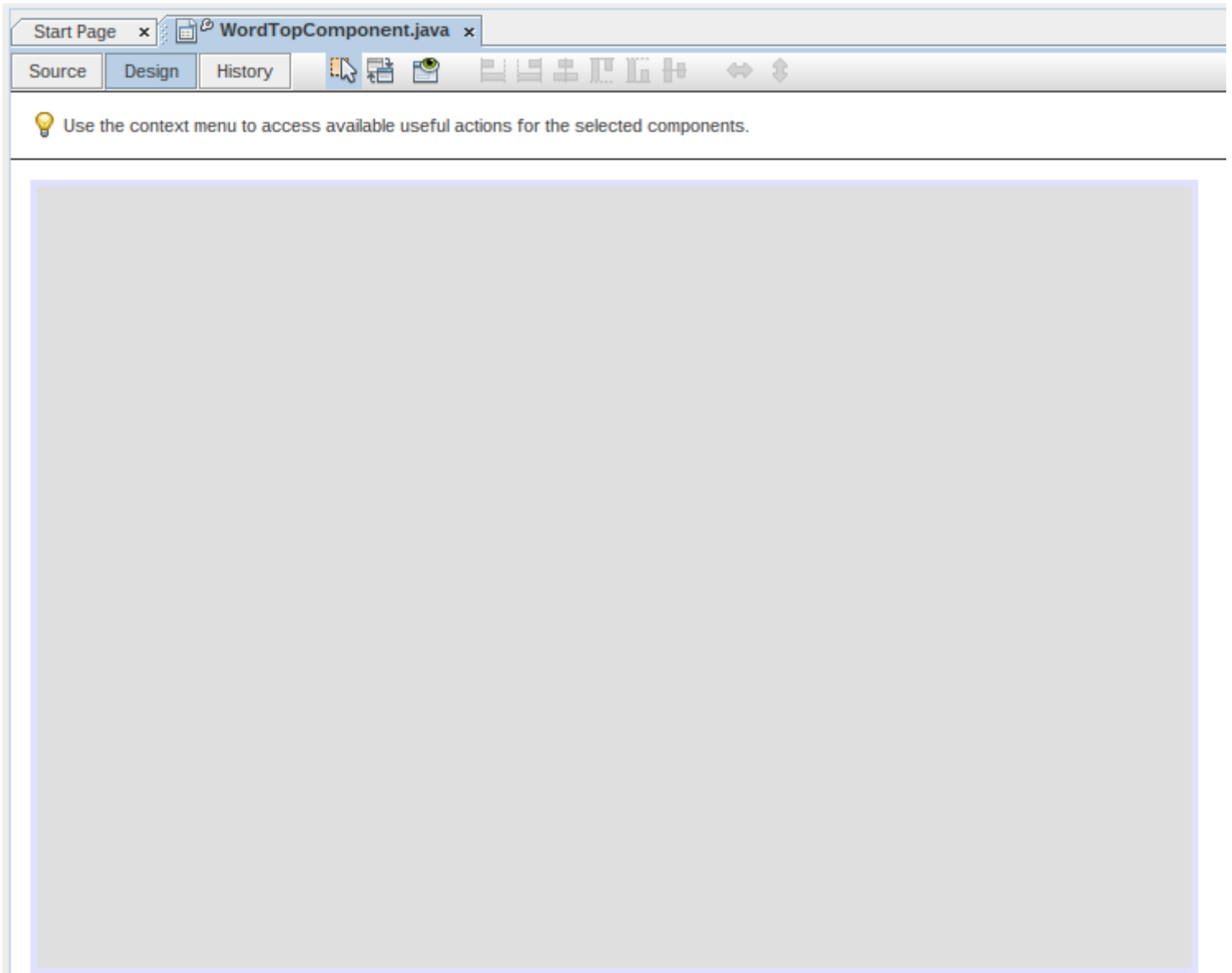


Kattintsunk a **befejezésre**.

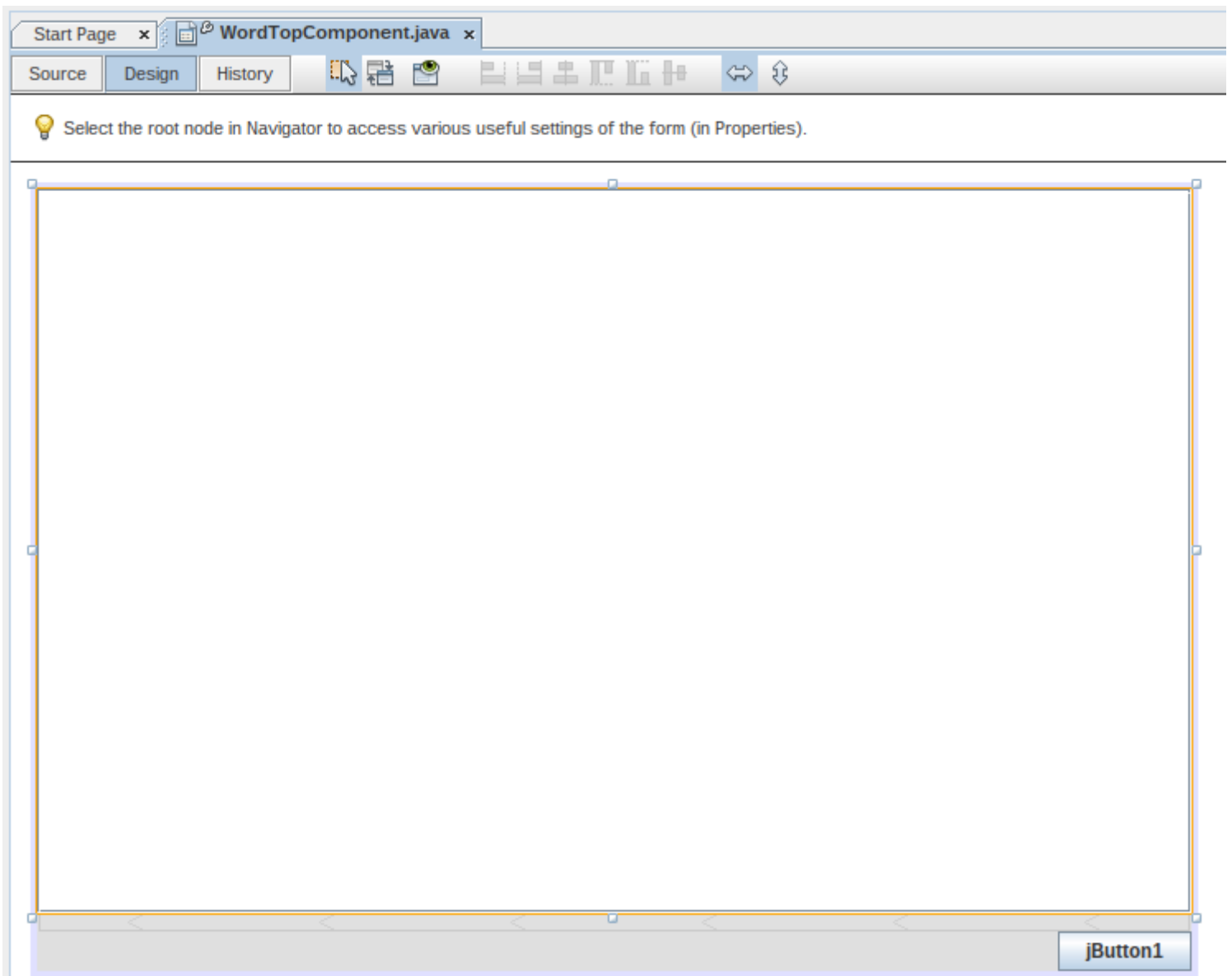
Az új ablak ("WordTopComponent.java") hozzáadódik a modul struktúrájához:



4. Az új ablaknak meg kell nyílnia a "Matisse" GUI Builder tervezési nézetében. Ha nem nyílt meg automatikusan, akkor kattintson rá duplán (vagy válassza a "Megnyitás" lehetőséget).

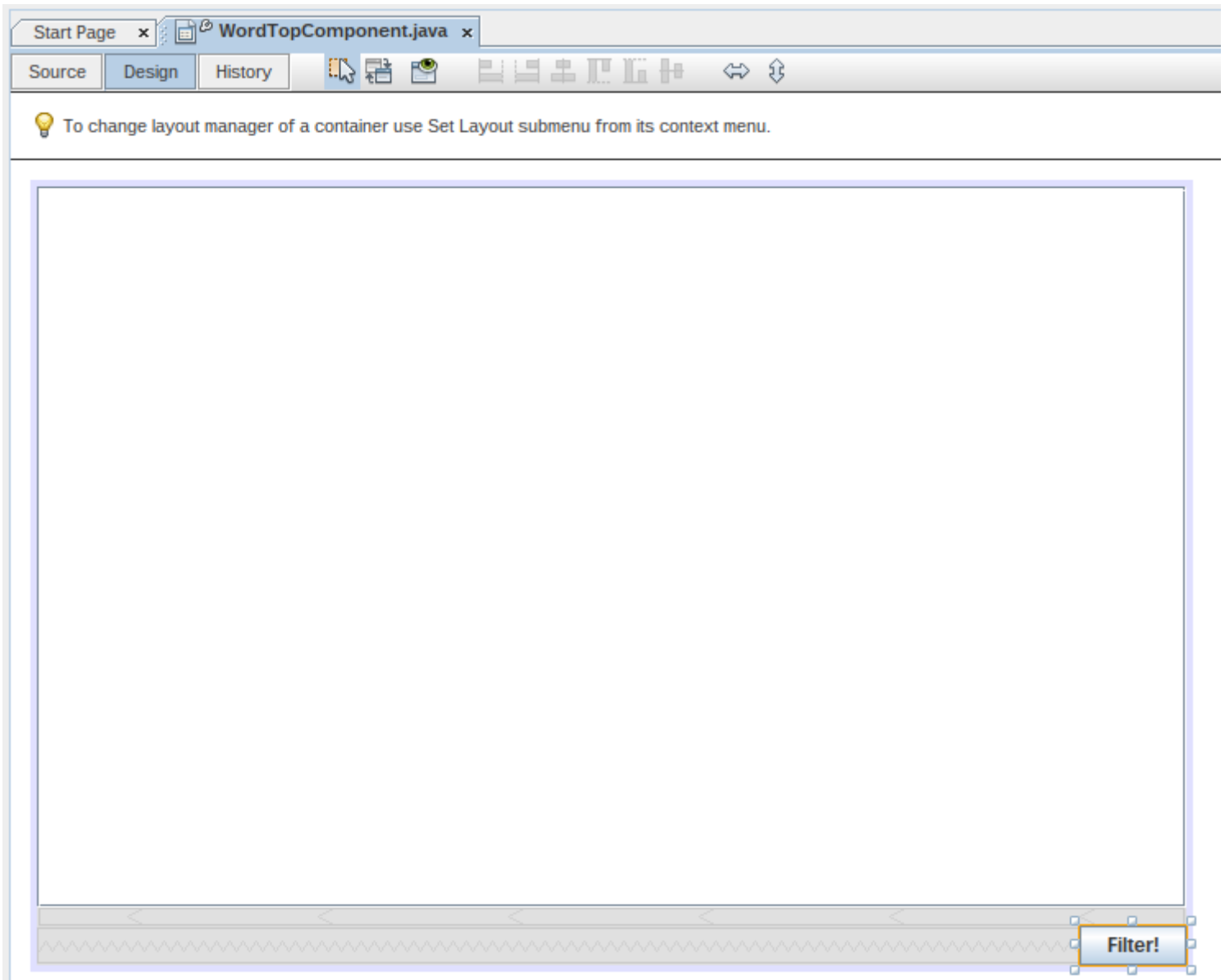


A Palettának meg kell nyílnia a jobb oldalon (megnyithatja a Window IDE Tools Palette menüelem vagy Ctrl+Shift+8 billentyűkombinációt segítségével, amennyiben nem nyílt meg). Húzzon egy gombot és egy szövegterületet a palettáról az ablakra:



Az új GUI-összetevők értelmesebbé tételéhez tegye a következőket:

- Kattintson a jobb gombbal a szövegterületre, válassza a "Változó neve módosítása" lehetőséget, majd nevezze el szövegnek.
- Kattintson a jobb egérgombbal a felhelyezett gombra, válassza a "Szöveg szerkesztése" lehetőséget, majd állítsa a gomb szövegét a *Szűrés!* értékre. Nevezze át a változót is *filterButton*-ra.



5. Kattintson duplán a gombra. Ez létrehoz egy eseménykezelési metódust a forrásszerkesztőben. A metódus minden alkalommal meghívásra kerül, amikor a gombra kattintunk. Módosítsa a metódus szövegét a következő kódra:

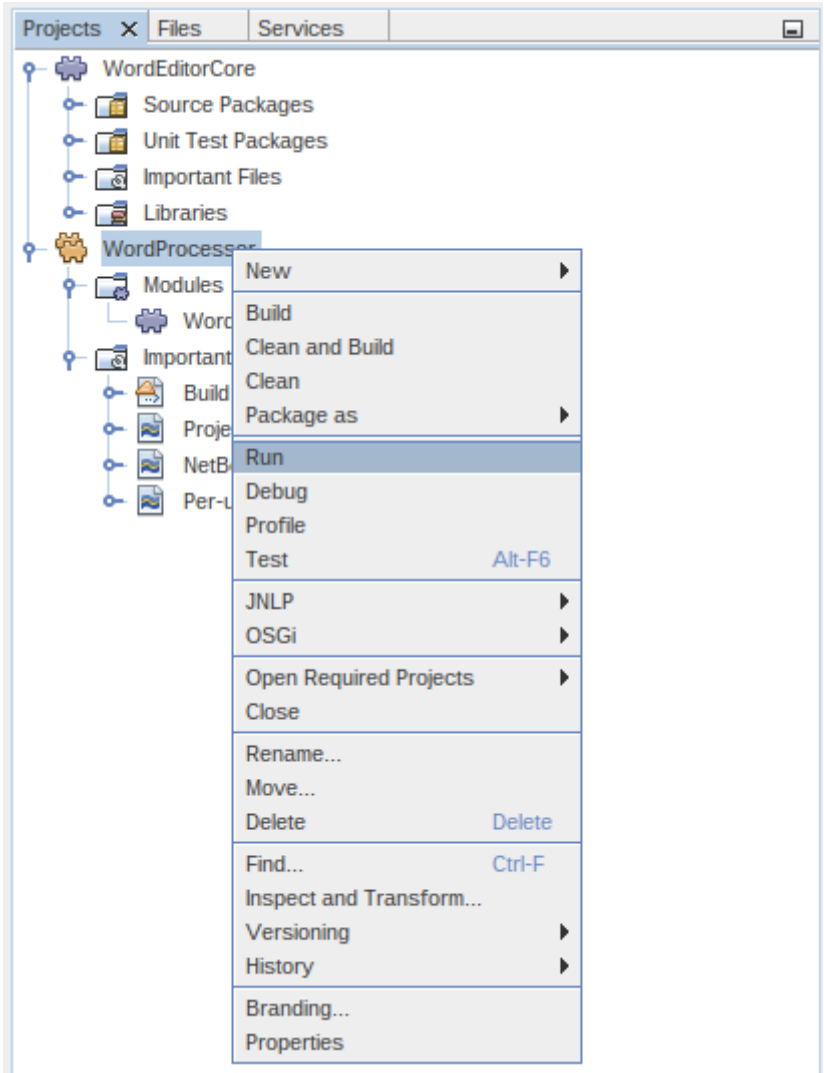
```
private void filterButtonActionPerformed(java.awt.event.ActionEvent evt) {
    String s = text.getText();
    s = s.toUpperCase();
    text.setText(s);
}
```

Ezzel létrehozta az ablakmodult. A "Filter!" gombra való kattintáskor a filterButtonActionPerformed metódus meghívásra kerül, amely megkapja a szövegterület tartalmát, átalakítja a szöveget nagybetűsre, és a nagybetűs változatot helyezi a szövegterületbe.

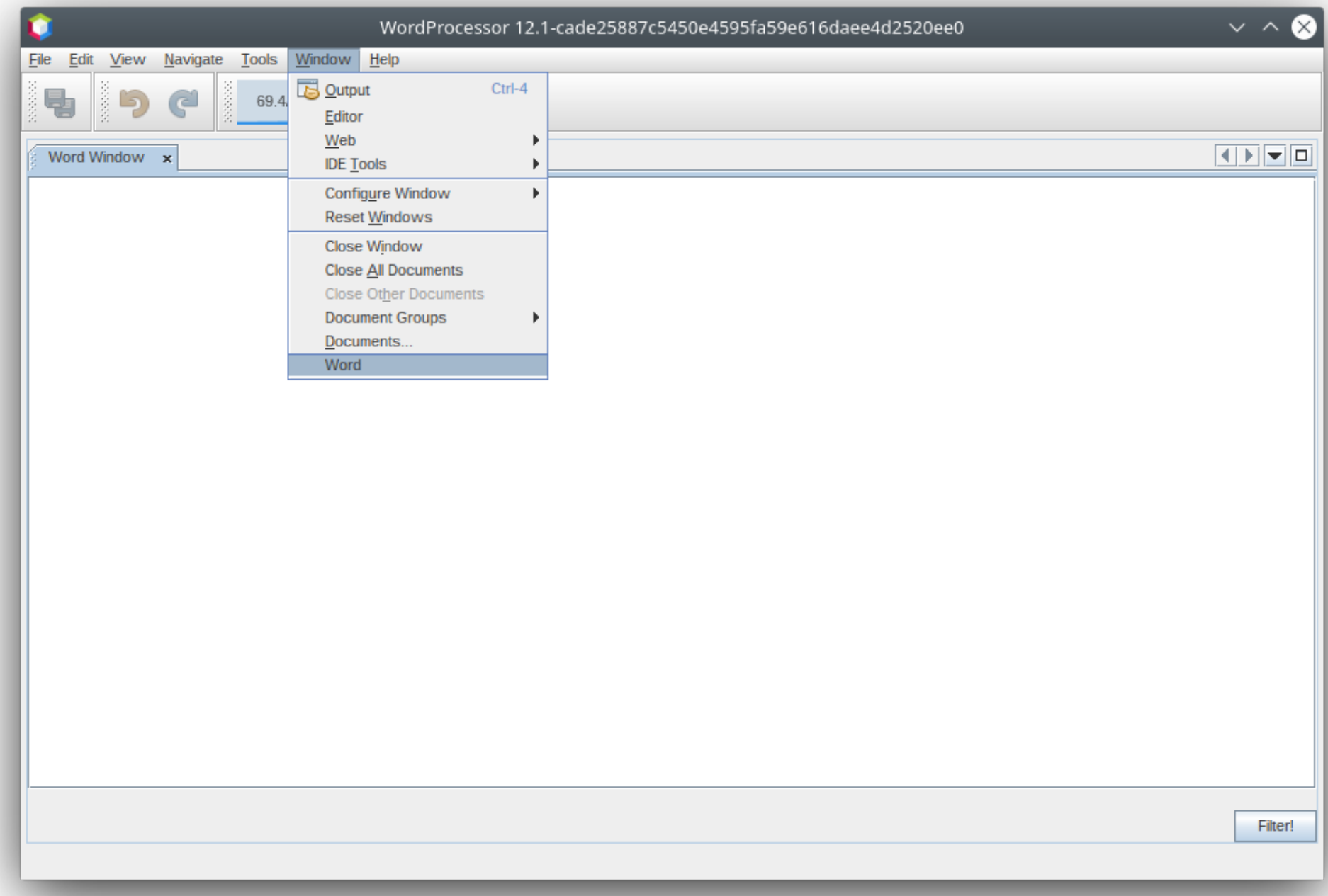
Alkamazás futtatása

Ebben a szakaszban lefuttatjuk az alkalmazást.

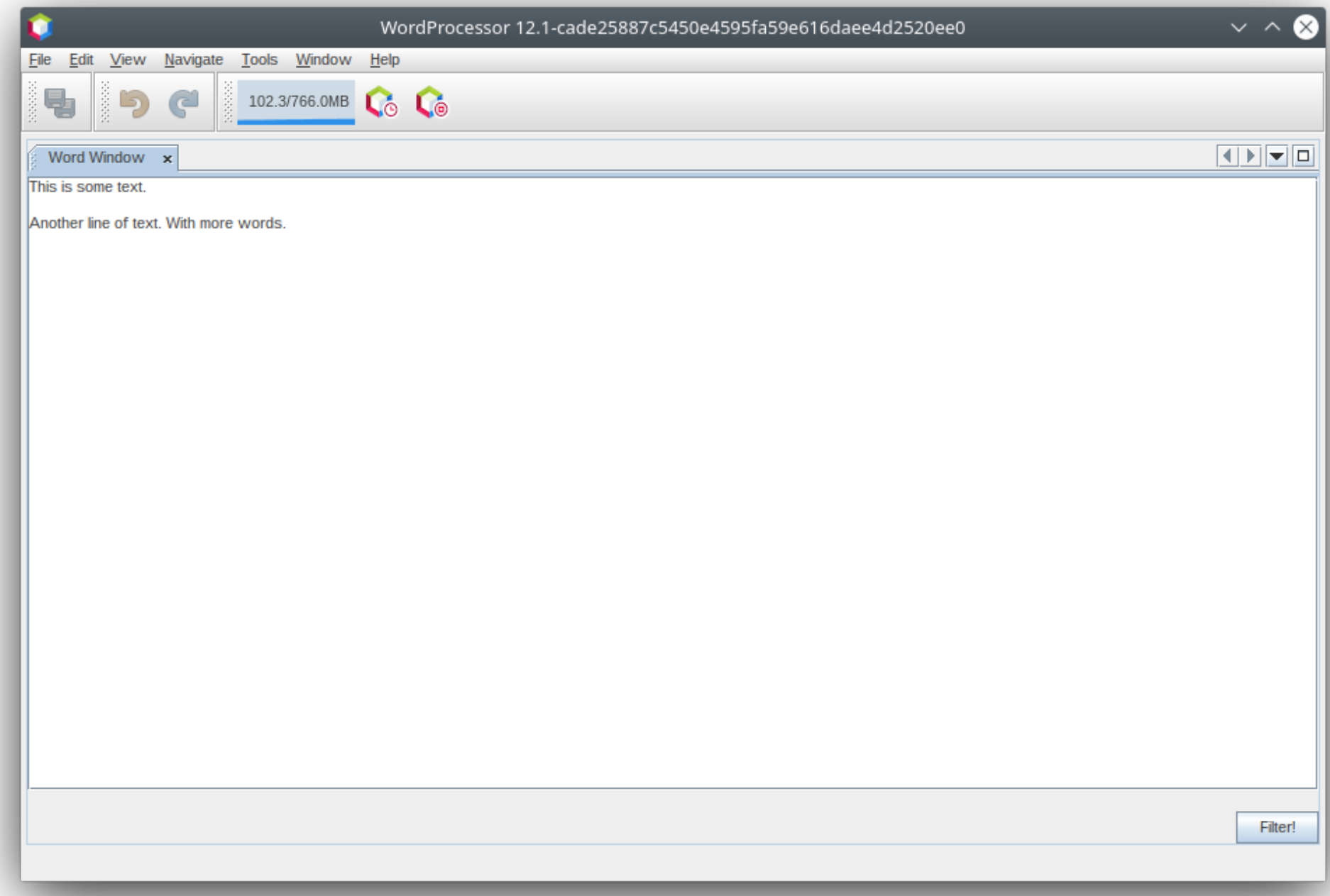
1. Kattintson a jobb gombbal a WordProcessor alkalmazásra (nem a WordEditorCore modulra), és válassza a Futtatás parancsot.



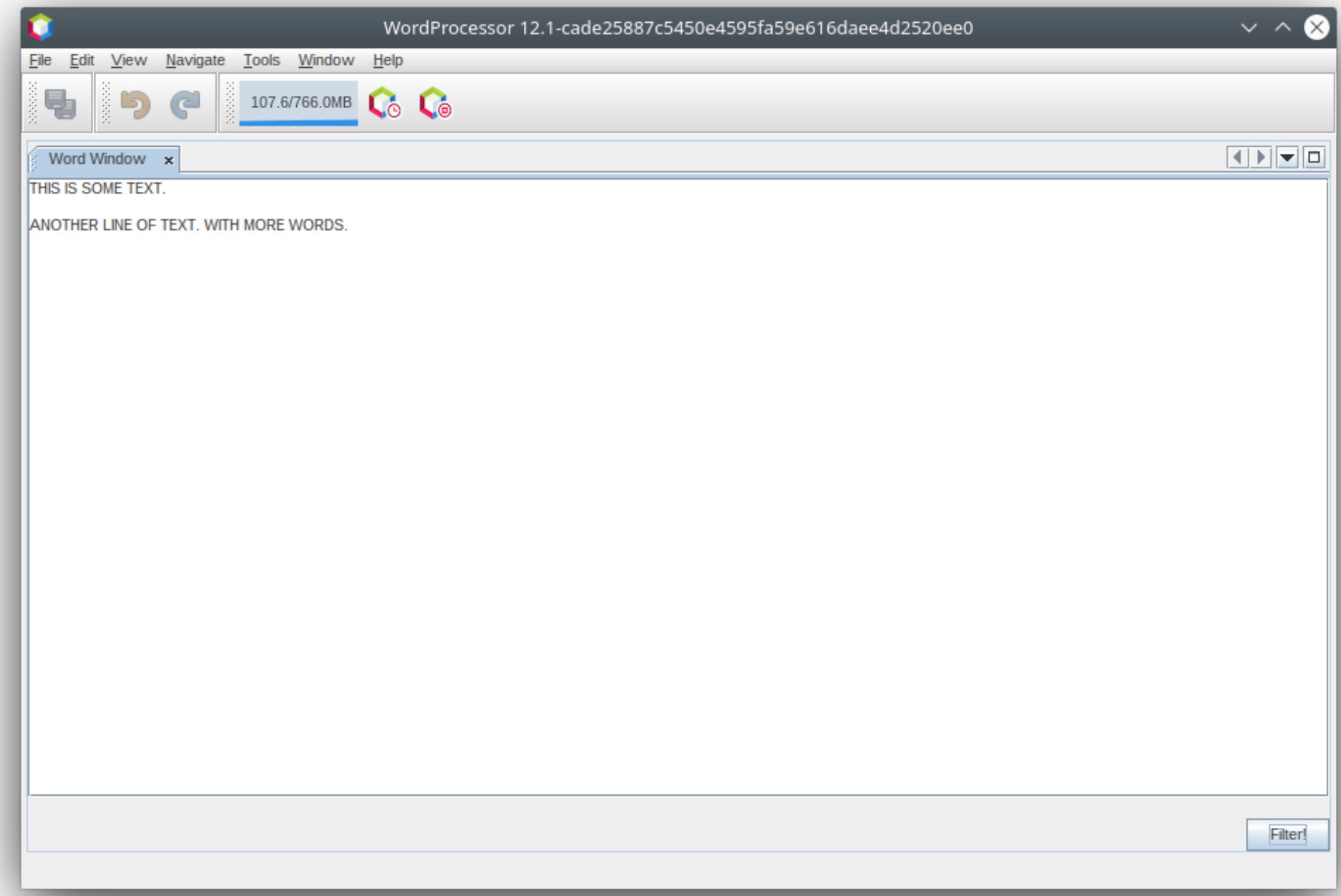
Ezzel elindítja az új NetBeans Platform alkalmazást és telepíti a modult. Lesz egy új ablakod, valamint egy új menüpont a megnyitásához, ahogy az alábbiakban látható:



2. Írja be a szöveget kisbetűvel a szövegmezőbe, és kattintson a "Szűrés!" gombra.



Látható, hogy a szöveg most már nagybetűvel jelenik meg:



Megtanulta, hogyan hozhat létre egy új Apache NetBeans Platform alkalmazást, és hogyan adhat hozzá új modulokat. A következő részben megismerkedik az Apache NetBeans Platform pluggable service infrastruktúrájával.

Moduláris alkalmazás a Lookup segítségével

Ebben a szakaszban két további modult hoz létre. Az első új modul, a "WordEditorAPI" egy szolgáltatói interfészt tartalmaz. A második modul, az "UppercaseFilter", az interfész szolgáltatója.

Az előző szakaszban létrehozott GUI modul lazán kapcsolódik az "UppercaseFilter" szolgáltatóhoz, mivel a GUI modul nem fog hivatkozni az "UppercaseFilter" szolgáltató kódjára. Ez azért lesz lehetséges, mert az "UppercaseFilter" szolgáltató a META-INF/services mappában lesz regisztrálva és a NetBeans Lookup osztályon keresztül töltődik be, amely a JDK 6 ServiceLoader osztályhoz hasonló.

Ezután létrehoz egy másik lazán kapcsolt szolgáltatót, amelynek a neve "LowercaseFilter".

A koncepció lényege, hogy ahelyett, hogy az összes különböző funkciót hozzá kellene adni a "WordEditorCore"-hoz, minden egyes funkciót külön-külön lehet megvalósítani, anélkül, hogy a szűrési műveletet az eredmény megjelenítéséhez kötnénk.

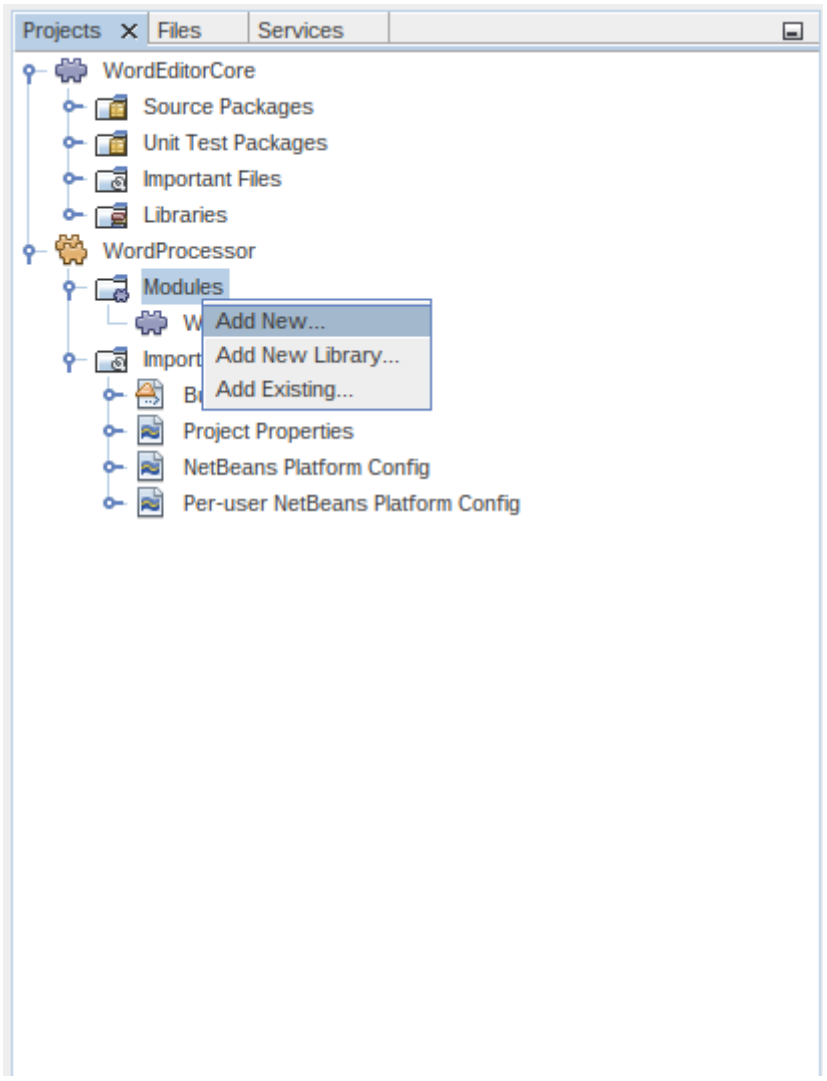
A lépések a következők:

- Az API létrehozása
- Az API megvalósítása
- Az alkalmazás futtatása

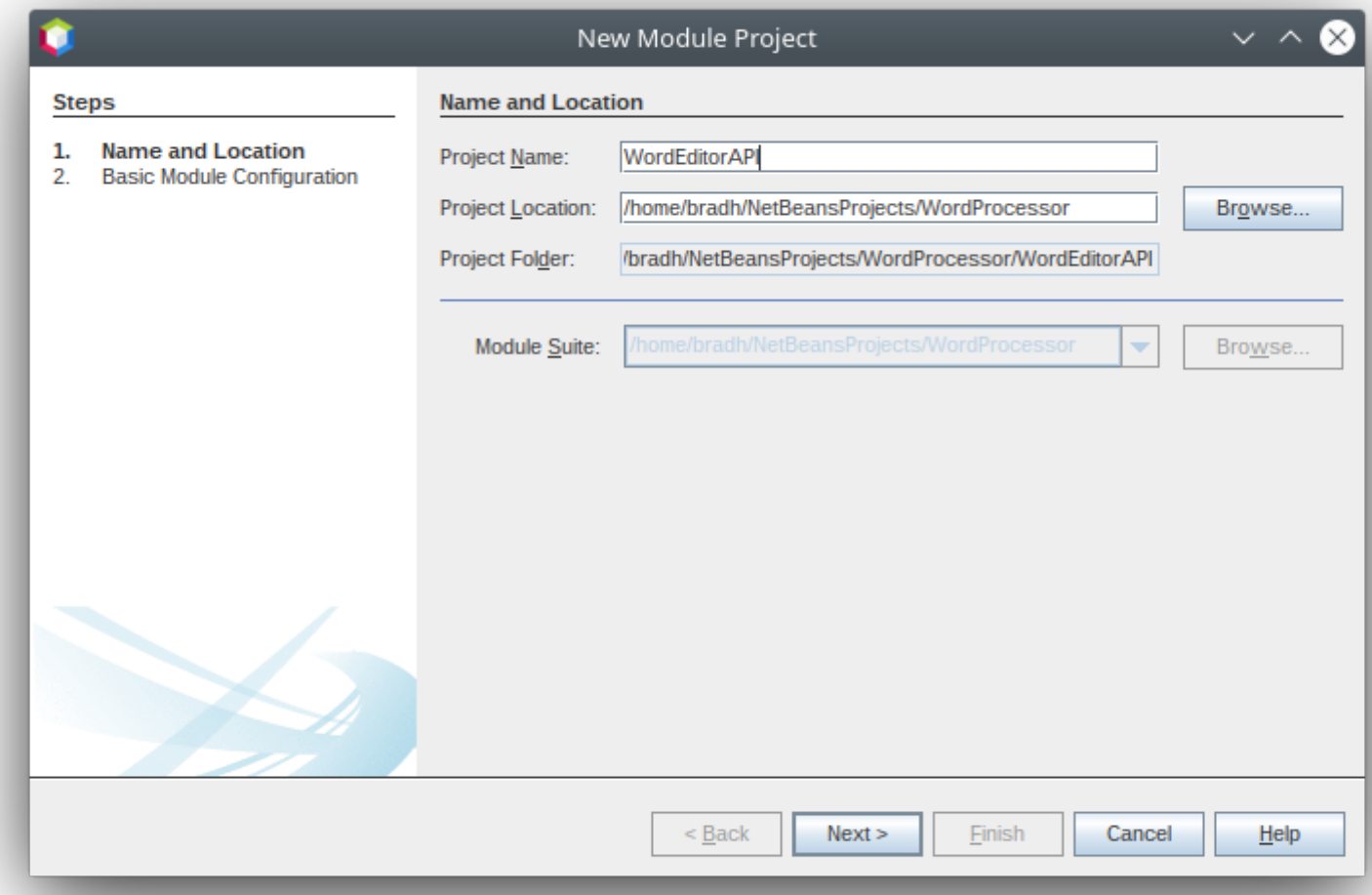
Az API létrehozása

Ebben a szakaszban létrehoz egy API-t.

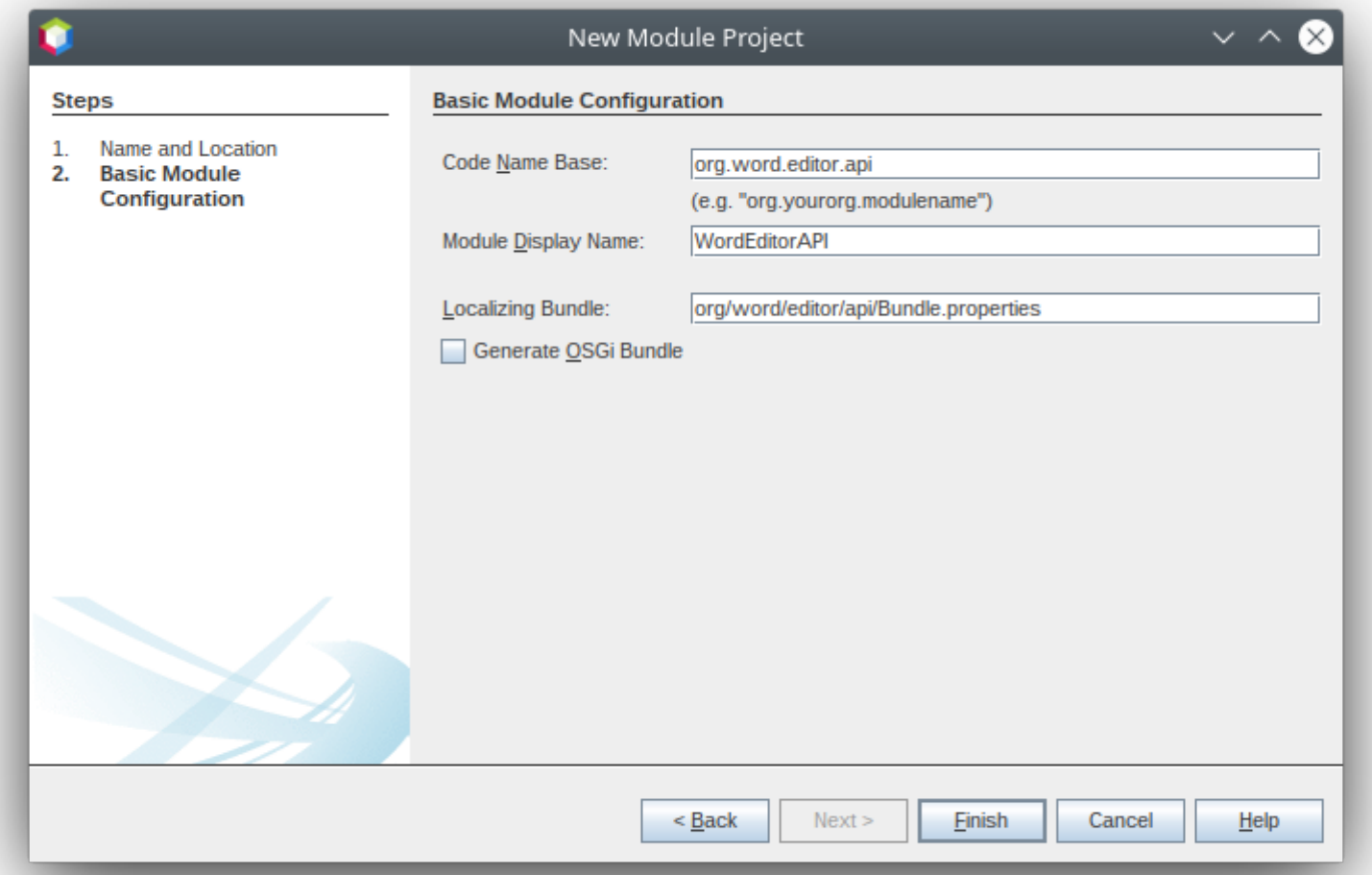
1. Kattintson az új alkalmazásra a Projektek ablakban, ezután kattintson a jobb gombbal a Modulok csomópontra, és válassza az "Új hozzáadása" lehetőséget:



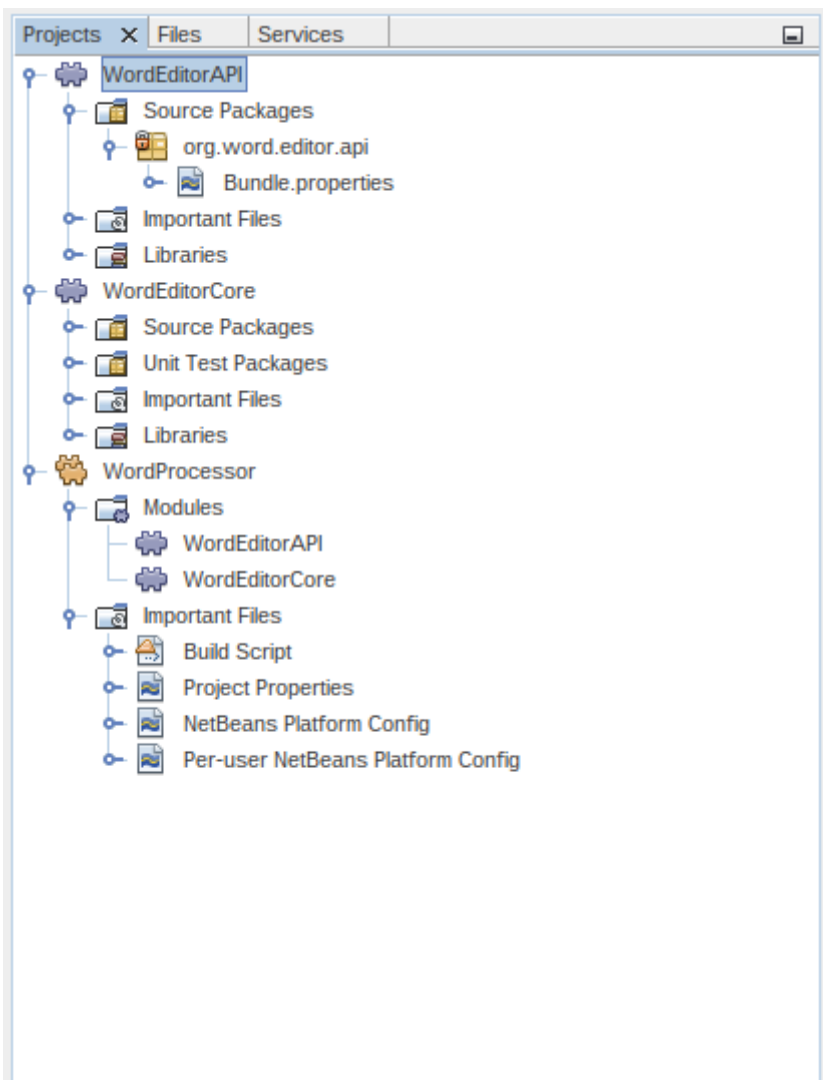
Megjelenik az Új modulprojekt párbeszédpanel. Állítsa be az új modul projektnevének a "WordEditorAPI" nevet:



Kattintson a Tovább gombra. Állítsa be a kódnév alapjának az org.word.editor.api-t, az alábbiakban látható módon:

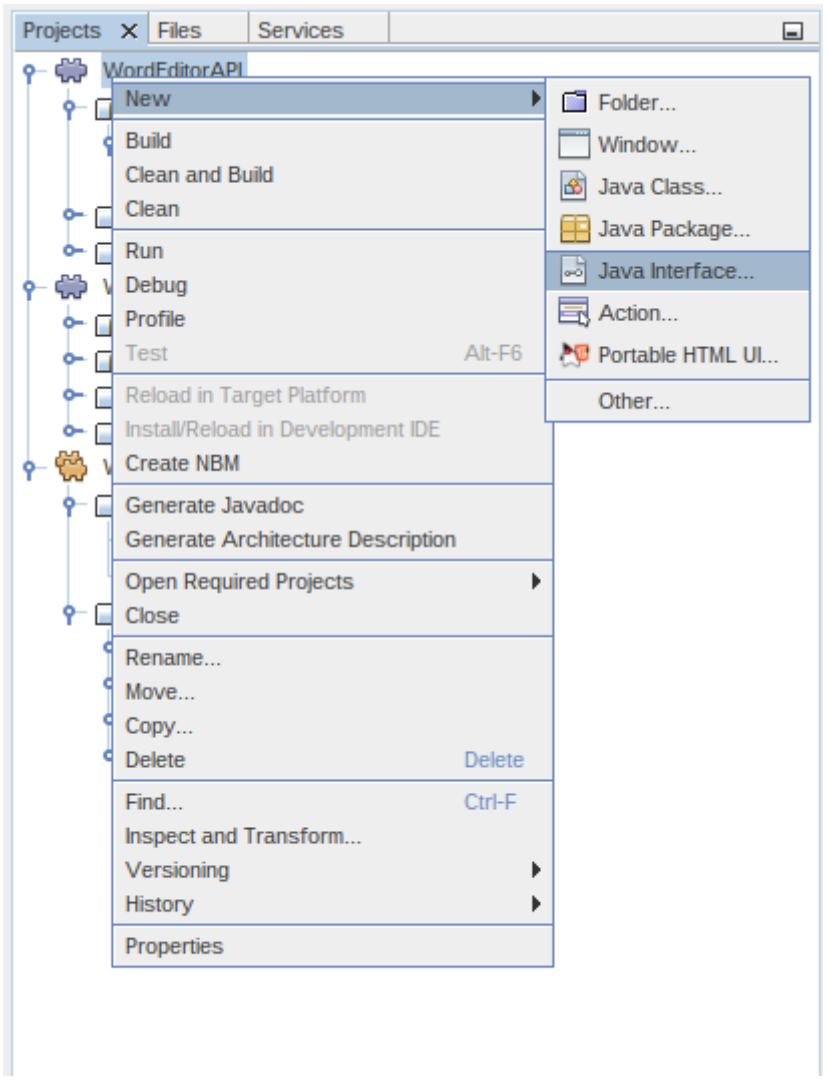


Kattintson a Befejezés gombra a varázsló befejezéséhez, amely hozzáadja a modult a korábban létrehozott alkalmazáshoz, akárcsak az előző szakaszban:

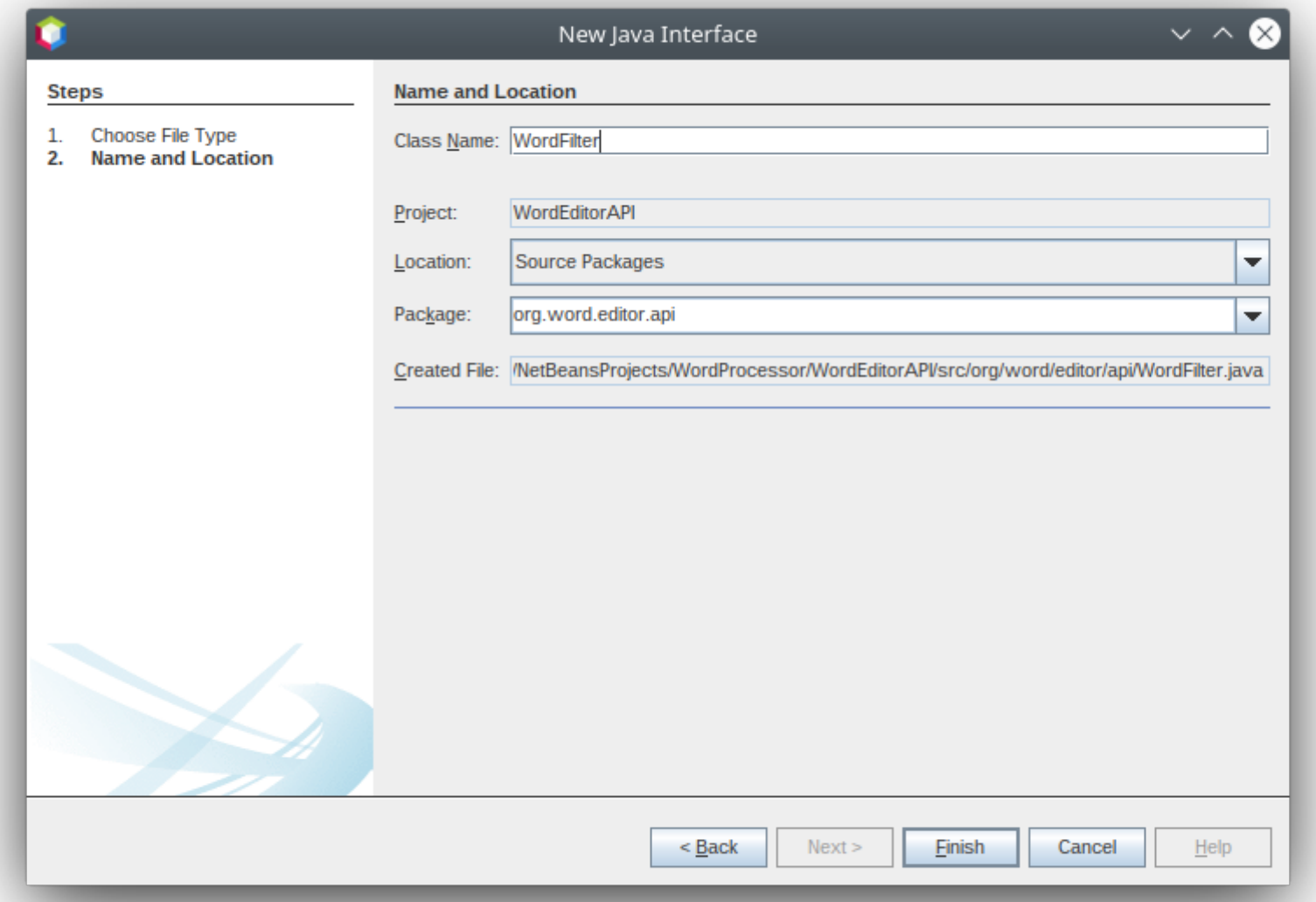


Miután létrehoztuk a modult, a következő lépés egy interfész hozzáadása.

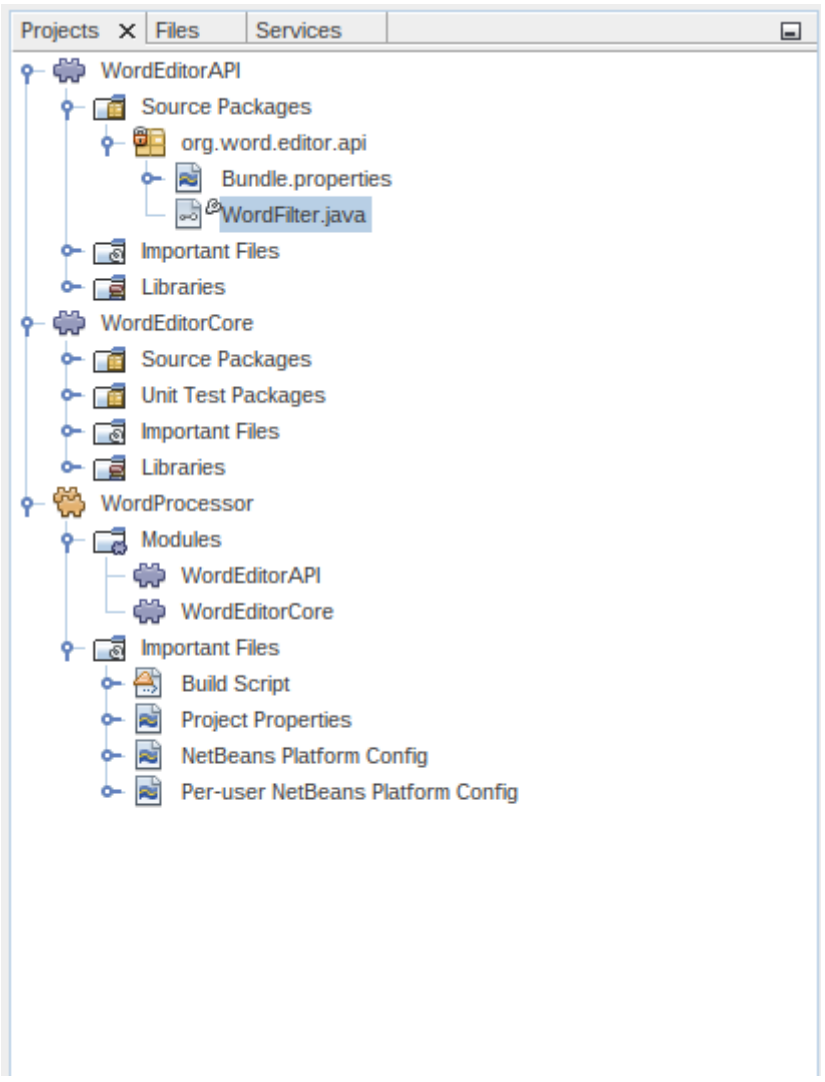
2. Kattintson a jobb gombbal a "WordEditorAPI" modulra, és válassza az Új | Java interfész lehetőséget.



Nevezze a Java-interfészt WordFilter-nek, az org.word.editor.api csomagban:



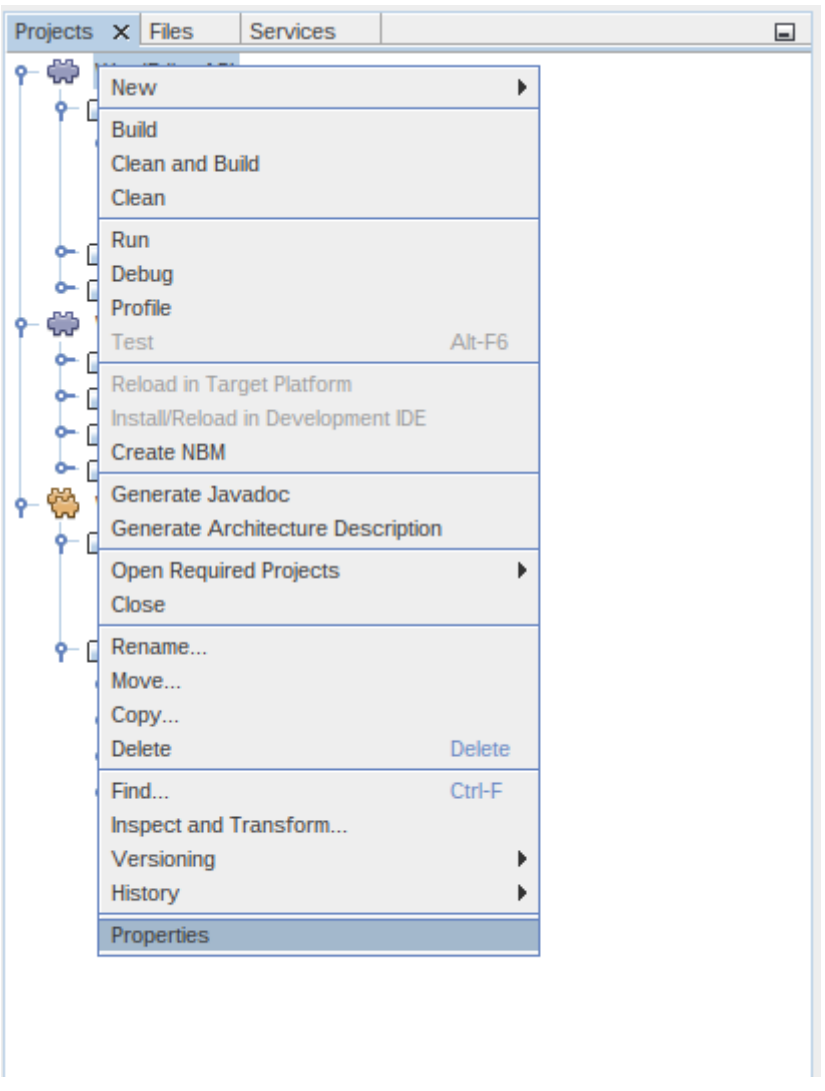
Kattintson a Befejezés gombra a varázsló befejezéséhez, amely hozzáadja az interfészt a modulhoz.



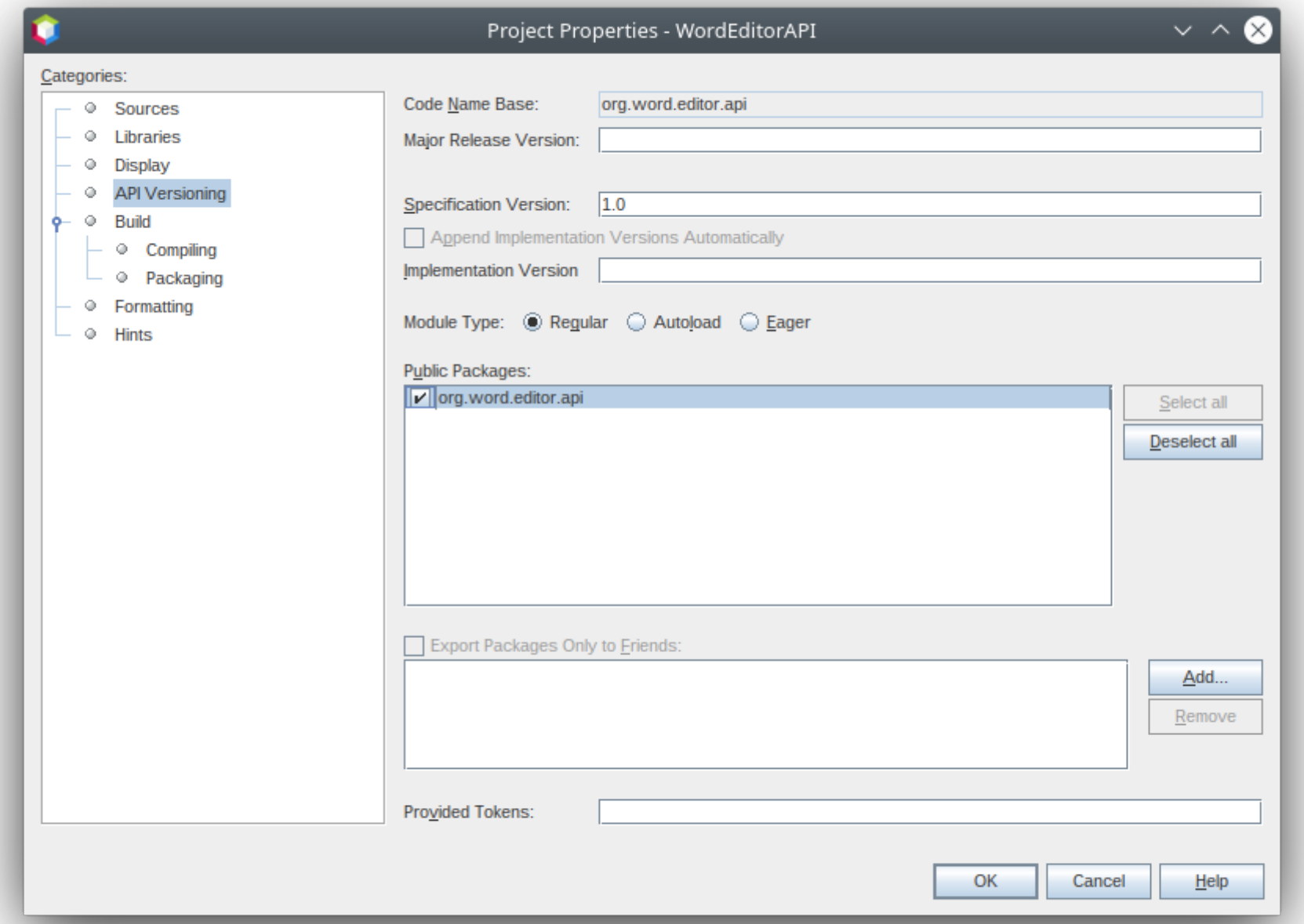
3. A WordFilter.java interfésznek meg kell nyílnia. A szerkesztő segítségével definiáljuk a következőképpen:

```
package org.word.editor.api;

public interface WordFilter {

    String process(String s);

}
```

4. Kattintson a jobb gombbal a "WordEditorAPI" modulra, majd válassza a Tulajdonságok lehetőséget a Projekt tulajdonságai ablak megnyitásához.



Válassza ki az "API verziószámozás" kategóriát, és jelölje be a "Nyilvános csomagok" alatti négyzetet, hogy az interfészt tartalmazó csomag az egész alkalmazásban elérhető legyen:



Kattintson az OK gombra.

Másik lehetőségként a Projektek ablakban bontsa ki a "WordEditorAPI" projektben a "Fontos fájlok" menüpontot, majd kattintson duplán a "Projekt metaadatok" menüpontra.

A "project.xml" fájl megnyílik, és látnia kell, hogy a csomagot nyilvánossá nyilvánították:

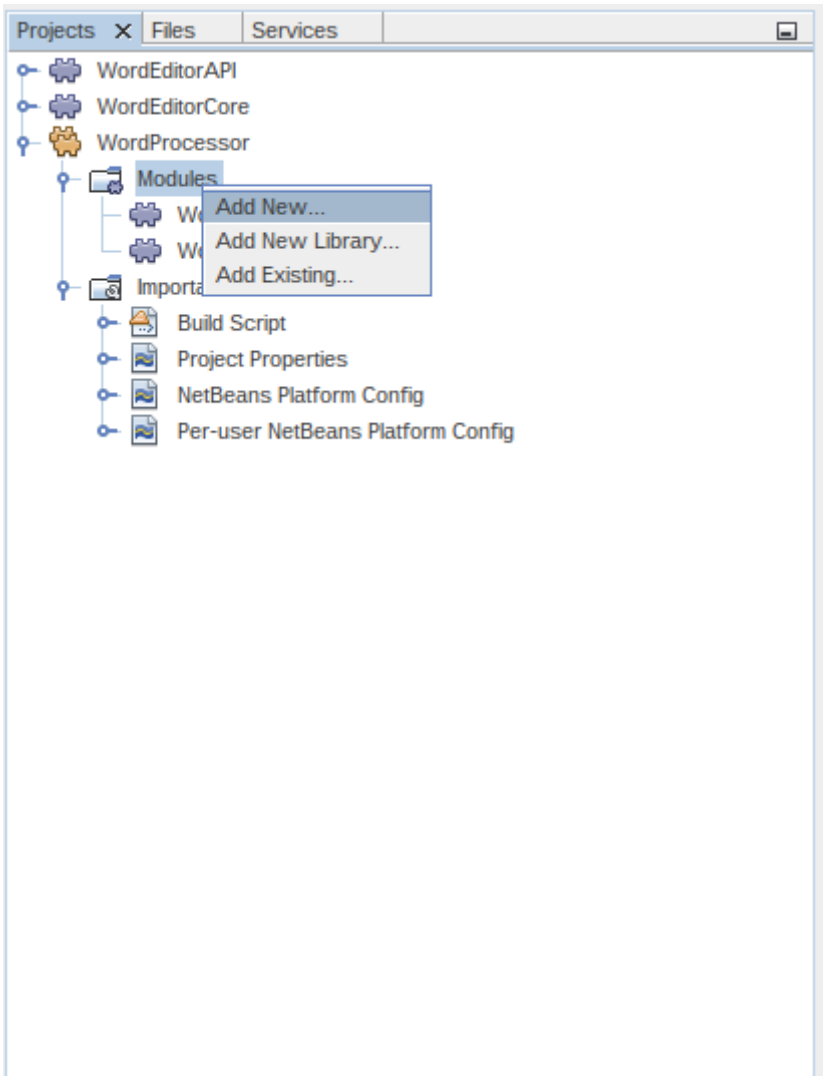
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://www.netbeans.org/ns/project/1">
    <type>org.netbeans.modules.apisupport.project</type>
    <configuration>
        <data xmlns="http://www.netbeans.org/ns/nb-module-project/3">
            <code-name-base>org.word.editor.api</code-name-base>
            <suite-component/>
            <module-dependencies/>
            <public-packages>
                <package>org.word.editor.api</package>
            </public-packages>
        </data>
    </configuration>
</project>
```

Az API definíciója most már teljes.

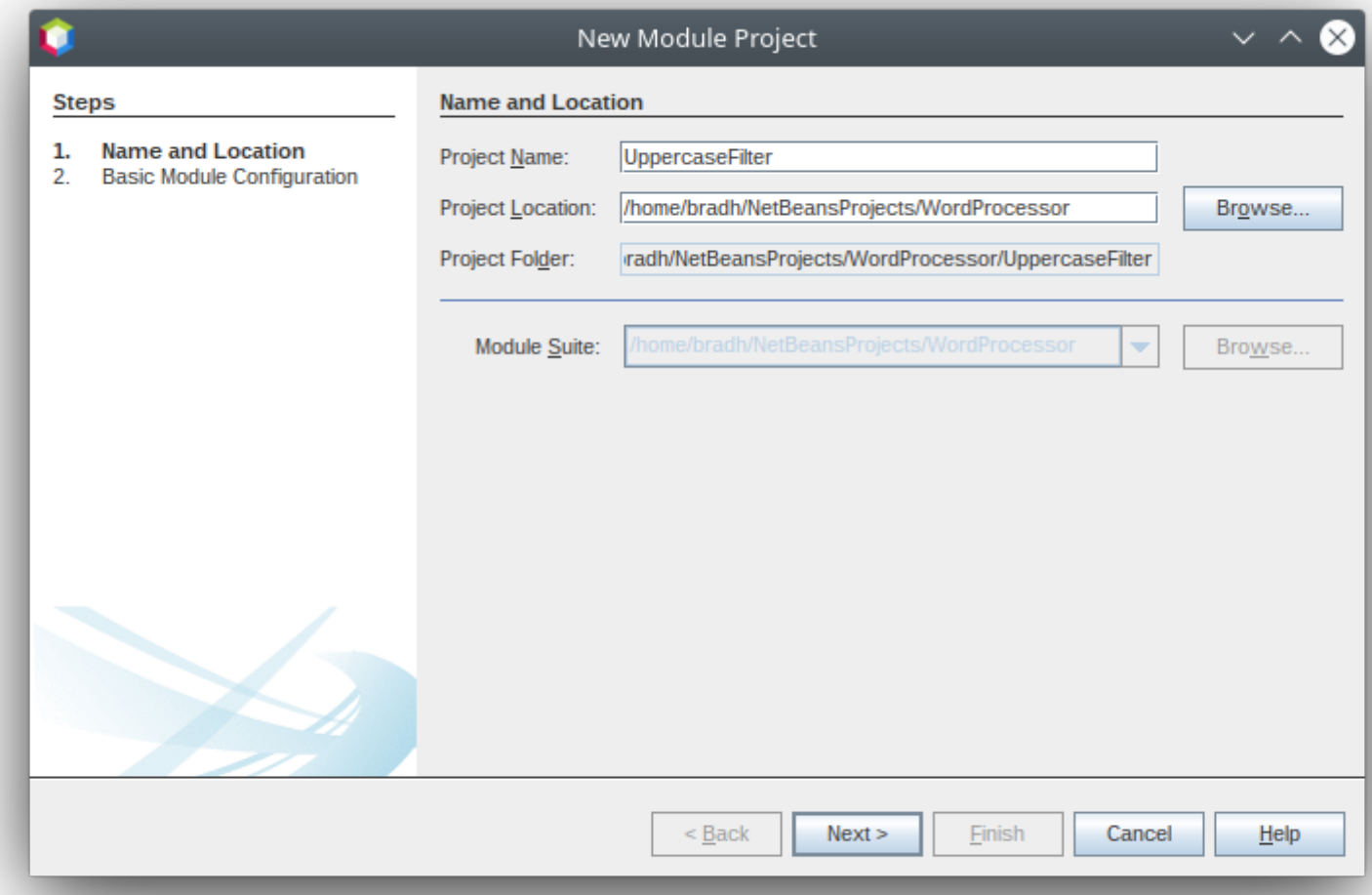
Az API megvalósítása

Ebben a szakaszban az imént definiált API-t valósítjuk meg, ismét egy külön modul segítségével. Ez az implementáció ugyanazt a nagybetűs átalakítást végzi el, de laza csatolással.

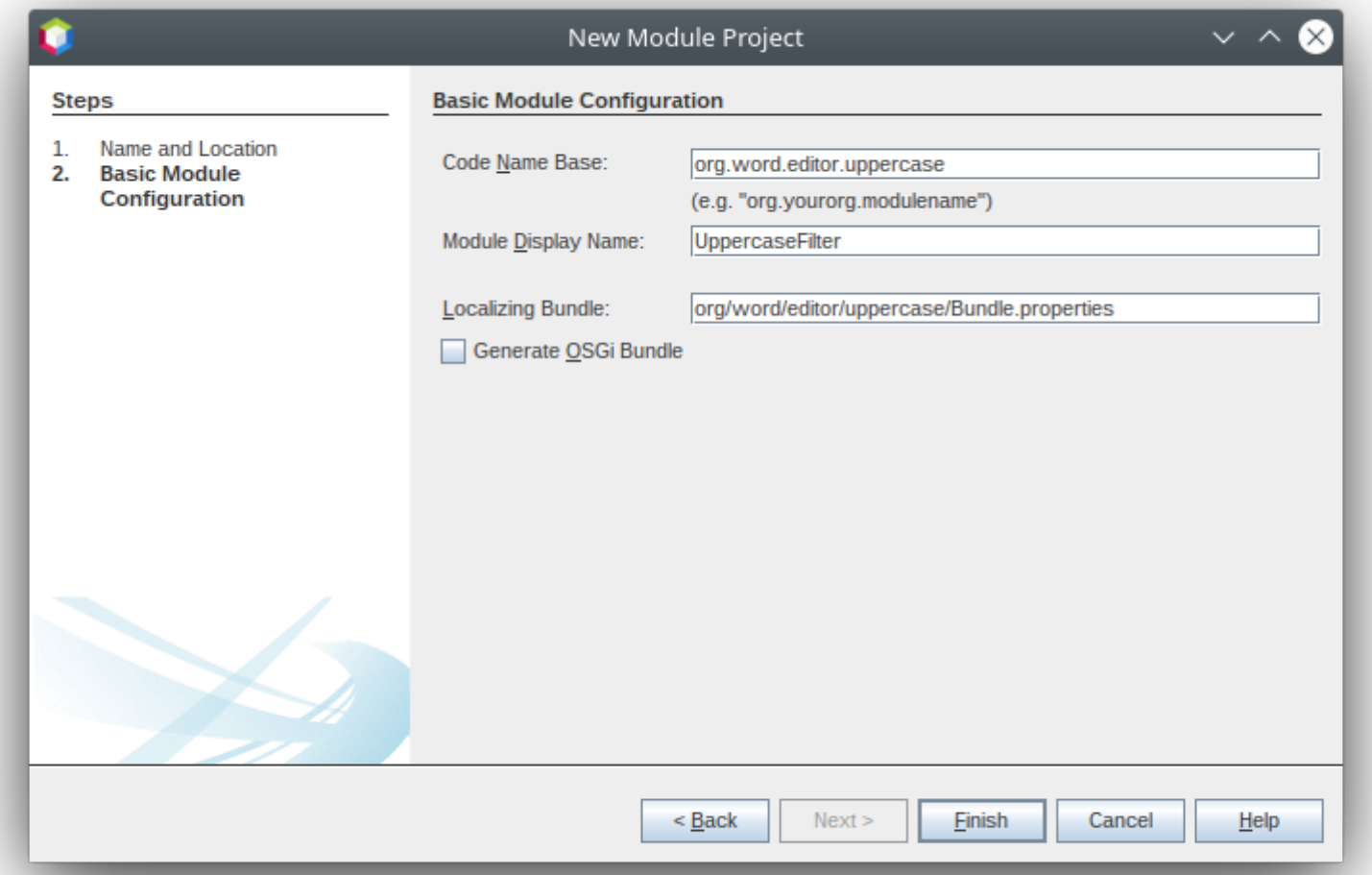
1. A Projektek ablakban kattintson a jobb gombbal az alkalmazás Modulok csomópontjára, és válassza ismét az "Új hozzáadása" lehetőséget:



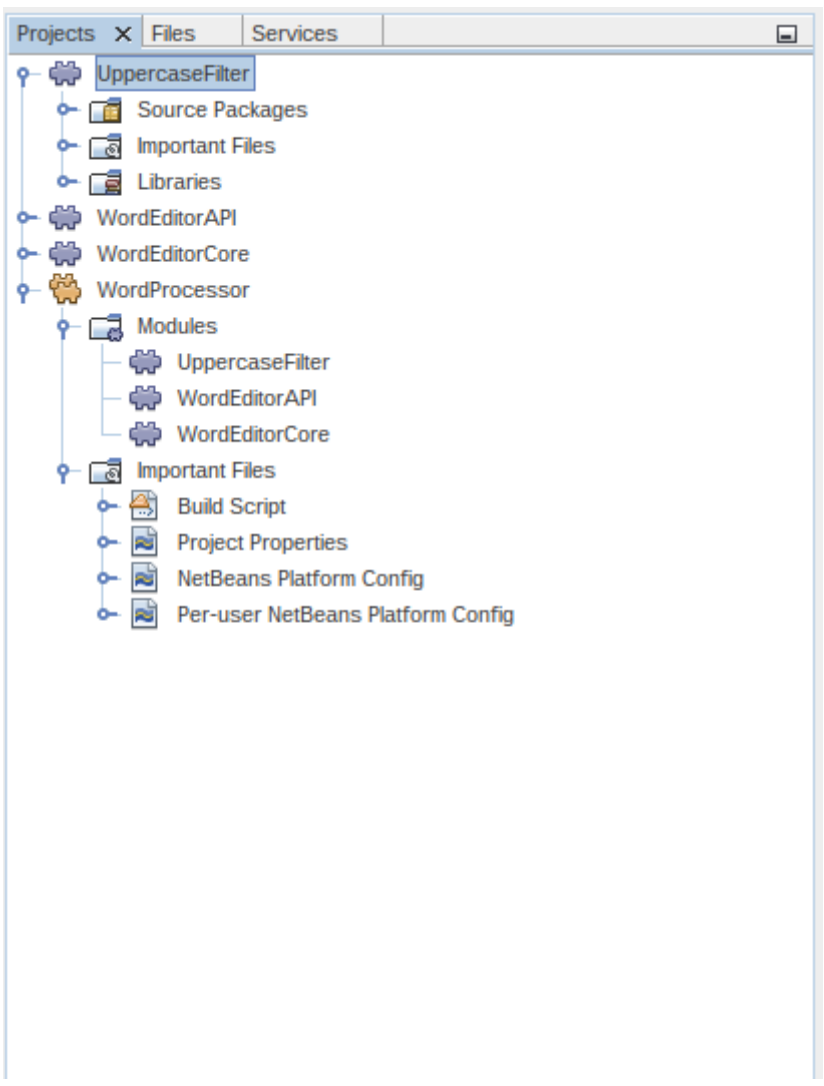
Nevezd el az új modult "UppercaseFilter"-nek:



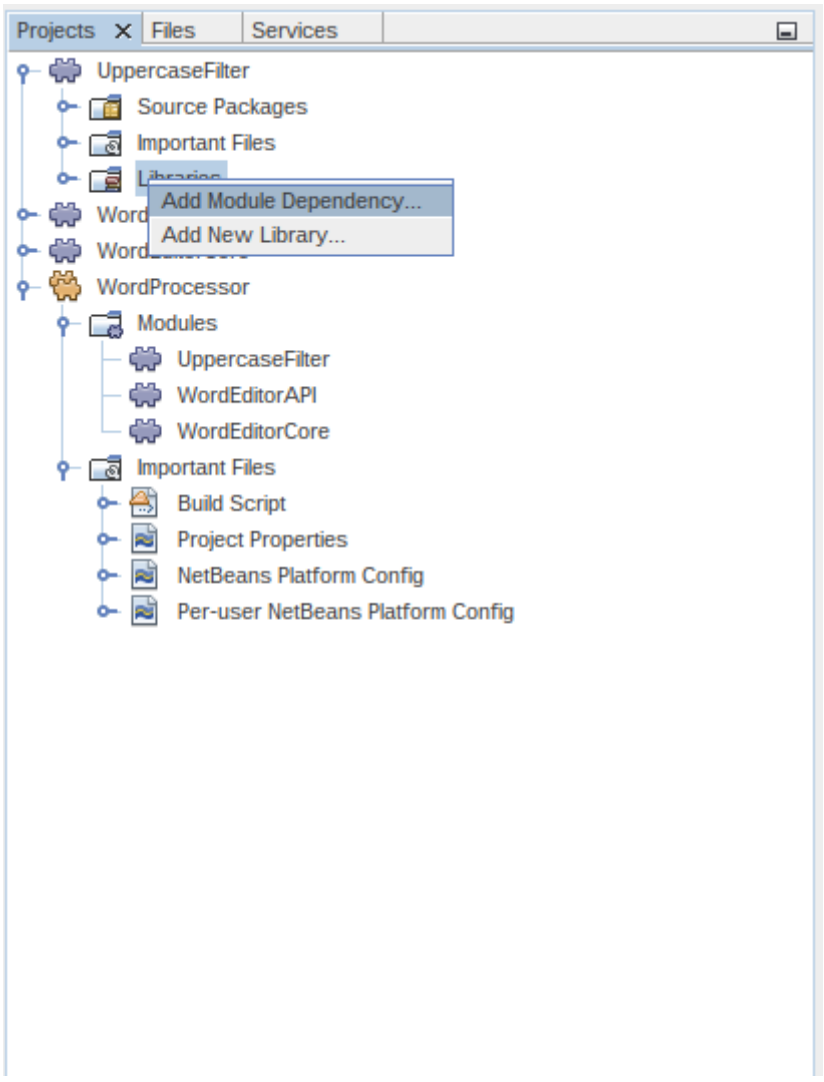
Kattintson a Tovább gombra. Állítsa be a Kódnév alapját az org.word.editor.uppercase értékre, az alábbiakban látható módon:



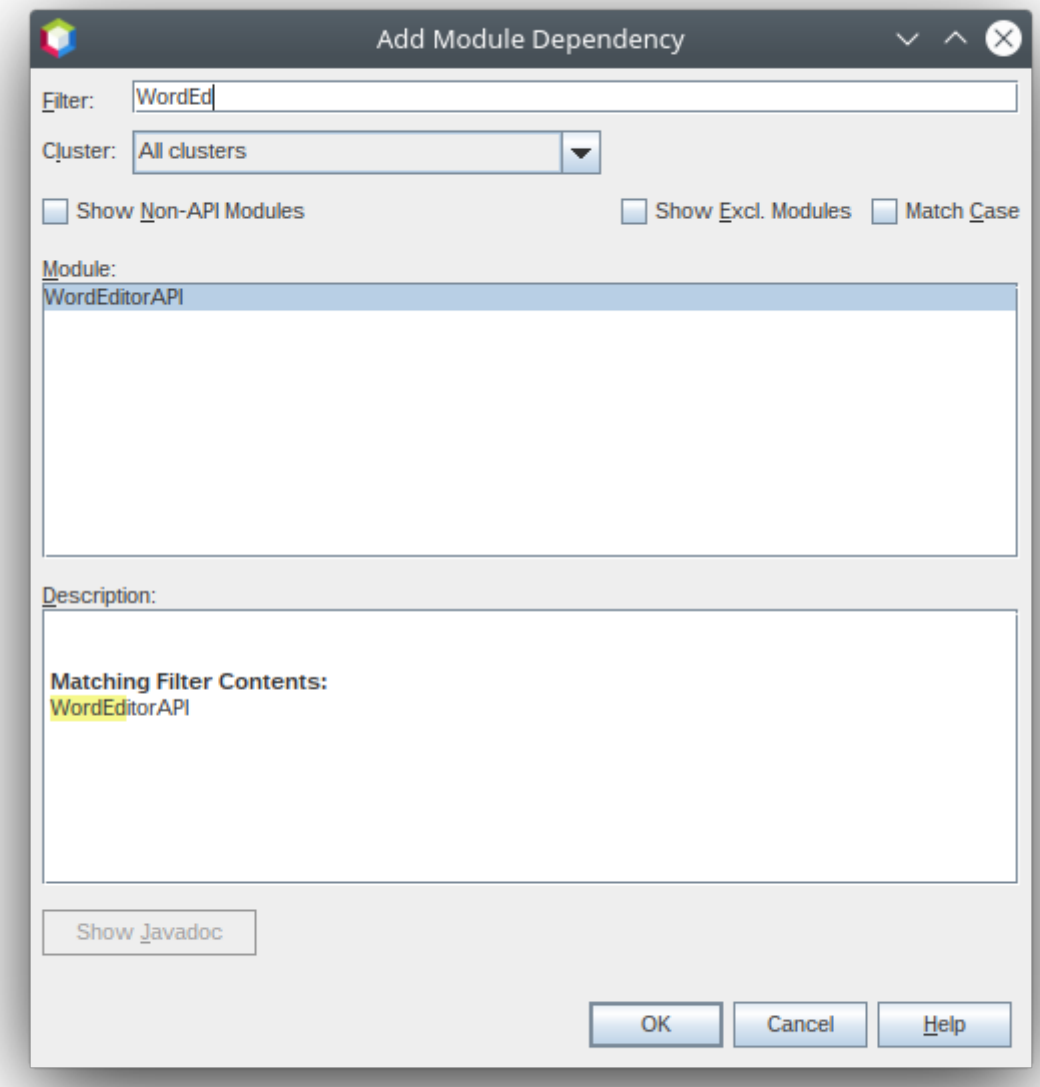
Kattintson a Befejezés gombra a varázsló befejezéséhez, amely hozzáadja a modult a korábban létrehozott alkalmazáshoz, ahogyan azt az előző szakaszban is megtette:



2. Kattintson a jobb gombbal az "UppercaseFilter" modul Libraries csomópontjára, és válassza a Modulfüggőség hozzáadása parancsot, ahogy az alább látható:

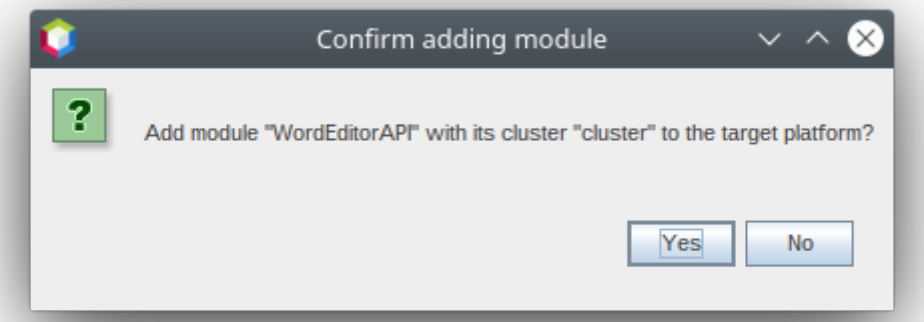


Kezdje el beírni az API osztály nevét (WordEditorAPI), és vegye észre, hogy a lista addig szűkül, amíg meg nem találja az osztályt tartalmazó modult:



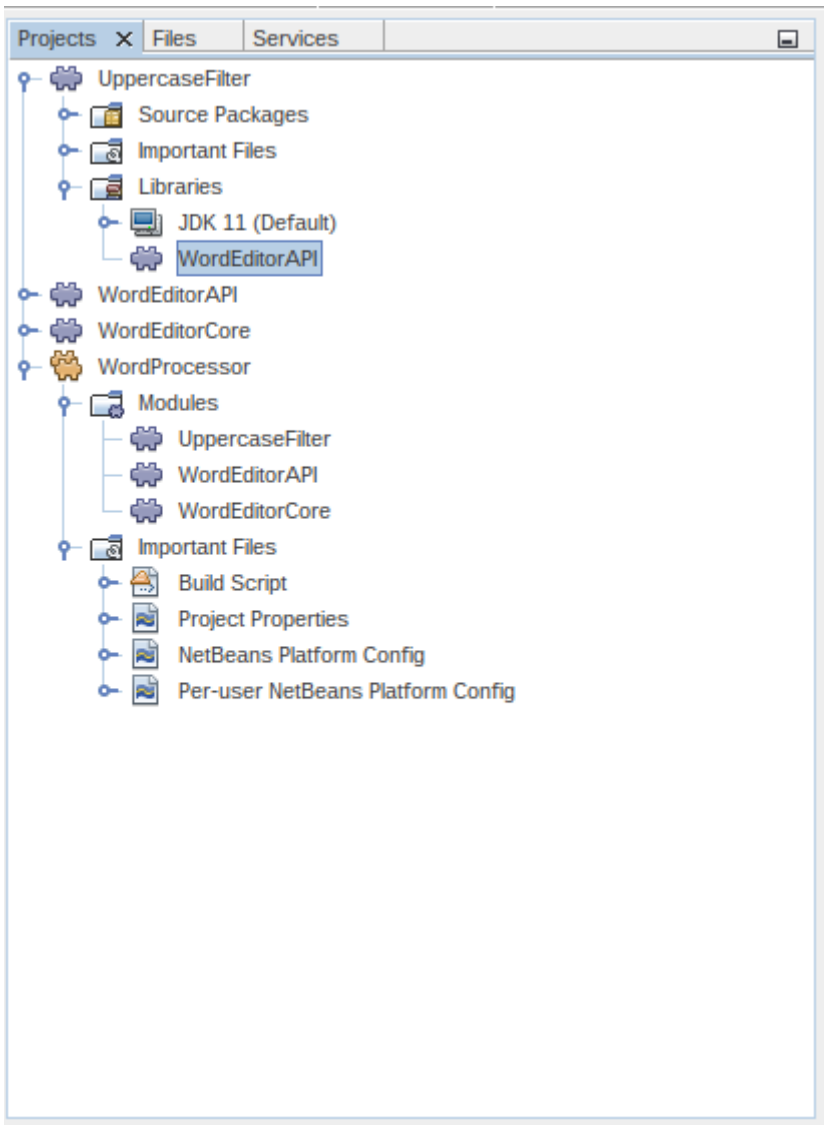
Kattintson az OK gombra.

Megjelenik egy megerősítő párbeszédpanel:



Kattintson az Igen gombra a függőség hozzáadásához.

A Projektek ablakban bontsa ki a "Könyvtárak" részt az "UppercaseFilter" projektben, hogy láthassa, hogy a "WordEditorAPI" függőség hozzá lett adva:

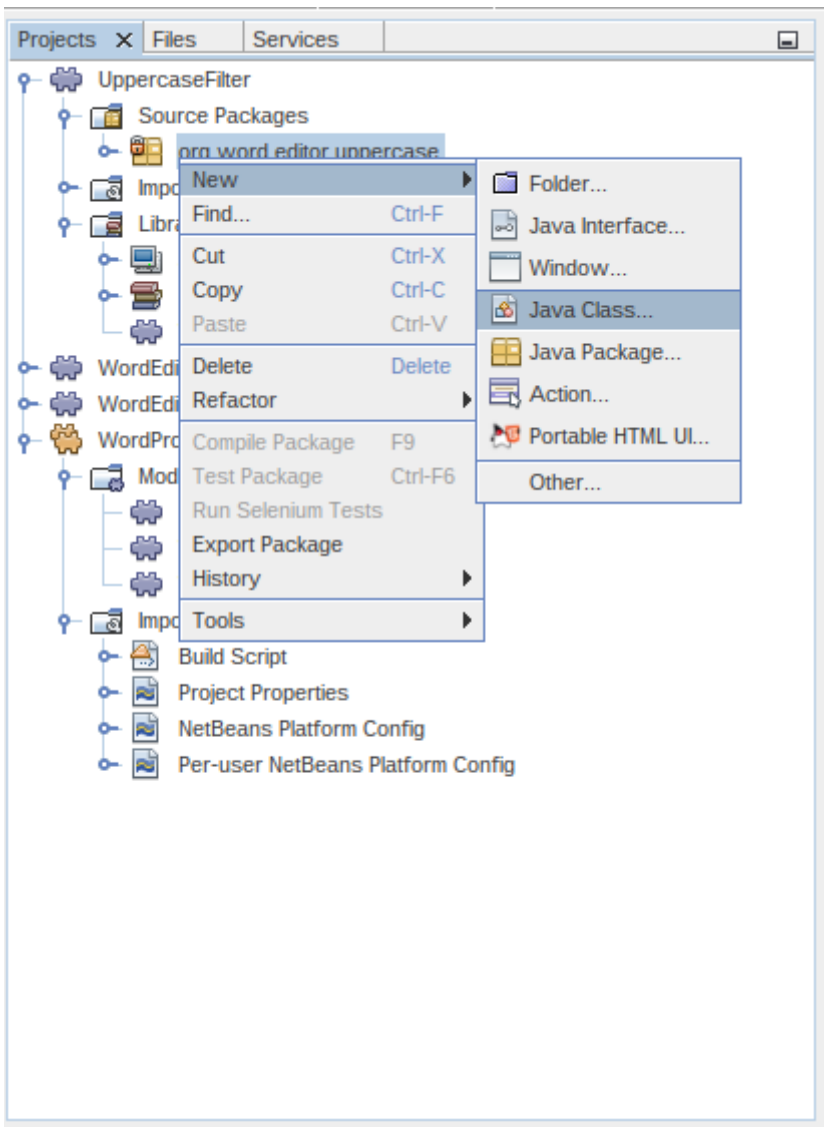


Egy másik lehetőség ennek megtekintésére, hogy a Projektek ablakban kattintson a "Fontos fájlok" menüpontra az "UppercaseFilter" projektben, majd kattintson duplán a "Projekt metaadatok" menüpontra. Megnyílik a "project.xml" fájl, és látnia kell, hogy egy új függőség lett deklarálva:

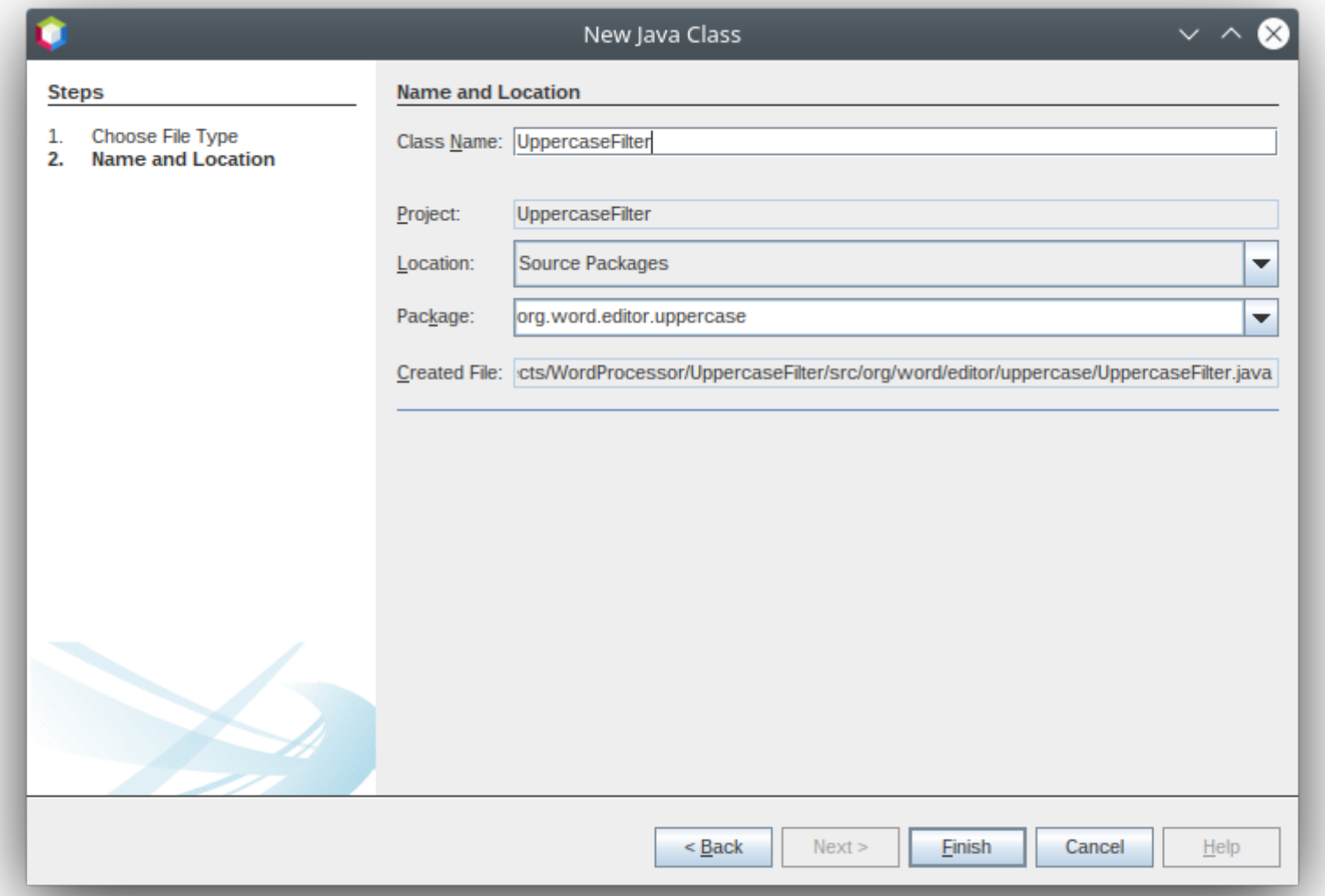
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://www.netbeans.org/ns/project/1">
    <type>org.netbeans.modules.apisupport.project</type>
    <configuration>
        <data xmlns="http://www.netbeans.org/ns/nb-module-project/3">
            <code-name-base>org.word.editor.uppercase</code-name-base>
            <suite-component/>
            <module-dependencies>
                <dependency>
                    <code-name-base>org.word.editor.api</code-name-base>
                    <build-prerequisite/>
                    <compile-dependency/>
                    <run-dependency>
                        <specification-version>1.0</specification-version>
                    </run-dependency>
                </dependency>
            </module-dependencies>
            <public-packages/>
        </data>
    </configuration>
</project>
```

3. Az előző lépésben bemutatott módon állítsunk függőséget a Lookup API modulra, amely biztosítja a @ServiceProvider megjegyzést, amelyet a következő lépésben fogunk használni.

4. Most már megvalósíthatja a WordEditorAPI modulban definiált interfészt. Az "UppercaseFilter" modulban hozzon létre egy új osztályt az org.word.editor.uppercase csomagban, az alábbiakban látható módon.



Nevezzük el az új osztályt UppercaseFilter-nek:



Kattintson a Befejezés gombra a varázslóból való kilépéshez és a fájl létrehozásához. A fájlnak automatikusan meg kell nyílnia szerkesztéshez.

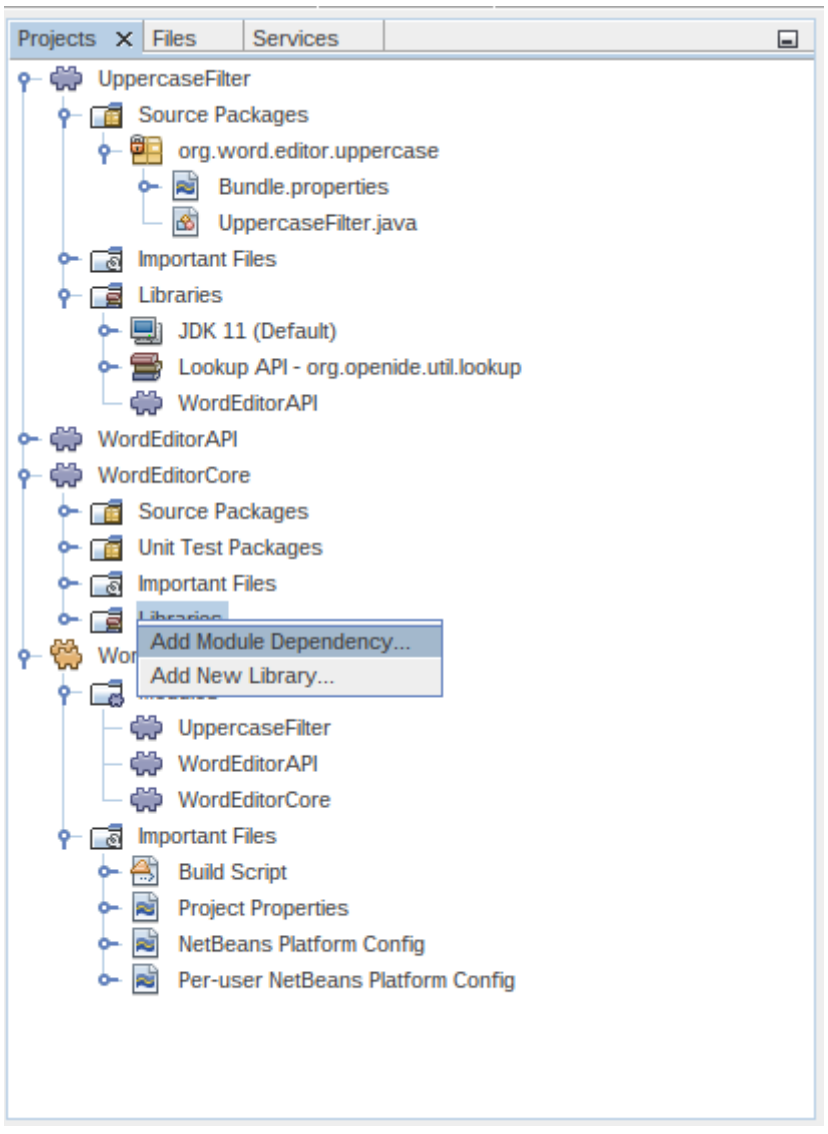
Definiálja az osztályt a következőképpen:

```
package org.word.editor.uppercase;

import org.openide.util.lookup.ServiceProvider;
import org.word.editor.api.WordFilter;

@ServiceProvider(service = WordFilter.class)
public class UppercaseFilter implements WordFilter {

    @Override
    public String process(String s) {
        return s.toUpperCase();
    }

}
```

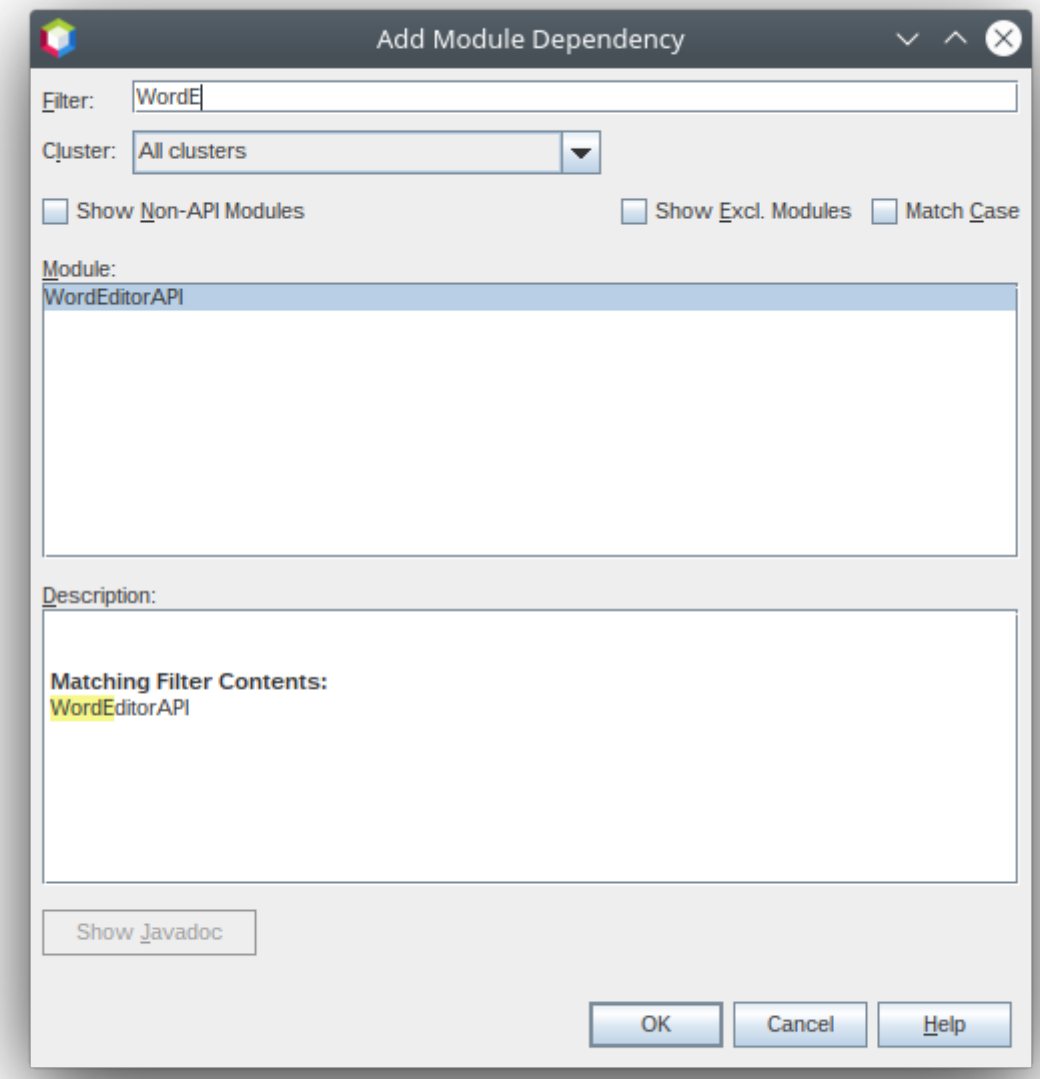
Fordításkor a @ServiceProvider annotáció létrehoz egy META-INF/services mappát egy olyan fájlal, amely a JDK 6 ServiceLoader mechanizmusát követve regisztrálja a WordFilter interfész implementációját.

Most frissítenünk kell a WordEditorCore modult, hogy a "WordFilter" interfész minden implementációja megtalálható és betöltődik. Amikor minden egyes implementációt megtaláltunk, meghívjuk a process metódusát a szöveg szűrésére. Mielőtt ezt megtehetnénk, a "WordEditorCore" modulban hozzá kell adnunk egy függőséget a "WordEditorAPI" modulhoz, hasonlóan ahhoz, ahogyan az UppercaseFilter esetében tettük.

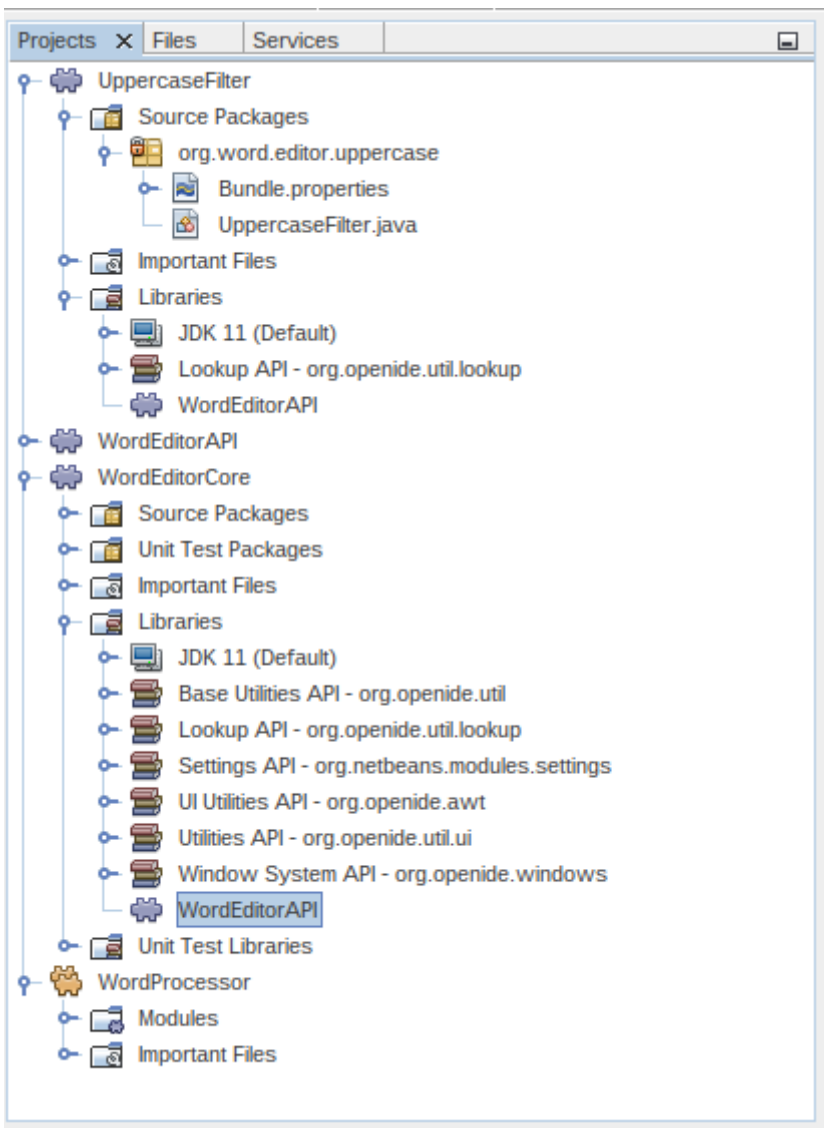
5. A Projektek fában bontsa ki a WordEditorCore modult, és keresse meg a Libraries csomópontot. Kattintson a jobb gombbal, és válassza a "Add Modules Dependency..." lehetőséget.



Adja hozzá a WordEditorAPI függőséget:



Nyissa ki a Könyvtárak bejegyzéseket a függőség hozzáadásának ellenőrzéséhez:



6. Most módosíthatjuk a WordTopComponent.java implementációt, hogy betöltse a "WordFilter" interfész implementációit. Cseréljük le a korábbi implementációt (amely keményen kódolva volt, hogy csak nagybetűs szöveget tartalmazzon) a következőkre:

```
private void filterButtonActionPerformed(java.awt.event.ActionEvent evt) {
    String enteredText = text.getText();
    Collection<? extends WordFilter> allFilters = Lookup.getDefault().lookupAll(WordFilter.class);
    StringBuilder sb = new StringBuilder();
    for (WordFilter textFilter : allFilters) {
        String processedText = textFilter.process(enteredText);
```

```
            sb.append(processedText).append("\n");
        }
        text.setText(sb.toString());
    }
```

A szükséges importok a következők:

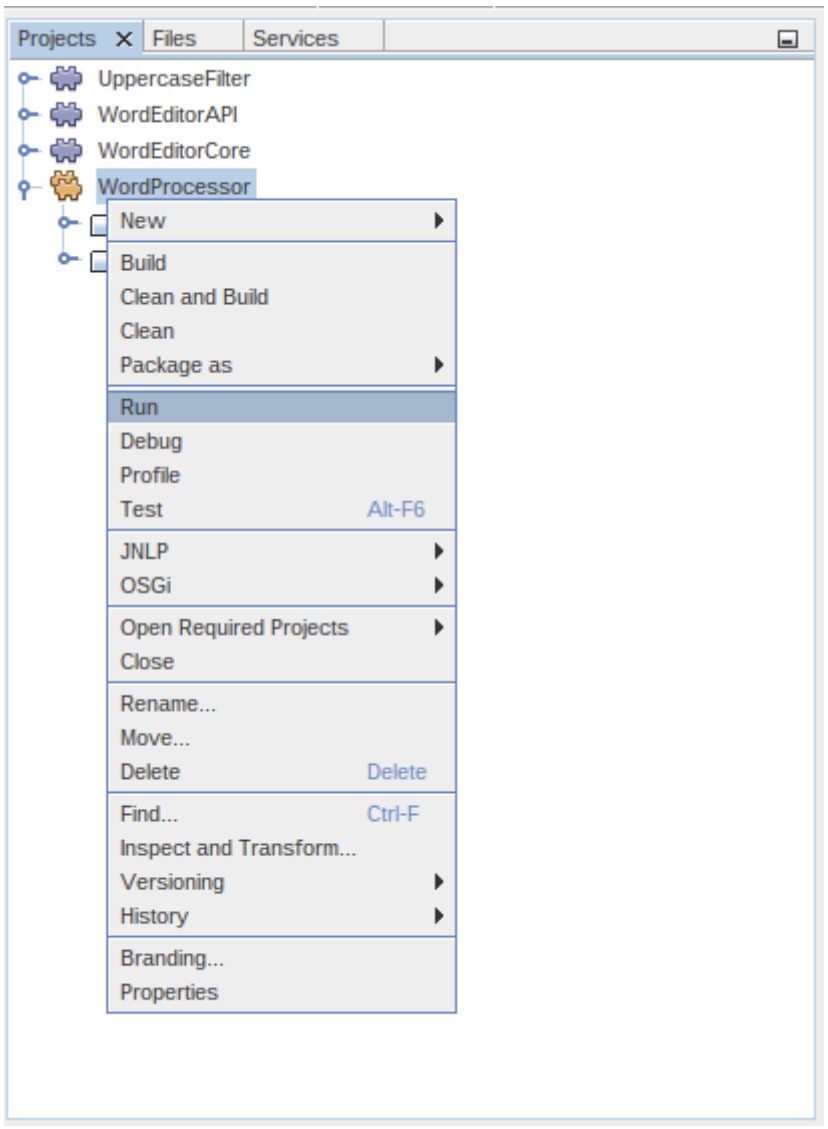
```
import java.util.Collection;
import org.netbeans.api.settings.ConvertAsProperties;
import org.openide.awt.ActionID;
import org.openide.awt.ActionReference;
import org.openide.util.Lookup;
import org.openide.windows.TopComponent;
import org.openide.util.NbBundle.Messages;
import org.word.editor.api.WordFilter;
```

A Lookup lehetőséget biztosít a szolgáltatás betöltésére anélkül, hogy a fogyasztókat konkrét szolgáltatásimplementációkhoz kötnénk. Ez a kulcsa az Apache NetBeans Platform által biztosított rugalmas, bővíthető architektúrának.

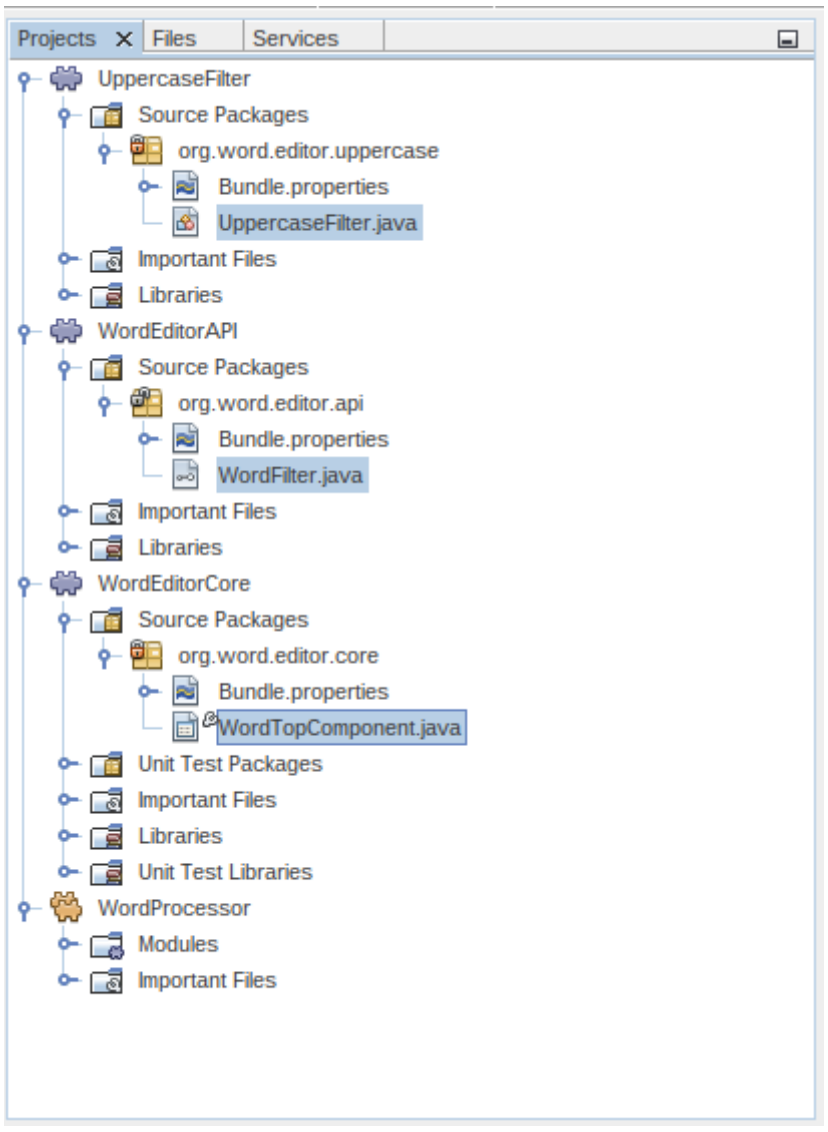
Az alkalmazás futtatása

Ebben a szakaszban újra futtatja az alkalmazást.

1. Most újra futtathatja az alkalmazást, és ellenőrizheti, hogy minden ugyanúgy működik-e, mint korábban.



Bár a funkcionalitás ugyanaz, az új moduláris felépítés egyértelműen elválasztja a grafikus felületet és a szűrő végrehajtását. Az alkalmazás felépítésének az alábbiakban bemutatott módon kell kinéznie:



2. Az új alkalmazás könnyen bővíthető új szolgáltatók hozzáadásával az alkalmazás osztályútvonalához. Gyakorlatként adjunk hozzá egy új modult, amely az API "LowercaseFilter" implementációját biztosítja az alkalmazáshoz.

Megjegyzés: Ha egynél több szűrő van, akkor minden szűrő eredménye hozzáadódik a szövegterülethez.

Most az alapértelmezett Lookupot, azaz a "Lookup.getDefault()"-t használtuk egy interfész megvalósításainak betöltésére a META-INF/services mappából.

Publikálás és feliratkozás a keresőre

Ebben a szakaszban létrehozunk egy negyedik modult, amely dinamikusan kap szövegeket, amikor az első modulban a "Filter!" gombra kattintunk.

- Közzététel a Lookuphoz
- Feliratkozás a keresőre

Közzététel a keresőbe

Ebben a szakaszban egy karakterláncot tesz közzé a TopComponent Lookupjában. Amikor a TopComponent kiválasztásra kerül, a karakterláncot az alkalmazás kontextusában közzéteszi.

1. A "WordEditorCore" modulban egy karakterláncot teszünk közzé, amikor a felhasználó a "Filter!" gombra kattint. Ehhez adjunk hozzá egy tagváltozót, és frissítsük a "WordTopComponent" konstruktorát az alábbiak szerint:

```
private final InstanceContent content;

public WordTopComponent() {
    initComponents();
    setName(Bundle.CTL_WordTopComponent());
    setToolTipText(Bundle.HINT_WordTopComponent());
    content = new InstanceContent();
    this.associateLookup(new AbstractLookup(content));
}
```

2. Módosítsa a szűrőgomb kódját úgy, hogy a beírt szöveg a gombra kattintáskor hozzáadódjon az InstanceContent objektumhoz.

```
private void filterButtonActionPerformed(java.awt.event.ActionEvent evt) {
    String enteredText = text.getText();
    Collection<? extends WordFilter> allFilters = Lookup.getDefault().lookupAll(WordFilter.class);
    StringBuilder sb = new StringBuilder();
    for (WordFilter textFilter : allFilters) {
        String processedText = textFilter.process(enteredText);
        sb.append(processedText).append("\n");
```

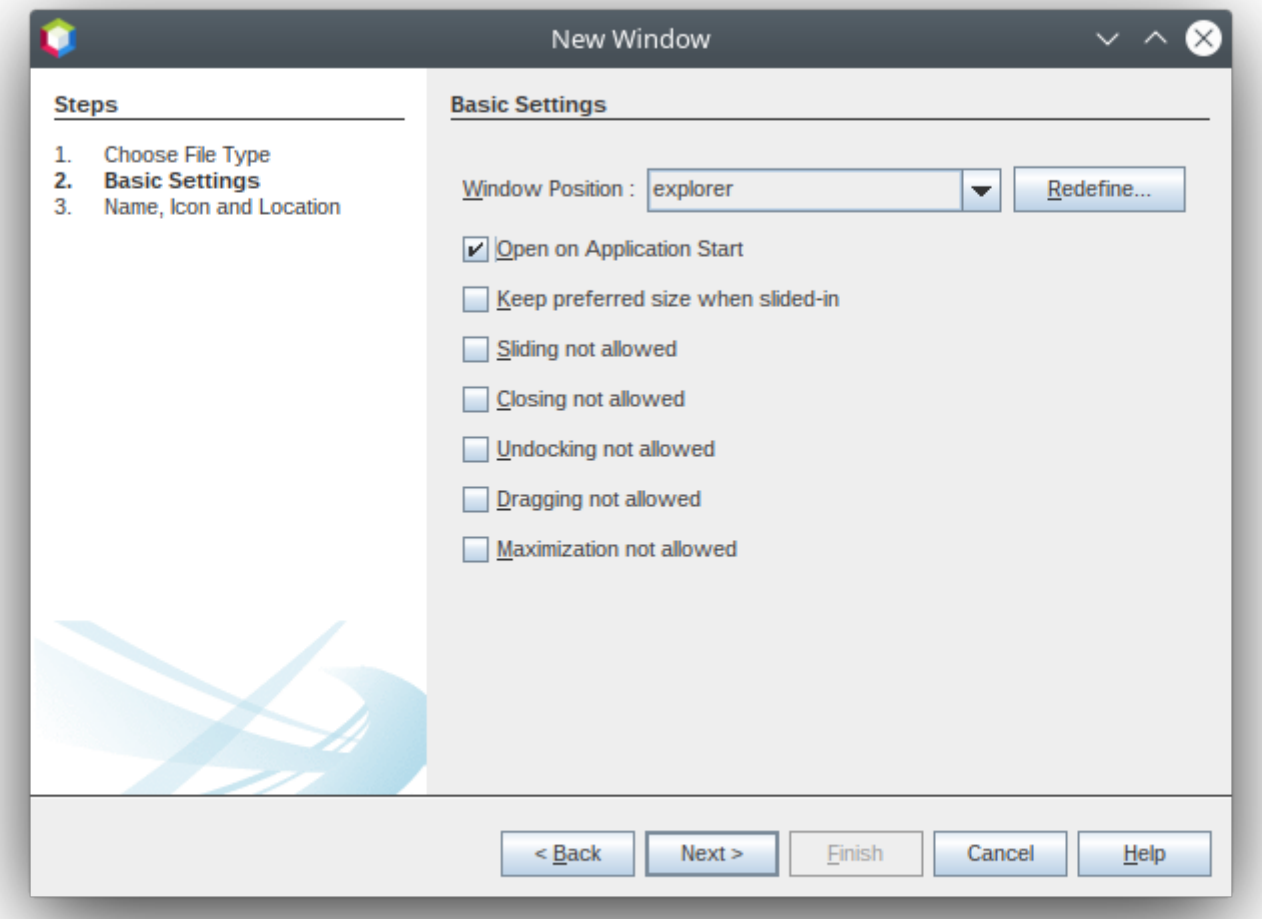
```
            content.add(enteredText);
        }
        text.setText(sb.toString());
    }
```

Feliratkozás a keresésre

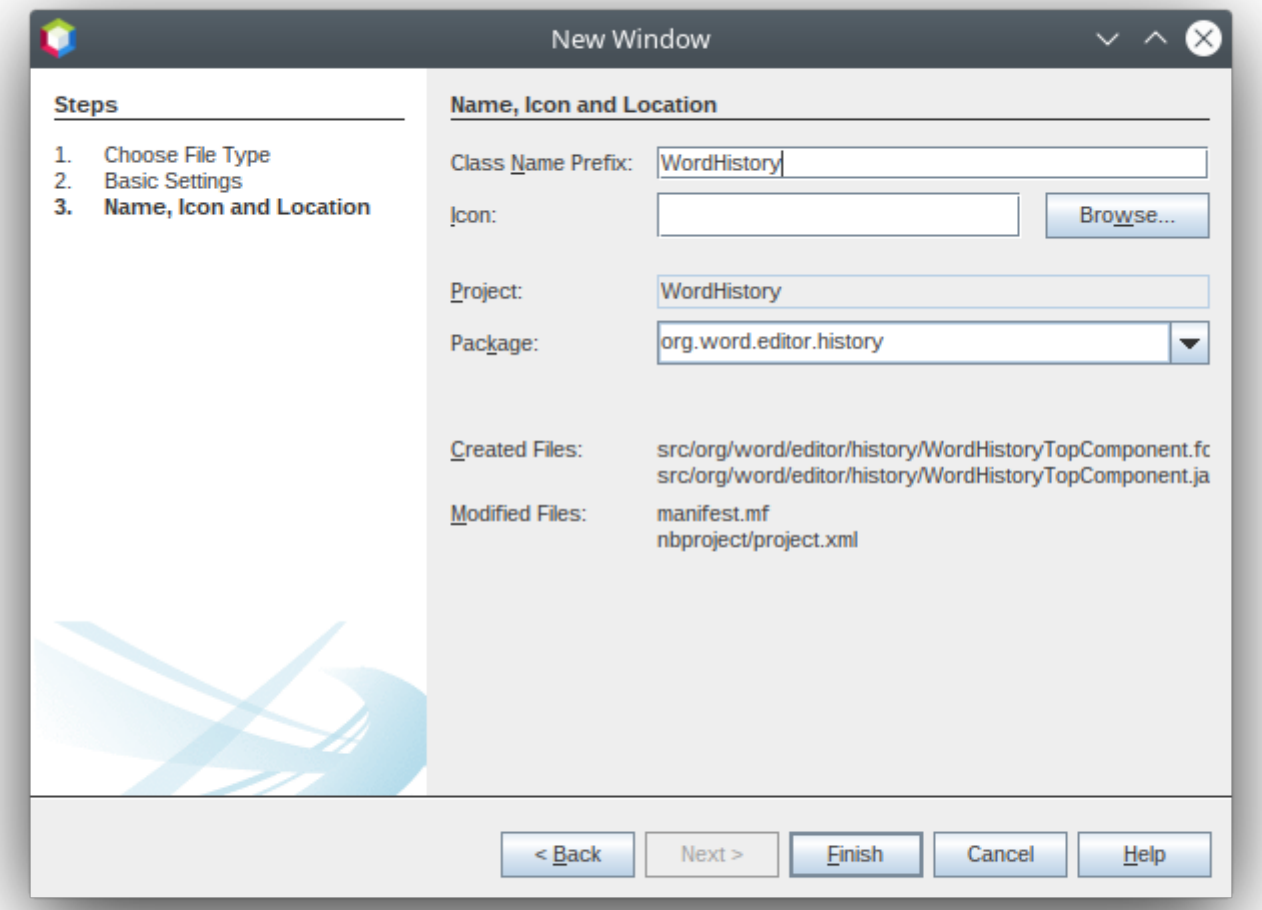
Ebben a szakaszban egy új modult hoz létre, egy új ablakkal. Az új ablakban meghallgatja az alkalmazás kontextusában a Strings. Ha a Lookupban van egy új karakterlánc, akkor megjeleníti azt az ablakban.

1. Ugyanúgy, ahogy az előző szakaszokban történt, hozzon létre egy másik modult az alkalmazásban, és nevezze el "WordHistory"-nak. Állítsuk be a kódnév alapját úgy, hogy org.word.editor.history.

2. A WordHistory modulban kattintson a jobb gombbal az org.word.editor.history csomagra, és válassza az Új | Ablak parancsot. Az Új ablak varázsló segítségével hozzon létre egy új ablakkomponenst, amely automatikusan az alkalmazáskeret bal oldalán, azaz a felfedező pozíciójában nyílik meg:

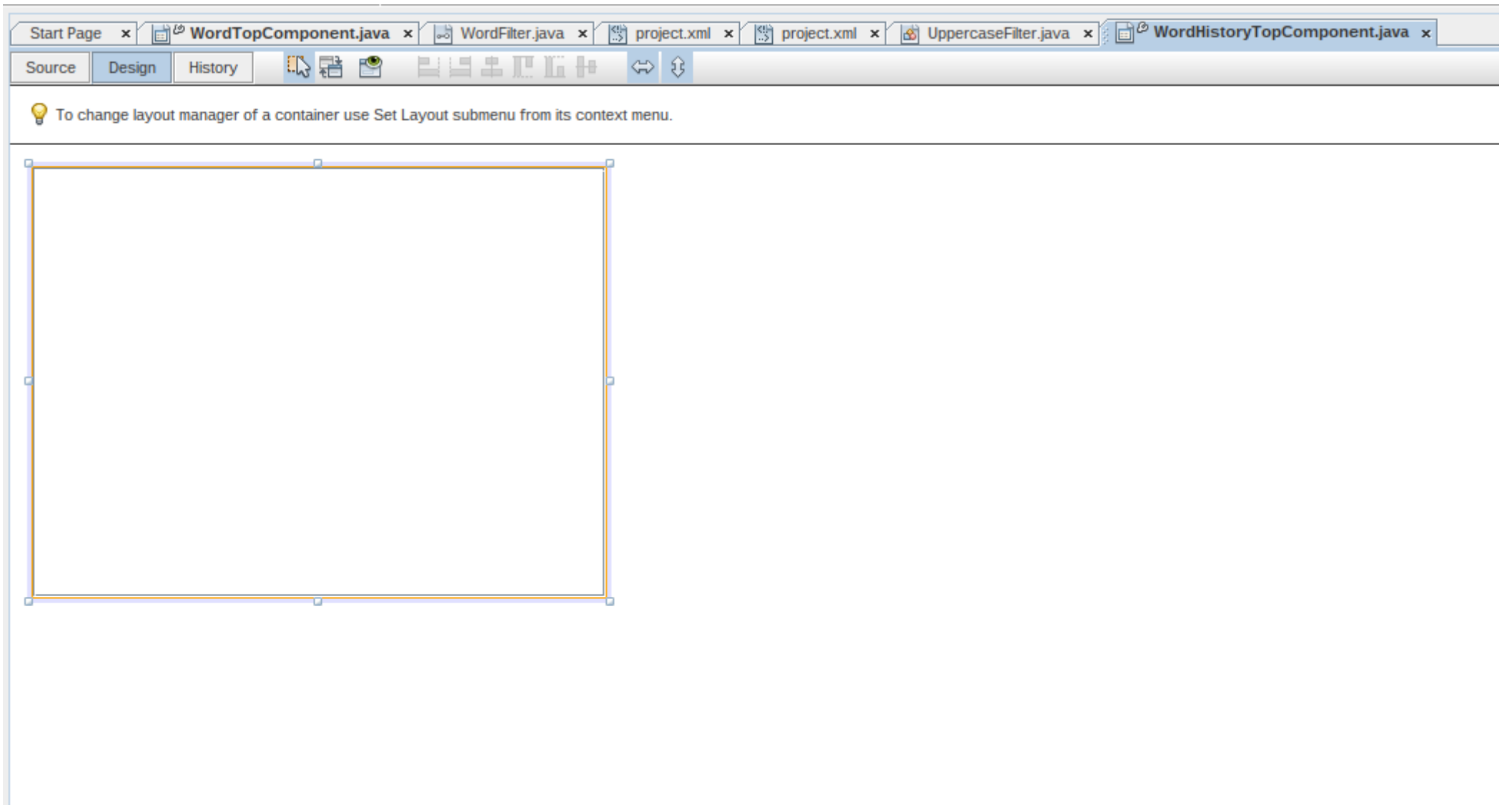


Kattintson a Tovább gombra. Használja a WordHistory előtagot, és adja meg, hogy az új ablak az org.word.editor.history csomagban kerüljön tárolásra.



Kattintson a Befejezés gombra a varázsló befejezéséhez és az Ablak létrehozásához.

3. Miután létrehozta az ablakot, adjon hozzá egy szövegterületet (JTextArea), és méretezze át úgy, hogy az ablak teljes területét betöltse:



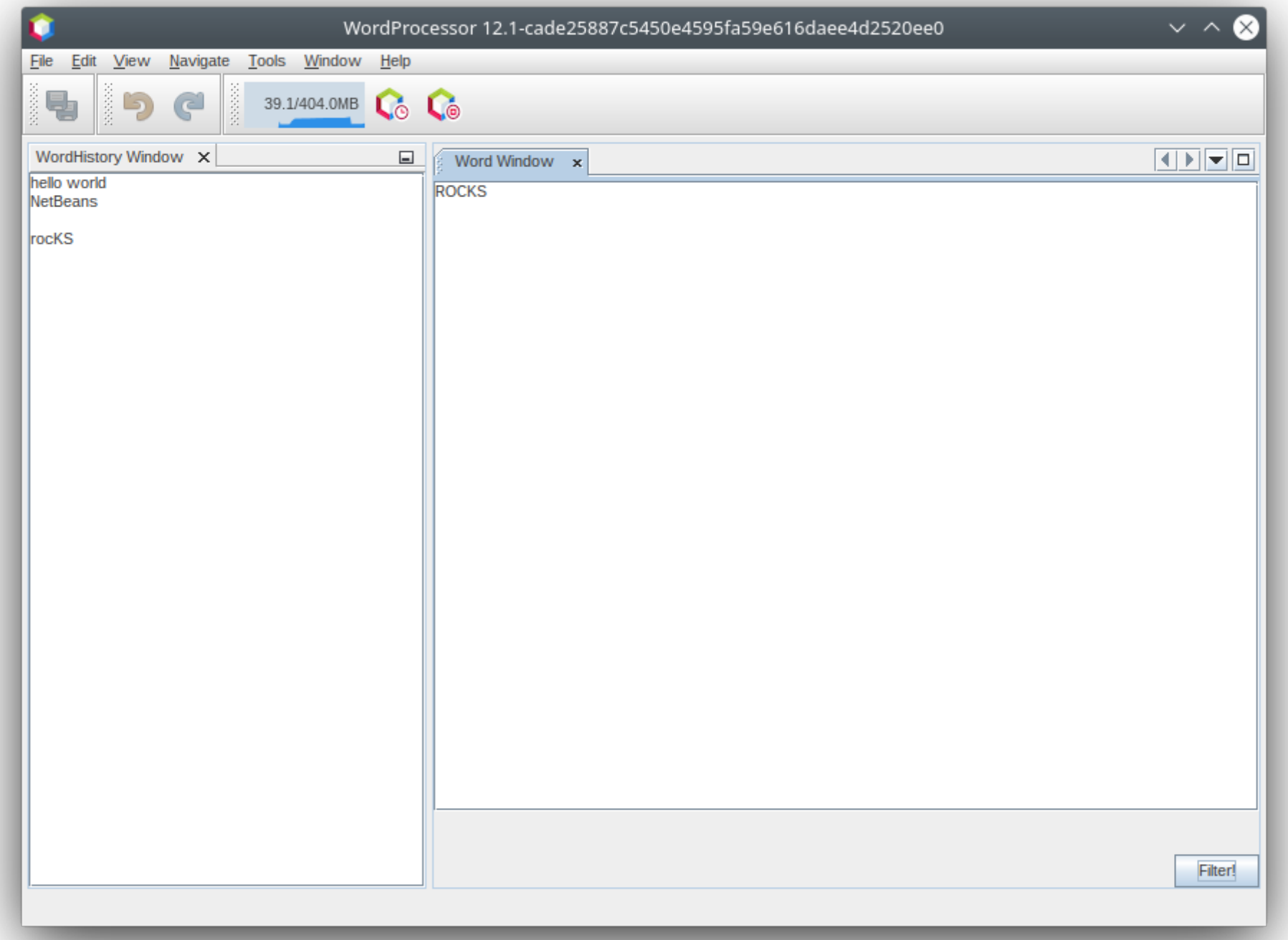
Módosítsa a szövegterület változójának nevét "historyText"-re.

4. A Source nézetben adjunk hozzá kódot a HistoryTopComponent osztályhoz, hogy az figyelje az aktuálisan aktív ablak String osztályának keresését (implementálja az org.openide.util.LookupListener-t), és jelenítse meg az összes lekérdezett String objektumot a szövegterületen. Frissítse a componentOpened() és componentClosed() metódusokat, és adjon hozzá result és resultChanged tagokat, valamint az implements és további importálásokat.

```
import org.openide.util.LookupEvent;
import org.openide.util.LookupListener;

public final class WordHistoryTopComponent extends TopComponent implements LookupListener {
```

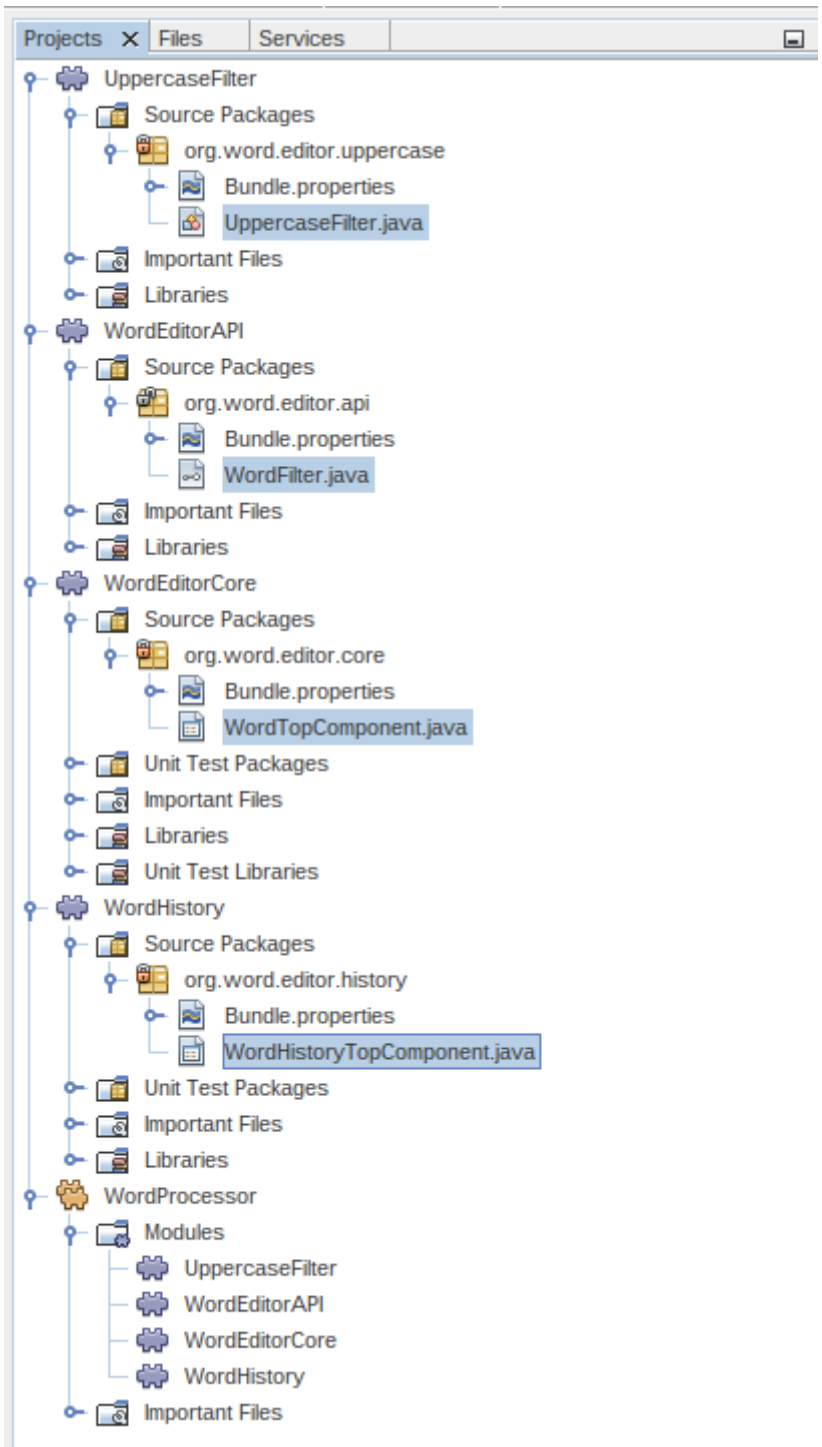
```
    private org.openide.util.Lookup.Result<String> result;

    @Override
    public void componentOpened() {
        result = org.openide.util.Utilities.actionsGlobalContext().lookupResult(String.class);
        result.addLookupListener(this);
    }

    @Override
    public void componentClosed() {
        result.removeLookupListener(this);
    }

    @Override
    public void resultChanged(LookupEvent le) {
        Collection<? extends String> allStrings = result.allInstances();
        StringBuilder sb = new StringBuilder();
        for (String string : allStrings) {
            sb.append(string).append("\n");
        }
        historyText.setText(sb.toString());
    }
```

5. Ezután elindíthatja az alkalmazást, és kísérletezhet vele. Az eredménynek az alábbi képernyőképen láthatóhoz hasonlóan kell kinéznie:



Gyakorlatként tervezze át a "WordTopComponent" felhasználói felületét úgy, hogy egy JList jelenítse meg a szűrőket.

Gratulálunk! Ebben a szakaszban, nagyon kevés kódolással, létrehoztál egy kis példát egy lazán kapcsolt moduláris alkalmazásra:



Két fontos fogalomról van szó ebben a bemutatóban.

1. Az alkalmazás négy modulból áll. Az egyik modul kódját csak akkor használhatja egy másik modul, ha (1) az első modul kifejezetten csomagokat állít ki, és (2) a második modul függőséget állít az első modultól. Ily módon az Apache NetBeans Platform segít a kód szigorú moduláris architektúra szerinti szervezésében, biztosítva, hogy a kód nem véletlenszerűen kerül újrafelhasználásra, hanem csak akkor, ha a kódot biztosító modulok között szerződések vannak beállítva.

2. Másodszor, a Lookup osztály a modulok közötti kommunikáció mechanizmusaként került bevezetésre. A megvalósítások betöltése az interfészeiken keresztül történik. A "WordEditorCore" modul anélkül, hogy bármilyen kódot használna egy implementációból, képes megjeleníteni az implementátor által nyújtott szolgáltatást. Ez lehetővé teszi a laza csatolást az Apache NetBeans Platform alkalmazásaiban.

A modularitással és a NetBeans Platformmal kapcsolatos további ismeretekért látogasson el a négyrészes "NetBeans Platform Selection Management" sorozatba, amely itt kezdődik. Ezt követően kezdje el a NetBeans Platform tanulási tanösvényt, és válassza ki az adott üzleti forgatókönyv szempontjából legmegfelelőbb oktatóanyagokat. Továbbá, ha bármilyen kérdése van a NetBeans Platformmal kapcsolatban, írjon bátran a **dev@platform.netbeans.org** levelezési listára; a kapcsolódó archívumot is itt találja.

Eredeti oldal

NetBeans Platform bemutató integrált fejlesztőkörnyezetekhez

A NetBeans Platform szilárd infrastruktúra a saját szoftverfejlesztő eszközök létrehozásához. Az egyedi szoftverfejlesztő eszközökön túl azonban egy integrált fejlesztőkörnyezet (IDE) több különböző eszközt egyetlen összefüggő rendszerbe foglal össze. A NetBeans IDE maga is egy ilyen összefüggő rendszer példája, bár sok más is a NetBeans Platformot használja alapként.

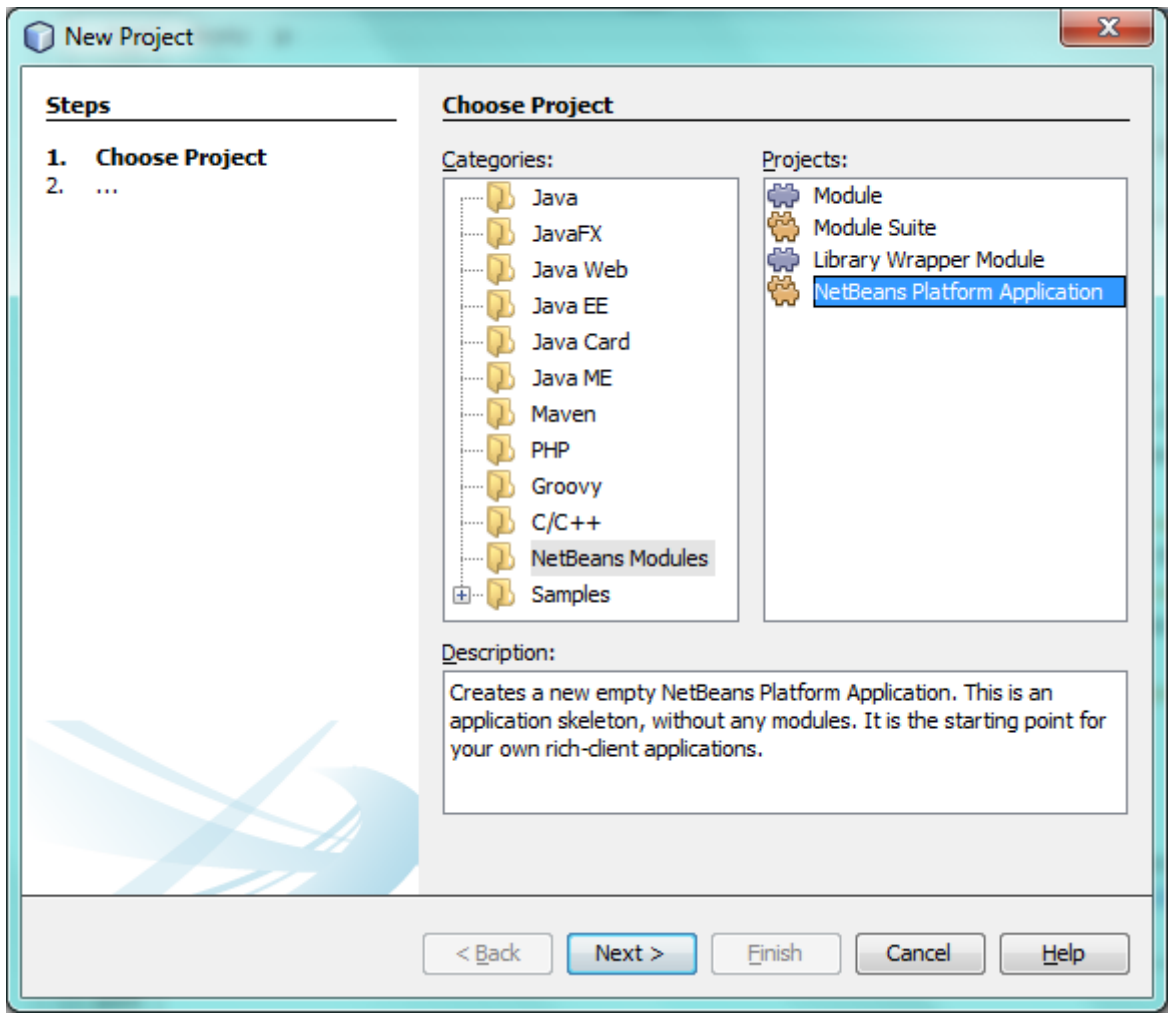
Ebben a gyorstalpalóban bemutatjuk, hogyan állíthat be egy integrált fejlesztőkörnyezetet a NetBeans Platformra.

1. rész: Általános alkalmazásalap létrehozása
2. rész: Előre definiált IDE-szolgáltatások beépítése
3. rész: Egyéni IDE-szolgáltatások kódolása

Általános alkalmazásalap létrehozása

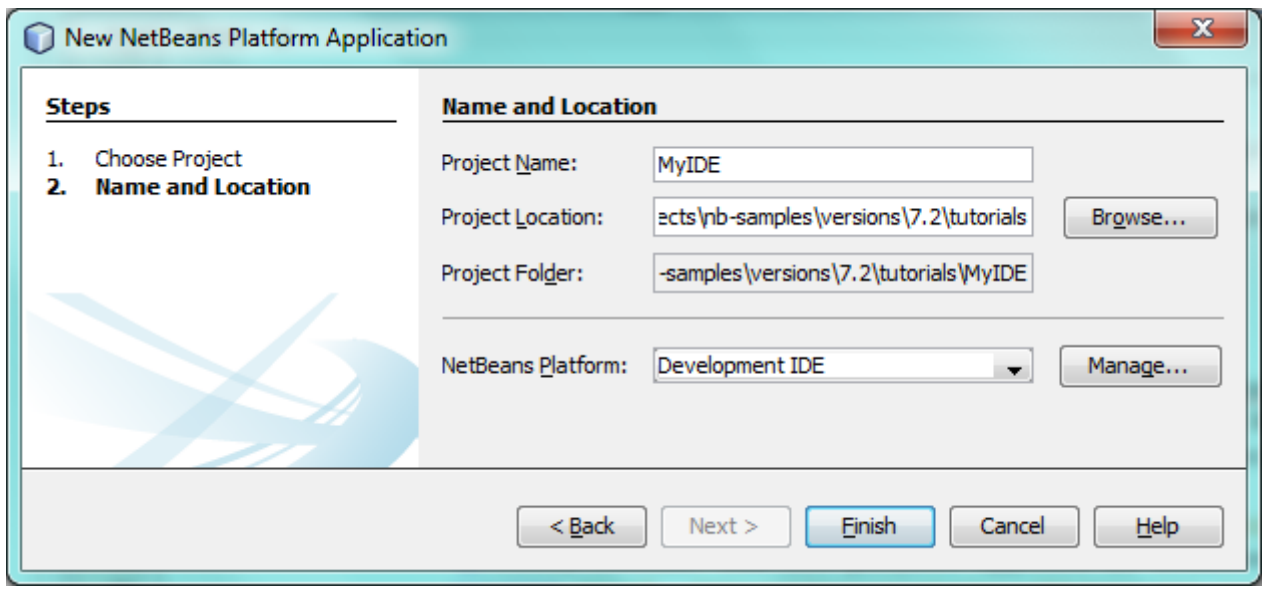
Kezdjük egy új NetBeans Platform alkalmazás létrehozásával, amely a térinformatikai rendszerünk kiindulópontja.

1. Válasszuk a Fájl | Új projekt, majd a NetBeans modulok menüpontot. Válasszuk ki a "NetBeans Platform alkalmazás" lehetőséget:

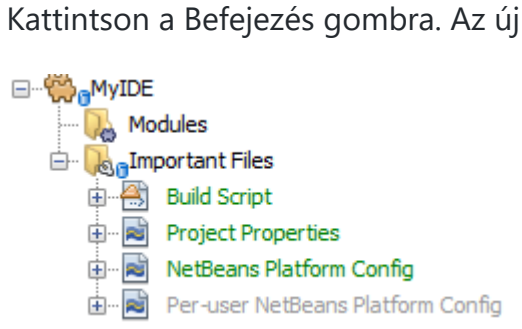


Kattintson a Tovább gombra.

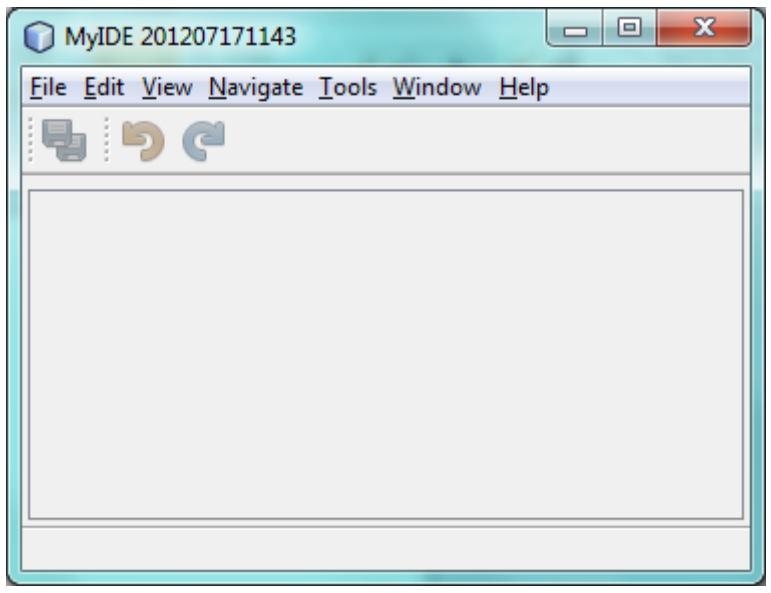
2. Nevezze el az új alkalmazást "MyIDE" névre, és adjon meg egy mappát a lemezen a tárolásához:



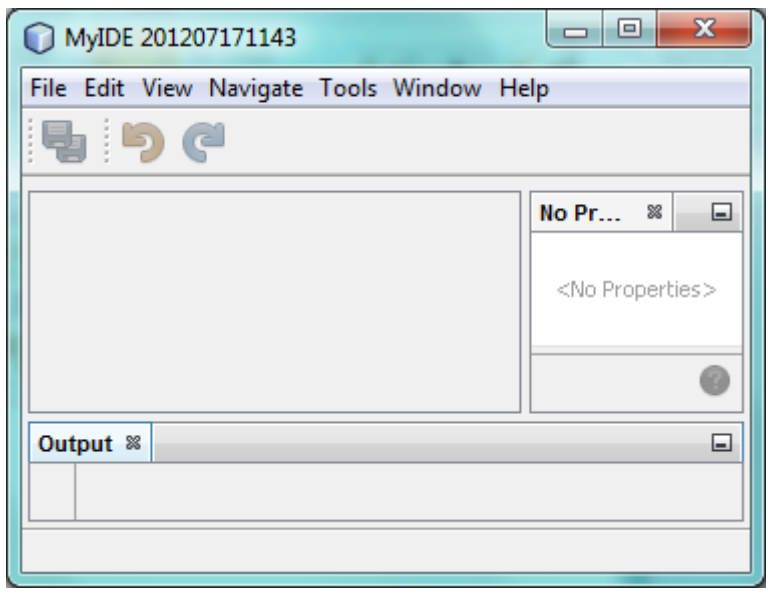
Kattintson a Befejezés gombra. Az új projekt a következőképpen jelenik meg a Projektek ablakban:



3. Kattintson a jobb gombbal az alkalmazásra, és válassza a Futtatás parancsot. Az alkalmazás települ, és ezt kell látnia:



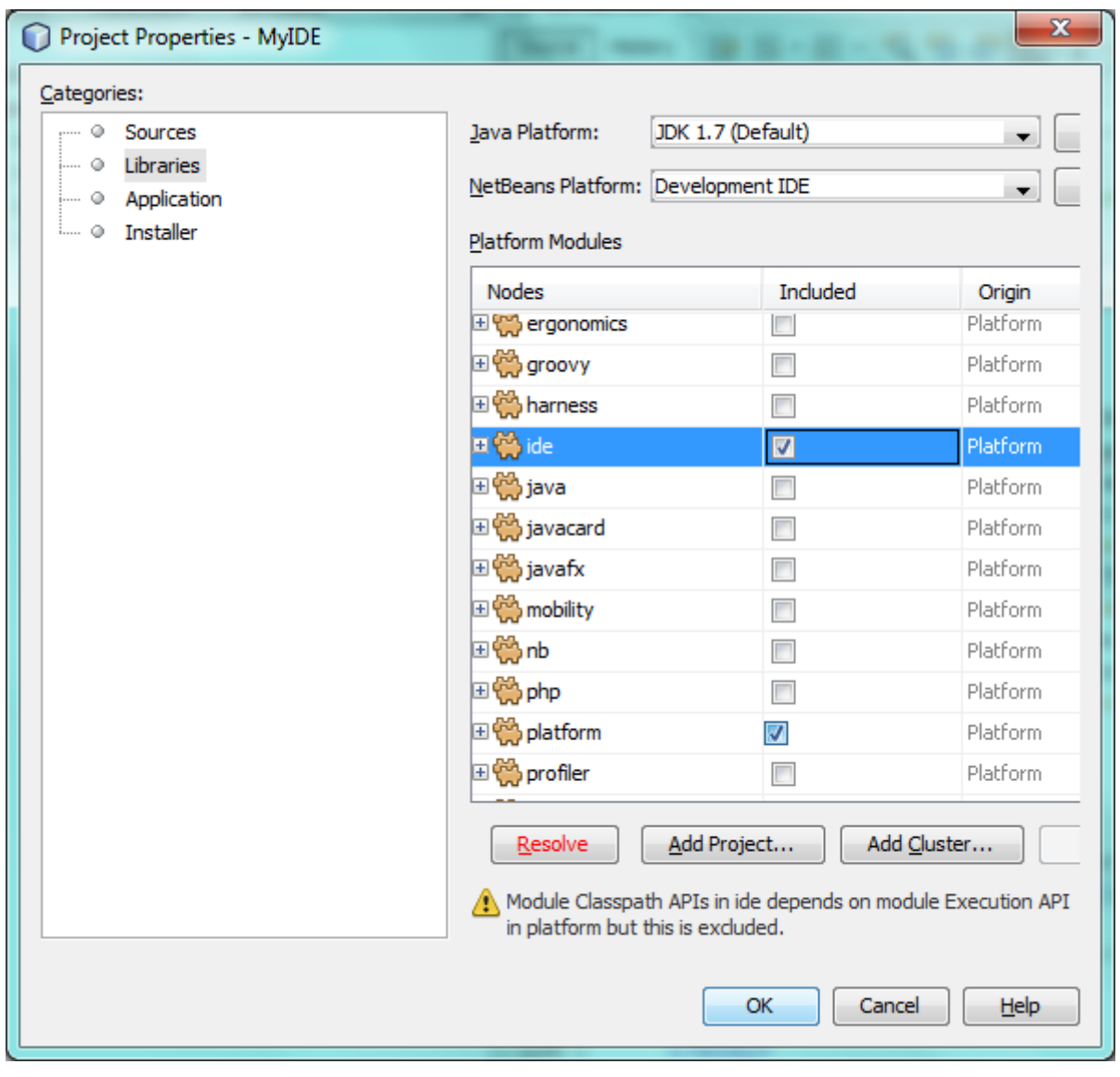
4. Fedezze fel az alkalmazás menüit és eszköztárait, és vegye észre, hogy egy általános asztali alkalmazás struktúrájával rendelkezik:



Előre definiált IDE-szolgáltatások beépítése

Ebben a részben számos további, a NetBeans IDE-ben használt modult mutatunk be, amelyek bármely más IDE-ben is hasznosak lehetnek.

1. Kattintson a jobb gombbal a MyIDE projekt csomópontjára, és válassza a Tulajdonságok parancsot. A Projekt tulajdonságai párbeszédpanel Könyvtárak paneljén kattintson az "ide" jelölőnégyzetre, ahogy az alább látható:



Kattintson a piros Feloldás gombra, ha olyan modulokat szeretne bevonni, amelyektől a bevont modulok függenek. Kattintson a Befejezés gombra.

2. Kattintson a jobb gombbal az alkalmazásra, és válassza a "Clean" lehetőséget, hogy eltávolítsa az előző futtatásból származó felhasználói gyorsítótárat, és visszaállítsa az alkalmazás alapértelmezett állapotát. Ezután kattintson a jobb gombbal az alkalmazásra, és válassza a Futtatás

parancsot. Az új projekt a következőképpen jelenik meg a Projektek ablakban. Vegye észre, hogy most már számos további, minden IDE-re jellemző funkcióval rendelkezik, amelyeket a következő részben részletesebben is megvizsgálunk.

Egyéni IDE-szolgáltatások kódolása

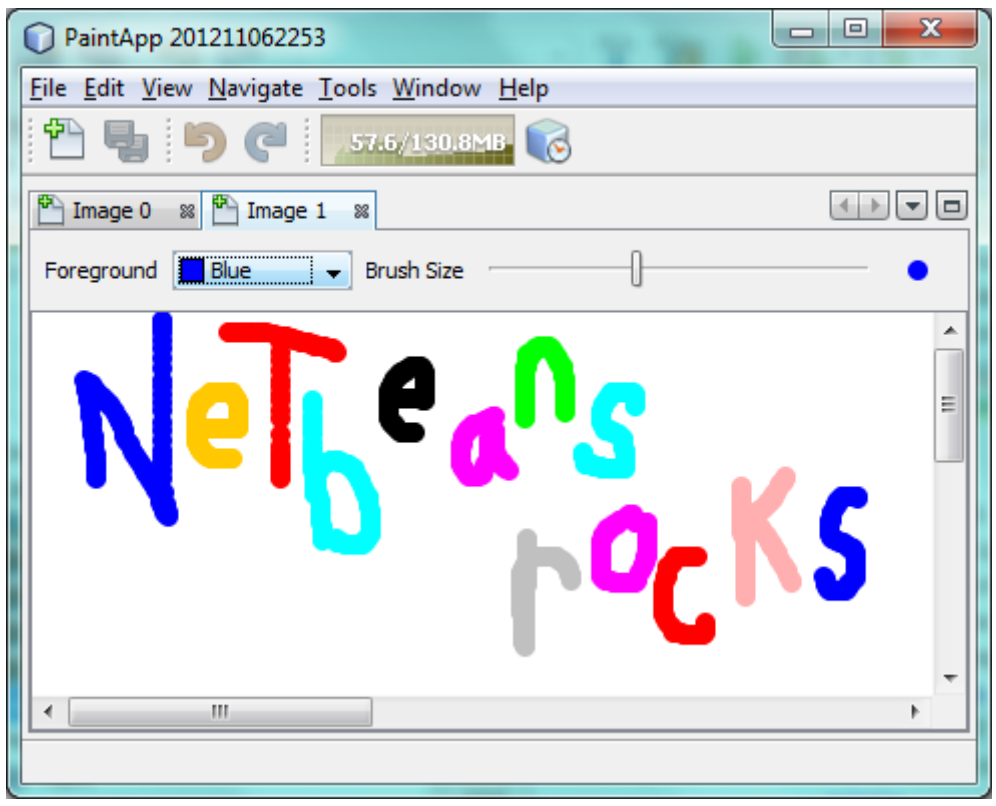
Igényeitől függően létrehozhat olyan egyéni funkciókat az alkalmazásban, amelyek integrálhatók az előző szakaszban szereplő előre definiált funkciókkal. Az egyéni funkciókat a NetBeans Platform Tanösvényen található útmutatókon keresztül ismertetjük.

Eredeti oldal

PAINT ALKALMAZÁS

Bevezetés a Paint alkalmazás elkészítéséhez

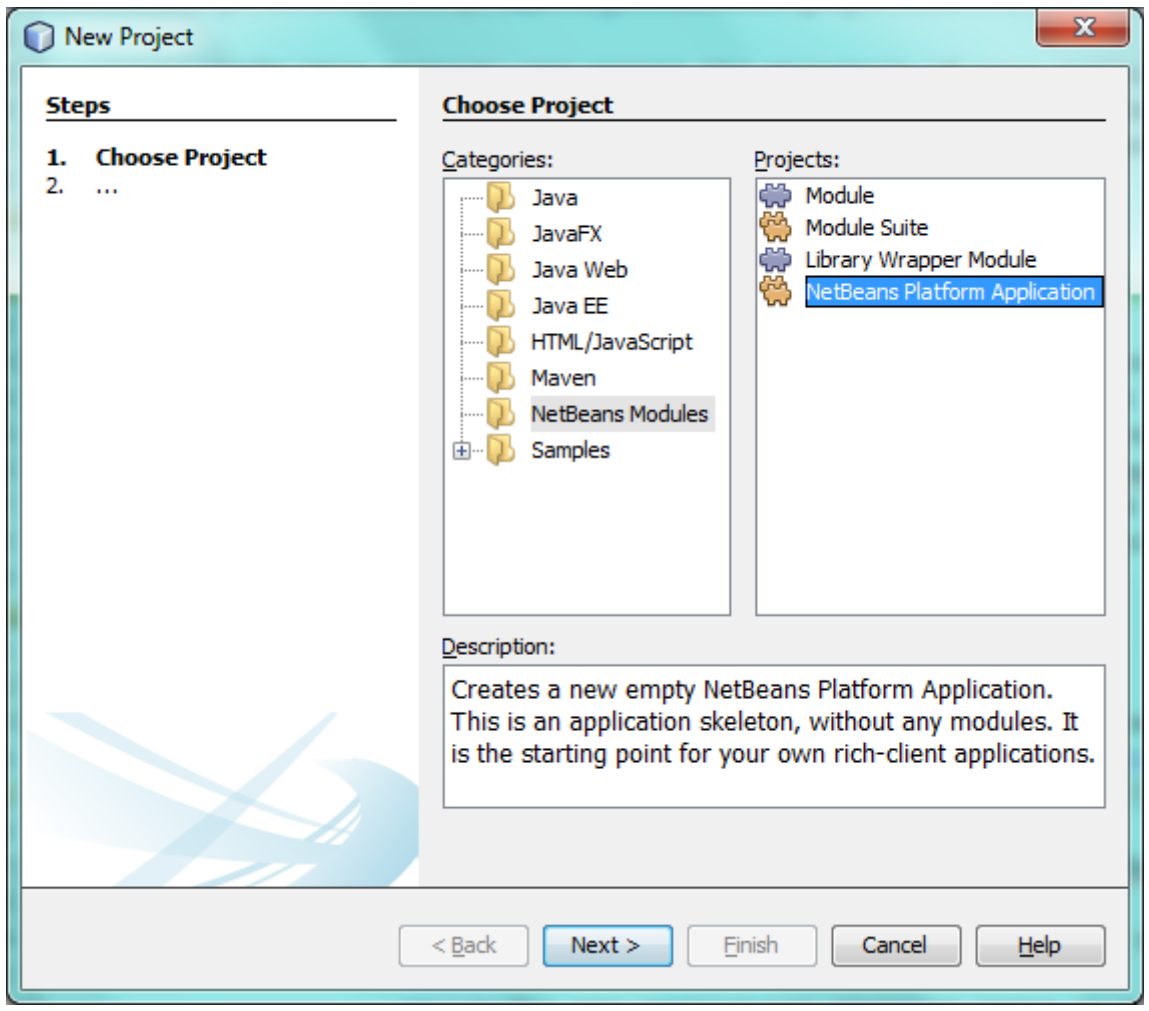
Ez a bemutató arra szolgál, hogy a lehető leggyorsabban elindulhasson. Egy egyszerű alkalmazást fog létrehozni a NetBeans Platformon. Az alkalmazás lehetővé teszi a felhasználó számára, hogy a képernyőre fessen.



Ez a kezdeti verzió messze nem egy teljes értékű Paint alkalmazás, de egy nagyon egyszerű esetet mutat be a NetBeans Platformon alapuló alkalmazás létrehozására.

Az alkalmazás projektjének a beállítása

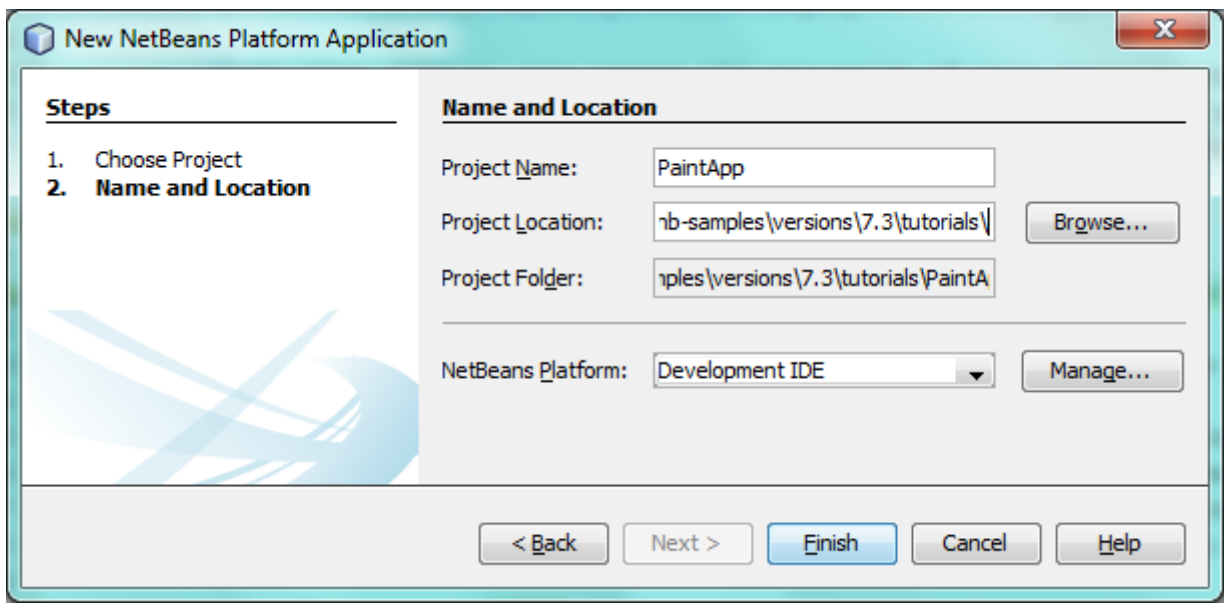
Ebben a szakaszban létrehozza az alkalmazás szerkezetét. Először is létre kell hoznia egy alkalmazás vázát, amit egy varázsló segítségével tehet meg. Ezután létrehozza a modult, amely a kódját fogja tartalmazni.



Az alkalmazás vázának a létrehozása

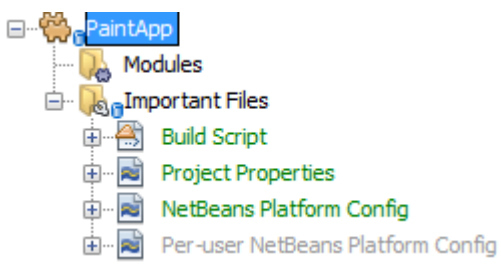
A "NetBeans Platform Application" sablon fogja létrehozni az alkalmazás vázát. A csontváz olyan modulokból áll, amelyek együttesen alkotják az alkalmazás alapját. A Projekt tulajdonságai párbeszédpanel segítségével rendelheti hozzá az alkalmazás kezdőképernyőjét, az alkalmazás nevét, valamint a használni kívánt NetBeans modulok típusát és számát. Olyan műveleteket is igénybe vehet, mint a ZIP-disztribúció létrehozása és a Java WebStart (JNLP) alkalmazás készítése, amelyek fontos eszközök az alkalmazás más felhasználók számára történő elérhetővé tételében.

1. Choose File > New Project. A Categories alatt válassza a NetBeans Modules menüpontot. Projects alatt válassza a NetBeans Platform Application lehetőséget:



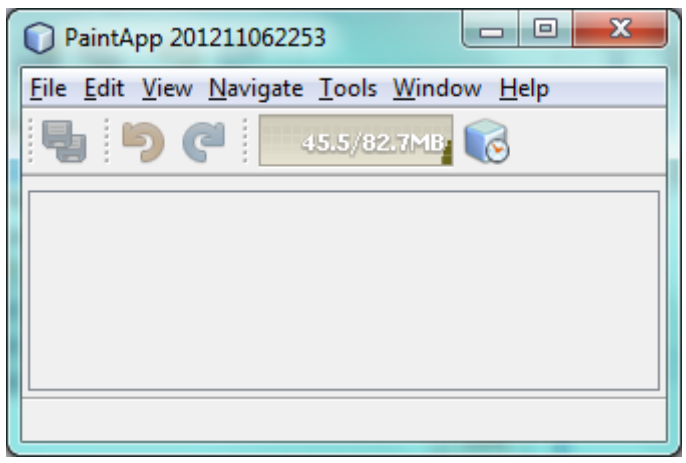
Kattintson **Next**-re.

2. A 'Name and Location' (Név és hely) panelen a 'Project Name' (Projekt neve) mezőbe írja be a PaintApp parancsot. Módosítsa a Project Location (Projekt helye) értéket a számítógépen lévő bármely könyvtárra:



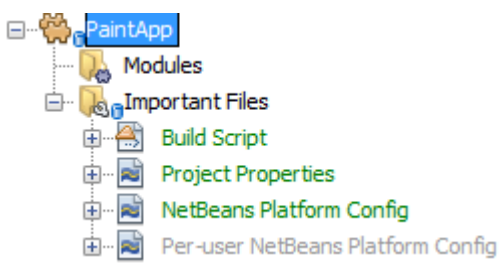
Kattintson **Finish**-re.

3. Az új alkalmazás váza megnyílik az IDE-ben. Nézze meg az új projektstruktúrát:



A Projektek ablakban két subnode-ot lát. Az első subnode, a "Modules" node mutatja az alkalmazás részét képező egyéni modulokat. Jelenleg, mint láthatja, nincs egy sem. Erre az subnode-ra jobb egérgombbal kattintva varázslókat hívhat elő új modulok létrehozására vagy külső JAR-ok alkalmazásba való beillesztésére. Az "Important Files" subnode-ban az alkalmazás által használt építési scriptek és egyéb támogató fájlok láthatók.

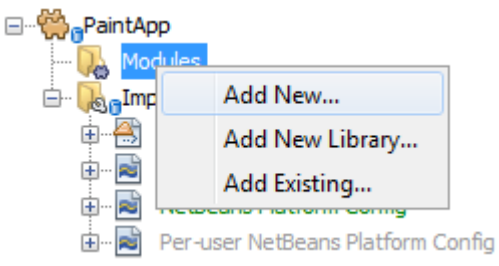
4. Kattintson a jobb egérgombbal az alkalmazásra, és válassza a Run parancsot. Megjelenik az alapértelmezett kezdőképernyő, majd megjelenik az új alkalmazás kiindulópontja:



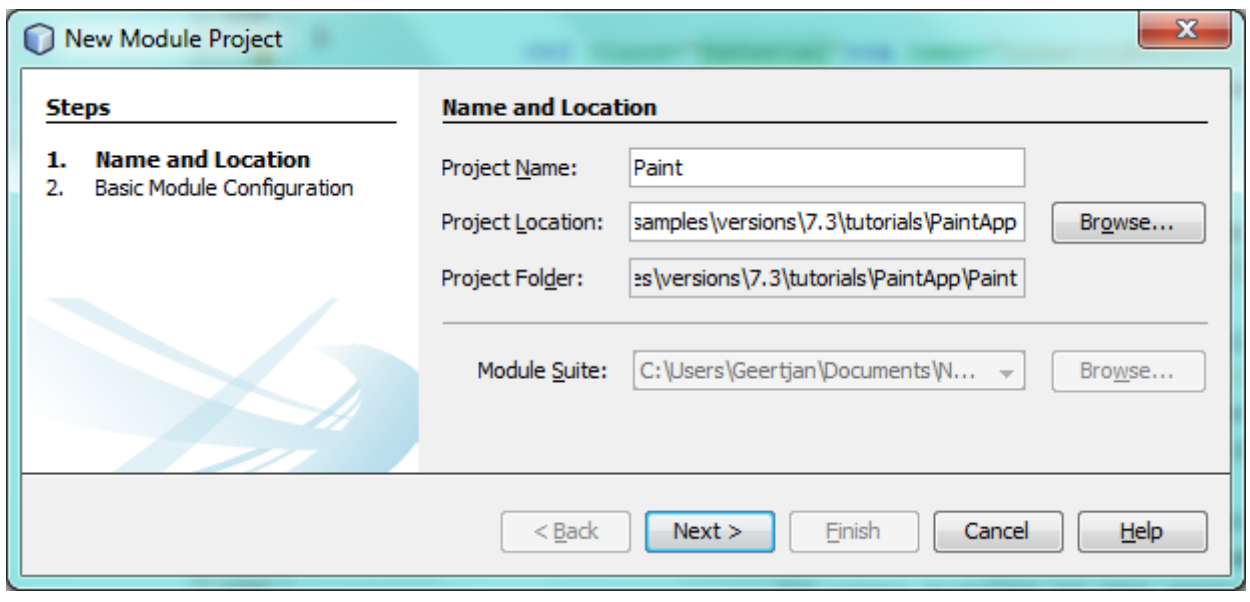
A funkcionalitás modul létrehozása

Nézze át a menüsort és az eszköztárat, hogy láthassa az alkalmazás már meglévő funkcióit, például az Exit menüpontot, a Properties ablakot és az Output ablakot. A funkcionalitás modul létrehozása Most egy modulra van szükséged, amely tartalmazza a tényleges kódot, amit írni fogsz.

1. Kattintson a jobb egérgombbal a Paint alkalmazás "Modules" subnode-jára. Válassza az Add New lehetőséget:

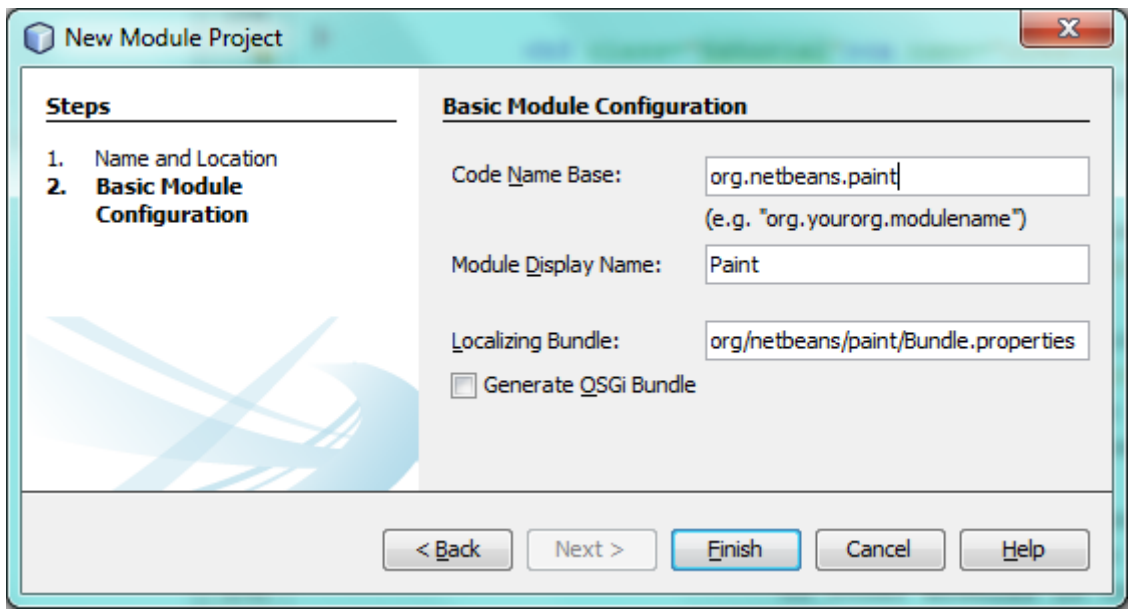


2. A Name and Location (Név és hely) panelen írja be a Paint szöveget a Project Name (Projekt neve) mezőbe.



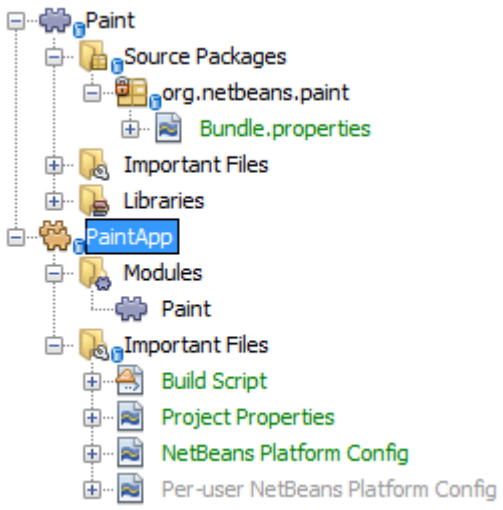
Vegyük észre, hogy a modulforrások az alkalmazás könyvtárában lévő mappában lesznek tárolva a lemezen. Kattintson a Next gombra.

3. A Basic Module Configuration panelen írja be az org.netbeans.paint kódnevet a "Code Name Base" mezőbe. A kódnévbázis egy egyedi karakterlánc, amely azonosítja a modult az alkalmazás más moduljai számára. Mindent hagyjon változatlanul.



Kattintson a Finish gombra. Az IDE létrehozza a Paint projektet.

4. Vessen egy pillantást az alkalmazás szerkezetére. A projekt tartalmazza az összes forrást és a projekt metaadatait, például a projekt Ant build scriptjét. A projekt megnyílik az IDE-ben. Logikai felépítését a Projects ablakban (Ctrl-1), fájlszerkezetét pedig a FIles ablakban (Ctrl-2) tekintheti meg. A Projects ablaknak például a következőképpen kell kinéznie:

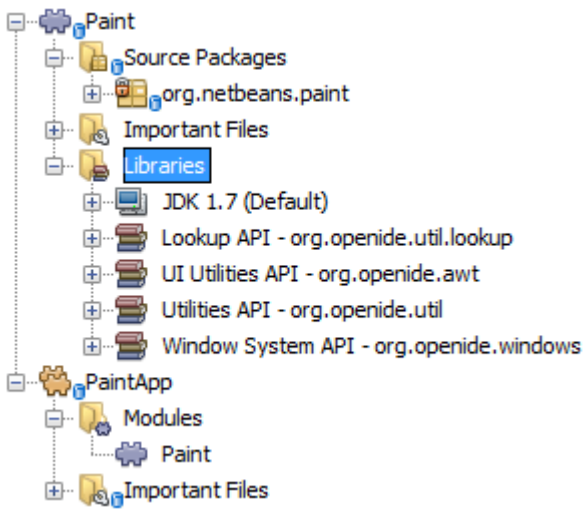


5. Több olyan osztályt kell alosztályoznia, amelyek a NetBeans API-khoz tartoznak. Minden NetBeans API-t modulok implementálnak, így ez a feladat valójában csak annyit jelent, hogy néhány modult hozzáadunk a modulok listájához, amelyekre a modulunknak szüksége van a futtatáshoz. A Projektek ablakban kattintson a jobb gombbal a Paint projekt node-ra, és válassza a Properties parancsot. Megnyílik a Project Properties párbeszédpanel. A Categories alatt kattintson a Libraries elemre. Az alábbi táblázatban felsorolt API-k mindegyikénél kattintson a "Add Dependency..." gombra, majd a Filter szövegmezőbe kezdje el beírni annak az osztálynak a nevét, amelyet alosztályozni szeretne.

Osztály	API	Célja
Lookup	Lookup API	Lehetővé teszi a modulok közötti lazán kapcsolt kommunikációt.
ActionID	UI Utilities API	Annotációkat biztosít a NetBeans Platform virtuális fájlrendszerében lévő műveletek regisztrálásához, valamint a ColorComboBox osztályhoz.
Messages	Utilities API	Különféle általános segédosztályokat biztosít, beleértve a nemzetköziesítés támogatását a Bundle osztályon és a @Messages megjegyzéseken keresztül.
TopComponent	Window System API	Hozzáférést biztosít a NetBeans ablakrendszeréhez.

A fenti táblázat első oszlopa felsorolja azokat az osztályokat, amelyeket ebben a bemutatóban alosztályozni fogsz. Minden esetben kezdje el beírni az osztály nevét a Filterbe, és figyelje, ahogy a Module lista szűkül. A táblázat második oszlopának segítségével válassza ki a megfelelő API-t (vagy a ColorChooser esetében a könyvtárat) a szűkített Module listából, majd kattintson az OK gombra a választás megerősítéséhez. Kattintson az OK gombra a Projekt tulajdonságai párbeszédpanel elhagyásához.

A Projects ablakban bontsa ki a Paint modul projekt node-ját, majd bontsa ki a Könyvtárak node-ot. Vegye észre, hogy az összes kiválasztott könyvtár megjelenik:



Bontsa ki a Paint modul Important Files node-ját, és kattintson duplán a Project Metadata node-ra. Vegye észre, hogy a kiválasztott API-kat modulfüggőségként deklarálták a fájlban. A modul fordításakor a bejelentett függőségek hozzáadásra kerülnek a modul manifest fájljához.

A festővászon létrehozása és beágyazása

A vászon létrehozása

A következő lépés a tényleges komponens létrehozása, amelyre a felhasználó festhet. Itt egy tiszta Swing-komponenst használunk - így kihagyjuk a megvalósítás részleteit, és csak a végleges változatot adjuk meg. A színválasztó babot, amelyhez létrehozta a könyvtárcsomagoló modult, ennek a panelnek a forráskódjában használjuk - amikor futtatjuk a kész alkalmazást, a képek szerkesztésére szolgáló panel eszköztárában fogjuk látni.

1. A Projects ablakban bontsa ki a Paint node-ot, majd bontsa ki a Source Packages node-ot, majd kattintson a jobb egérgombbal az org.netbeans.paint node-ra. Válassza az New > Java Class lehetőséget.

2. Adja meg a PaintCanvas osztálynevet. Győződjön meg róla, hogy az org.netbeans.paint szerepel a csomagok között. Kattintson a Finish gombra. A PaintCanvas.java megnyílik a forrásszerkesztőben.

3. Cserélje ki a fájl alapértelmezett tartalmát az itt található tartalommal. Ha a csomagot nem org.netbeans.paint-nek nevezte el, javítsa ki a csomag nevét a forrásszerkesztőben.

A vászon beágyazása egy ablakba

Most megírja az egyetlen olyan osztályt ebben az alkalmazásban, amelynek érintkeznie kell a NetBeans API-kkal. Ez egy TopComponent osztály. A TopComponent osztály nem más, mint egy JPanel osztály, amellyel a NetBeans ablakrendszer tudja, hogyan kell beszélni - így a főablakon belül egy füles konténerbe helyezhető.

1. A Projects ablakban bontsa ki a Paint node-ot, majd bontsa ki a Source Packages node-ot, majd kattintson a jobb egérgombbal az org.netbeans.paint node-ra. Válassza az New > Java Class lehetőséget. Írja be a PaintTopComponent osztálynévként. Győződjön meg róla, hogy az org.netbeans.paint szerepel a csomagként. Kattintson a Finish gombra. A PaintTopComponent.java megnyílik a forrásszerkesztőben.

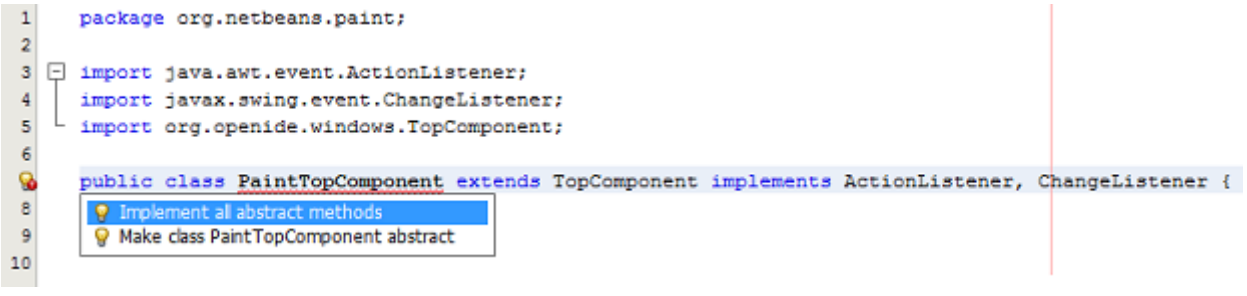
2. A fájl teteje közelében módosítsa az osztály deklarációját a következőkre:

```
public class PaintTopComponent extends TopComponent implements ActionListener, ChangeListener {
```

3. Nyomja meg a Ctrl-Shift-I billentyűt az importálás rögzítéséhez, majd kattintson az OK gombra. Az IDE elvégzi a szükséges import csomag deklarációkat a fájl tetején:

```
import java.awt.event.ActionListener;
import javax.swing.event.ChangeListener;
import org.openide.windows.TopComponent;
```

Figyelje meg a piros vonalat az osztály deklarációja alatt, amelyet az imént írt be. Helyezze a kurzort a sorba, és vegye észre, hogy a bal margón egy villanykörte jelenik meg. Kattintson az izzóra (vagy nyomja le az Alt-Enter billentyűt), ahogy az alább látható:



Válassza az Implement all abstract methods lehetőséget. Az IDE két metódus vázát generálja - actionPerformed() és stateChanged() . Ezeket később fogja kitölteni ebben a tutorialban.

4. Regisztráljuk a PaintTopComponentet az ablakrendszerben az osztály tetején lévő megjegyzések hozzáadásával, ahogy itt látható, majd nyomjuk le a Ctrl-Shift-I billentyűt, hogy az IDE generálja a megfelelő import utasításokat:

```
@TopComponent.Description
(
        preferredID = "PaintTopComponent",
        iconBase = "/org/netbeans/paint/new_icon.png",
        persistenceType = TopComponent.PERSISTENCE_ALWAYS)
@TopComponent.Registration(
        mode = "editor",
        openAtStartup = true)
@ActionID(
        category = "Window",
        id = "org.netbeans.paint.PaintTopComponent")
@ActionReferences({
@ActionReference(
        path = "Menu/Window",
        position = 0),
@ActionReference(
        path = "Toolbars/File",
        position = 0)
})
@TopComponent.OpenActionRegistration(
        displayName = "#CTL_NewCanvasAction")
@Messages({
        "CTL_NewCanvasAction=New Canvas",
        "LBL_Clear=Clear",
        "LBL_Foreground=Foreground",
        "LBL_BrushSize=Brush Size",
        "# {0} - image",
        "UnsavedImageNameFormat=Image {0}"})
public class PaintTopComponent extends TopComponent implements ActionListener, ChangeListener {
```

Adja hozzá ezt a két ikont az "org/netbeans/paint"-hez:

A 16x16 pixeles ikon a Kis eszköztár ikonok megjelenítéséhez lesz használva, míg a 24x24 pixeles ikon a Nagy eszköztár megjelenítéséhez, valamint az ablak fülén, ahogy azt a fenti @TopComponent.Description meghatározza.

6. A TopComponent osztály a korábban létrehozott Canvas osztály burkolata. Az összes új import utasítás és az alábbi kód a szokásos Java Swing kód. Másolja ki az alábbiakat, és illessze be a PaintTopComponent forrásfájljába:

```
package org.netbeans.paint;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
import javax.swing.JSlider;
import javax.swing.JToolBar;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import org.openide.awt.ActionID;
import org.openide.awt.ActionReference;
import org.openide.awt.ActionReferences;
import org.openide.awt.ColorComboBox;
import org.openide.util.NbBundle.Messages;
import org.openide.windows.TopComponent;

@TopComponent.Description(
        preferredID = "PaintTopComponent",
        iconBase = "/org/netbeans/paint/new_icon.png",
        persistenceType = TopComponent.PERSISTENCE_ALWAYS)
@TopComponent.Registration(
        mode = "editor",
        openAtStartup = true)
@ActionID(
```

```
        category = "Window",
        id = "org.netbeans.paint.PaintTopComponent")
@ActionReferences({
    @ActionReference(
        path = "Menu/Window",
        position = 0),
    @ActionReference(
        path = "Toolbars/File",
        position = 0)
})
@TopComponent.OpenActionRegistration(
        displayName = "#CTL_NewCanvasAction")
@Messages({
    "CTL_NewCanvasAction=New Canvas",
    "LBL_Clear=Clear",
    "LBL_Foreground=Foreground",
    "LBL_BrushSize=Brush Size",
    "# {0} - image",
    "UnsavedImageNameFormat=Image {0}"})
public class PaintTopComponent extends TopComponent implements ActionListener, ChangeListener {

    private PaintCanvas canvas = new PaintCanvas(); //The component the user draws on
    private final JComponent preview = canvas.getBrushSizeView(); //A component in the toolbar that shows the paintbrush size
    private final JSlider brushSizeSlider = new JSlider(1, 24); //A slider to set the brush size
    private final JToolBar toolbar = new JToolBar(); //The toolbar
    private final ColorComboBox color = new ColorComboBox(); //Our color chooser component from the ColorChooser library
    private final JButton clear = new JButton(Bundle.LBL_Clear()); //A button to clear the canvas
    private final JLabel label = new JLabel(Bundle.LBL_Foreground()); //A label for the color chooser
    private final JLabel brushSizeLabel = new JLabel(Bundle.LBL_BrushSize()); //A label for the brush size slider
    private static int ct = 0; //A counter you use to provide names for new images

    public PaintTopComponent() {
        initComponents();
        setDisplayName(Bundle.UnsavedImageNameFormat(ct++));
    }

    private void initComponents() {

        setLayout(new BorderLayout());

        //Configure our components, attach listeners:
        color.addActionListener(this);
        clear.addActionListener(this);
        brushSizeSlider.setValue(canvas.getBrushDiameter());
        brushSizeSlider.addChangeListener(this);
        color.setSelectedColor(canvas.getColor());
        color.setMaximumSize(new Dimension(16, 16));

        //Install the toolbar and the painting component:
        add(toolbar, BorderLayout.NORTH);
        add(new JScrollPane(canvas), BorderLayout.CENTER);

        //Configure the toolbar:
        toolbar.setLayout(new FlowLayout(FlowLayout.LEFT, 7, 7));
        toolbar.setFloatable(false);

        //Now populate our toolbar:
        toolbar.add(label);
        toolbar.add(color);
        toolbar.add(brushSizeLabel);
        toolbar.add(brushSizeSlider);
        toolbar.add(preview);
        toolbar.add(clear);

    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() instanceof JButton) {
            canvas.clear();
        } else if (e.getSource() instanceof ColorComboBox) {
            ColorComboBox cc = (ColorComboBox) e.getSource();
            canvas.setColor(cc.getSelectedColor());
        }
    }
```

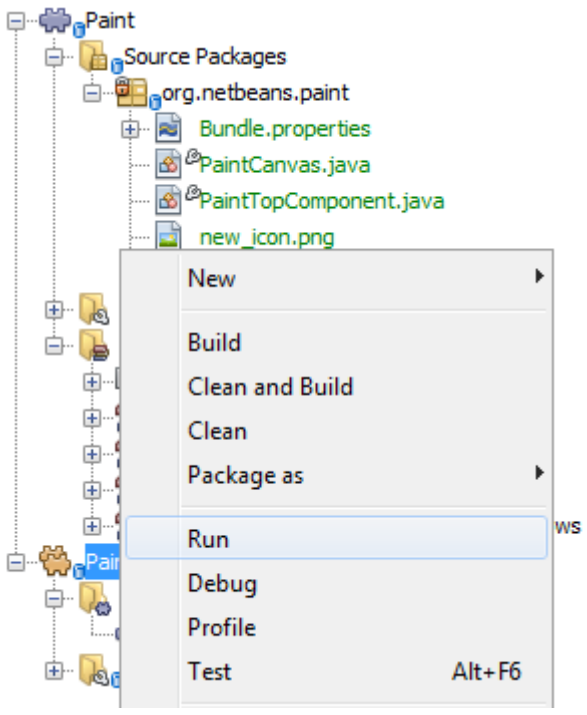
```
        @Override
        public void stateChanged(ChangeEvent e) {
            canvas.setBrushDiameter(brushSizeSlider.getValue());
        }

    }
```

Az alkalmazás futtatása, márkázása és csomagolása

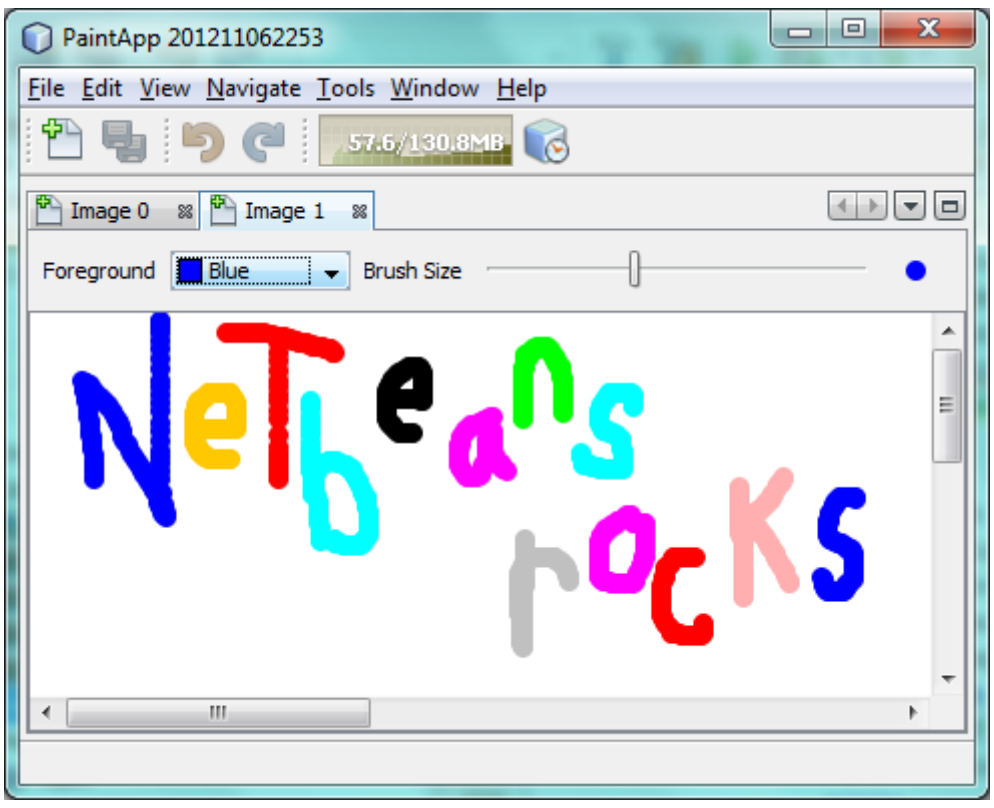
Ebben a szakaszban kipróbálja az alkalmazást, majd csomagolja azt a felhasználóknak való terjesztéshez.

Az alkalmazás futtatása

1. Kattintson a jobb egérgombbal az alkalmazásra, és válassza a Run parancsot:



2. Az alkalmazás elindul, megjelenik egy kezdőképernyő, majd megjelenik az alkalmazás. Fessünk valamit, ahogy az alább látható:

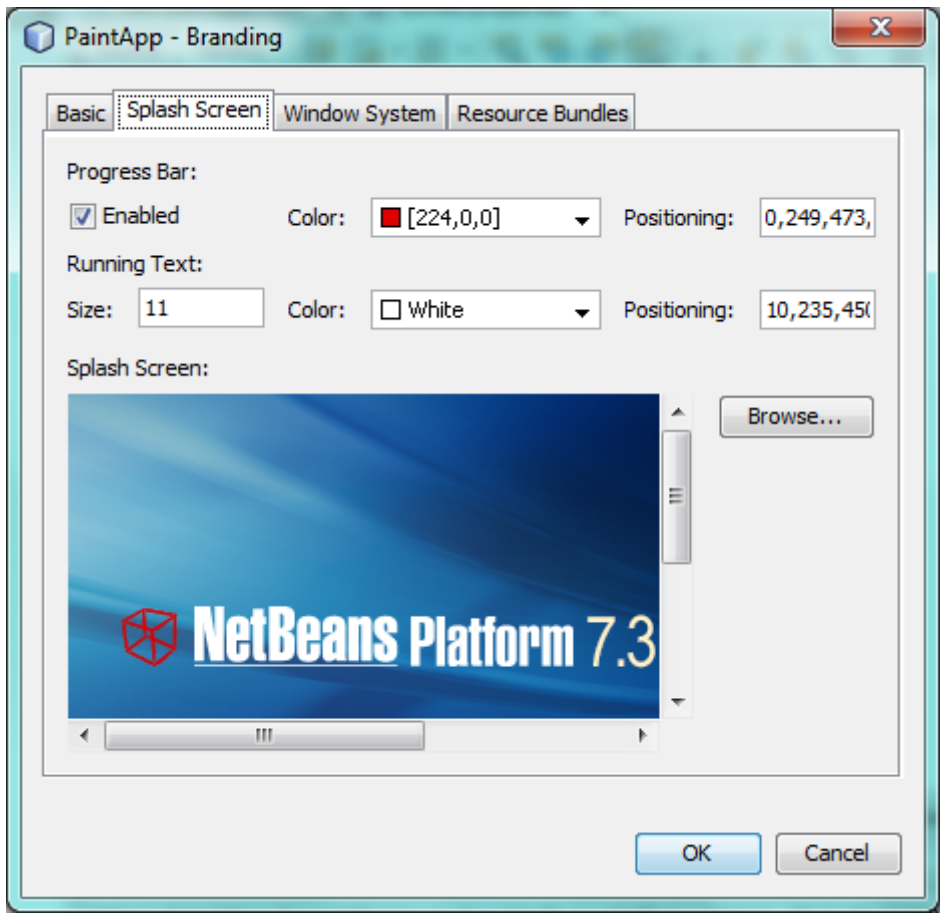


3. Használja az alkalmazást, és próbálja meg azonosítani azokat a területeket, ahol több funkciót szeretne biztosítani.

Az alkalmazás márkázása

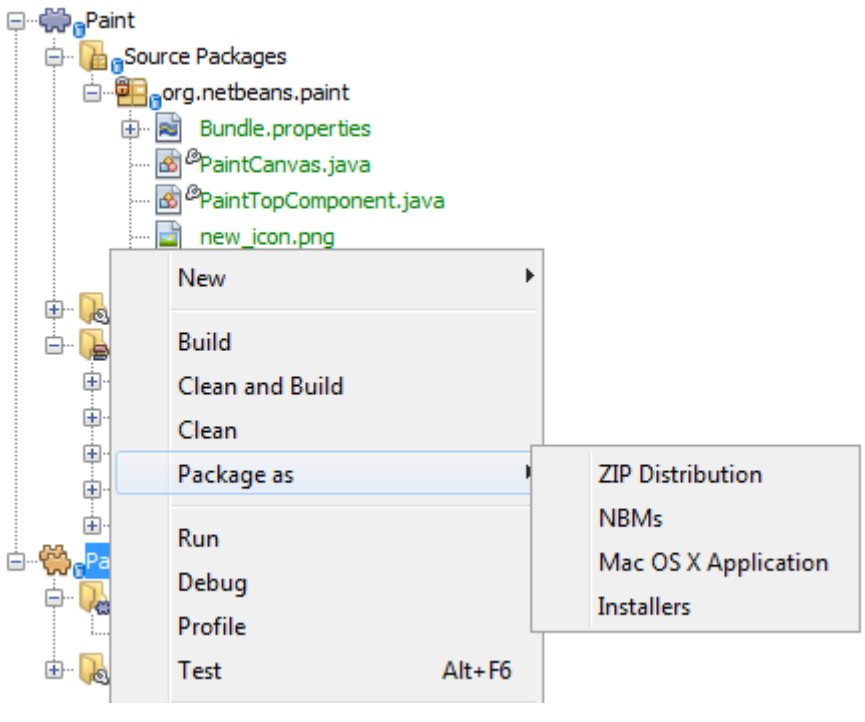
1. Kattintson a jobb egérgombbal az alkalmazásra, és válassza a Branding parancsot.ű

2. Megjelenik a Branding Window, amelynek segítségével megváltoztathatja az ikonokat, a kezdőképernyőt, az ablak jellemzőit és az alkalmazásban megjelenő karakterláncokat:



Az alkalmazás csomagolása

1. Kattintson a jobb egérgombbal az alkalmazásra, és válassza a Package as parancsot:



2. Válassza ki az üzleti igényeinek és a felhasználói követelményeknek megfelelő terjesztési mechanizmust.

3. Váltson a Files ablakra (Ctrl-2) az eredmény megtekintéséhez.

Ez az! Ön eljészítette a Paint alkalmazást. Megtanulta, hogyan kell beállítani egy NetBeans Platform alkalmazást, és hogyan kell létrehozni egy új ablakot, amely megjelenít valamit a felhasználónak.

Eredeti oldal

NetBeans kódkiegészítő bemutató

Ez a bemutató megmutatja, hogyan kell a Editor Code Completion API-t implementálni. Megmutatjuk, hogyan kell az API-t HTML-fájlok kontextusában implementálni. Amikor a felhasználó meghívja a kódkiegészítő funkciót, megjelenik egy kódkiegészítő doboz, amely a szerkesztőbe beírt szöveget kiegészítő szavakat jeleníti meg. A bemutató végén a HTML-fájlok kódkiegészítő doboza a következőképpen fog kinézni: A kódkiegészítő doboz tartalma a JDK java.util.Locale csomagjából kinyert országnevekből származik.

Bevezetés a kódkiegészítés integrációjába

Két osztály vonatkozik a kódkiegészítésre, és ezeket fogjuk kifejleszteni ebben a bemutatóban:

- CompletionItem
- CompletionProvider

Mindkettő a Editor Code Completion API-ból származik, és ebben az oktatómodulban CountriesCompletionItem és CountriesCompletionProvider néven van implementálva.

A modulprojekt létrehozása

Ebben a szakaszban egy varázslót használunk a forrásszerkezet létrehozásához, amelyre minden NetBeans modulnak szüksége van. A forrásszerkezet bizonyos mappákból áll, amelyek meghatározott helyeken vannak, valamint egy sor olyan fájlból, amelyekre mindig szükség van. Például minden Ant-alapú NetBeans modulnak szüksége van egy nbproject mappára, amely a projekt metaadatait tartalmazza.

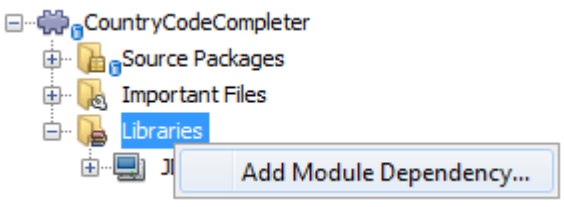
1. Válassza a File> New Project (Ctrl-Shift-N) lehetőséget. A Categoris alatt válassza a NetBeans Modules menüpontot. A Projects alatt válassza a Module lehetőséget. Kattintson a Next gombra.
2. A Name and Location panelen írja be a CountryCodeCompleter neveta Project Name mezőbe. Módosítsa a Project Locationt a számítógépen lévő bármely könyvtárra. Kattintson a Next gombra.
3. A Basic Module Configuration panelen írja be az org.netbeans.modules.countries kódnévbázist. Kattintson a Finish gombra.

Az IDE létrehozza a CountryCodeCompleter projektet. A projekt tartalmazza az összes forrást és a projekt metaadatait, például a projekt Ant build scriptjét. A projekt megnyílik az IDE-ben. Logikai szerkezetét a Projects ablakban (Ctrl-1), fájlszerkezetét pedig a Files ablakban (Ctrl-2) tekintheti meg.

A Completion Provider osztály implementálása

Az első osztály, amellyel foglalkozni fogunk, amikor HTML fájlok kódkiegészítő funkcióját készítjük el, a CompletionProvider . Ahogy a felhasználó gépel, a kódkiegészítő infrastruktúra az XML rétegfájlban regisztrált összes kódkiegészítő szolgáltatót megkéri, hogy hozzon létre CompletionTaskset . A feladatokat a CompletionProvider.createTask metódus hozza létre. Az, hogy mi történik a metódus meghívásakor, az implementációtól függ. A mi implementációnkban egy CompletionItem-et hozunk létre egy országok listájához.

1. Kattintson a jobb egérgombbal a CountryCodeCompleter projekt Libraries node-jára, és válassza a Add Module Dependency parancsot:



Set dependencies on the following:

- "Editor Code Completion", amely biztosítja azokat az API osztályokat, amelyekre ebben a bemutatóban szükségünk van.
- "MIME Lookup API", amely Java megjegyzést biztosít a kitöltési szolgáltatók regisztrálásához.
- "Utilities API".

2. Kattintson a jobb egérgombbal a CountryCodeCompleter projektre, és válassza az New > Java Class lehetőséget. Az Class Name mezőbe írja be a CountriesCompletionProvider nevet . A Package mezőben válassza az org.netbeans.modules.countries lehetőséget. Kattintson a Finish gombra.
3. A CountriesCompletionProvider osztályban módosítsa az szignatúrát úgy, hogy az osztály a CompletionProvidert valósítsa meg. Helyezze a kurzort az szignatúrát meghatározó sorra. Megjelenik egy villanykörte. Kattintson rá, és az IDE hozzáadja az org.netbeans.spi.editor.completion.CompletionProvider import utasítást. A villanykörte ismét megjelenik. Kattintson rá ismét, és az IDE létrehozza a CompletionProvider osztály által igényelt két metódus vázmetódusát. Most már ezt kell látnod:

```
package org.netbeans.modules.countries;

import javax.swing.text.JTextComponent;
import org.netbeans.spi.editor.completion.CompletionProvider;
import org.netbeans.spi.editor.completion.CompletionTask;

public class CountriesCompletionProvider implements  CompletionProvider {

    public CountriesCompletionProvider() {
    }

    @Override
    public  CompletionTask  createTask(int queryType, JTextComponent jtc) {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    @Override
    public int  getAutoQueryTypes(JTextComponent component, String typedText) {
```

```
        throw new UnsupportedOperationException("Not supported yet.");
    }

}
```

4. A CompletionProvider osztály kódolása előtt regisztráljuk azt az XML rétegfájlban, egy NetBeans Platform annotáción keresztül:

```
package org.netbeans.modules.countries;

import javax.swing.text.JTextComponent;
import org.netbeans.api.editor.mimelookup.MimeRegistration;
import org.netbeans.spi.editor.completion.CompletionProvider;
import org.netbeans.spi.editor.completion.CompletionTask;

@MimeRegistration(mimeType = "text/html", service = CompletionProvider.class)
public class CountriesCompletionProvider implements CompletionProvider {

    @Override
    public CompletionTask createTask(int queryType, JTextComponent jtc) {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    @Override
    public int getAutoQueryTypes(JTextComponent component, String typedText) {
        throw new UnsupportedOperationException("Not supported yet.");
    }

}
```

A createTask metódus implementálása

Ebben a szakaszban létrehozzuk az AsyncCompletionTask váz implementációját. A következő szakaszokban kitöltjük ezt a csontváz metódust.

1. A createTask metódusban, az előző szakasz kódja alatt a következő sorokkal egészítsük ki:

```
return new AsyncCompletionTask(new AsyncCompletionQuery() {
});
```

Itt az AsyncCompletionTask értéket adjuk vissza, ami lehetővé teszi a feladatunk aszinkron létrehozását. Az osztály az org.netbeans.spi.editor.completion.support csomagból származik, amely számos hasznos támogató osztályt biztosít a kódkiegészítési implementációk számára. Többet is használni fogunk közülük ebben a bemutatóban.

2. Helyezze a kurzort a sorra. Kattintson a megjelenő izzóra, és hagyja, hogy az IDE import utasításokat adjon hozzá. Hagyja, hogy létrehozzon egy vázmetódust a lekérdezési metódushoz.
3. Ezután meg kell adnunk, hogy milyen kódkiegészítési típussal dolgozunk. Amikor a felhasználó a Ctrl-Space billentyűre vagy egy általa meghatározott alternatív billentyűkombinációra kattint, a kódkiegészítő bejegyzéseinknek meg kell jelenniük. Ez a COMPLETION_QUERY_TYPE. Léteznek alternatív lekérdezéstípusok, mint például a DOCUMENTATION_QUERY_TYPE és a TOOLTIP_QUERY_TYPE. Meg kell vizsgálnunk, hogy a felhasználó megnyomta-e a COMPLETION_QUERY_TYPE-re vonatkozó billentyűket. Ezért a createTask metódus elejéhez adjuk hozzá a következő tesztet:

```
if (queryType != CompletionProvider.COMPLETION_QUERY_TYPE)
   return null;
```

Ebben a szakaszban a createTask metódusnak a következőképpen kell kinéznie:

```
@Override
public CompletionTask createTask(int queryType, JTextComponent jtc) {

    if (queryType != CompletionProvider.COMPLETION_QUERY_TYPE)
        return null;

    return new AsyncCompletionTask(new AsyncCompletionQuery() {
        protected void query(CompletionResultSet completionResultSet, Document document, int caretOffset) {
        }
    });

}
```

A getAutoQueryTypes metódus implementálása

Ebben a szakaszban 0-t adunk vissza AutoQueryType-ként, így a kódkiegészítő doboz nem jelenik meg automatikusan, hanem csak akkor, ha a felhasználó kéri. Mielőtt kitöltenénk a lekérdezési metódust, nézzük meg a getAutoQueryTypes(JTextComponent jTextComponent, String string) metódust. Ez a metódus határozza meg, hogy a kódkiegészítő doboz automatikusan megjelenjen-e vagy sem. Egyelőre adjuk vissza a 0 értéket. Ez azt jelenti, hogy a kódkiegészítő doboz soha nem jelenik meg, hacsak a felhasználó kifejezetten nem kéri. Így ennek a metódusnak most a következőképpen kell kinéznie:

```
@Override
public int getAutoQueryTypes(JTextComponent component, String string) {
    return 0;
}
```

Alapértelmezetten a felhasználó a Ctrl-Space billentyűt nyomja le, hogy megjelenjen a kódkiegészítő mező. Később egy új opciót adhatunk az Options ablakbővítményünkhöz, például egy jelölőnégyzetet, amely a metódusban visszaadott int értéket 0-ról 1-re változtatja, így a kódkiegészítő doboz automatikusan megjelenik. (Más típusú lekérdezések is léteznek, ahogy itt látható).

A Completion item osztály implementálása

Ebben a szakaszban létrehozunk egy olyan osztályt, amely implementálja a CompletionItem-et . Miután definiáltuk ezt az osztályt, kitöltjük a CompletionProvider osztály lekérdezési metódusát. A CompletionProvider fogja létrehozni a CompletionItem példányait.

1. Kattintson a jobb egérgombbal a CountryCodeCompleter projektre, és válassza az New > Java Class parancsot. Az Project Name mezőbe írja be a CountriesCompletionItem nevet . A Package mezőben válassza az org.netbeans.modules.countries-t. Kattintson a Finish gombra.
2. Erre az osztályra a későbbi lépésekben még visszatérünk. Most a CompletionProvider osztályban definiált lekérdezési metódust fogjuk kitölteni. Töltsük ki az AsyncCompletionTask-ot a következőképpen, és figyeljük meg a kódban található magyarázó megjegyzéseket.

```
return new  AsyncCompletionTask(new AsyncCompletionQuery() {

    @Override
    protected void query(CompletionResultSet completionResultSet, Document document, int caretOffset) {

        //Iterate through the available locales
        //and assign each country display name
        //to a CompletionResultSet:
        Locale[] locales = Locale.getAvailableLocales();
        for (int i = 0; i < locales.length; i++) {
            final Locale locale = locales[i];
            final String country = locale.getDisplayCountry();
            if (!country.equals("")) {
                completionResultSet.addItem(new CountriesCompletionItem(country, caretOffset));
            }
        }

        completionResultSet.finish();
    }

}, jtc);
```

Egy piros aláhúzás marad, miután az IDE hozzáadta a különböző import utasításokat. Az aláhúzás azt jelzi, hogy a CompletionItem konstruktora nem várja el a neki átadott értékeket. A következő lépésben kitöltjük a CompletionItemet, hogy megfeleljen a CompletionProvider követelményeinek.

3. A CountriesCompletionItem osztályban módosítsa az szignatúrát úgy, hogy az osztály a CompletionItem-et valósítsa meg. Hagyja, hogy az IDE létrehozza az osztály szükséges metódusainak importálási utasításait és vázimplementációit. Olvassa el a NetBeans Javadoc bejegyzését a CompletionItem osztályhoz, hogy megértse, mire szolgálnak az egyes metódusok. Egyelőre egy minimális kiegészítési elemet fogunk implementálni, ami éppen elég ahhoz, hogy lefordíthassuk a modult, és megjelenjen a kódkiegészítő doboz.
4. A CountriesCompletionItem osztályban definiálja a konstruktort a következőképpen:

```
private String text;
private static ImageIcon fieldIcon =
        new ImageIcon(ImageUtilities.loadImage("org/netbeans/modules/countries/icon.png"));
private static Color fieldColor = Color.decode("0x0000B2");
private int caretOffset;

public CountriesCompletionItem(String text, int caretOffset) {
    this.text = text;
```

```
        this.caretOffset = caretOffset;
    }
```

Vegye figyelembe, hogy itt egy ikonra hivatkozunk. Ez az ikon jelenik meg a kódkiegészítő mezőben a CompletionItem által képviselt minden egyes bejegyzés mellett. Az ikon bármilyen 16x16 pixeles méretű ikon lehet. Például használhatja ezt az ikont: Ha szeretné, kattintson a jobb egérgombbal a fenti képre, és mentse el a fenti ImageIcon definícióban megadott helyre.

5. Ezután definiáljuk a getPreferredWidth() és a render() metódusokat a következőképpen:

```
    @Override
    public int getPreferredWidth(Graphics graphics, Font font) {
        return CompletionUtilities.getPreferredWidth(text, null, graphics, font);
    }

    @Override
    public void render(Graphics g, Font defaultFont, Color defaultColor,
            Color backgroundColor, int width, int height, boolean selected) {
        CompletionUtilities.renderHtml(fieldIcon, text, null, g, defaultFont,
                (selected ? Color.white : fieldColor), width, height, selected);
    }
```

Definiálja a getSortText() metódust a következőképpen:

```
    @Override
    public CharSequence getSortText() {
        return text;
    }
```

Ezután definiáljuk a getInsertPrefix() metódust:

```
    @Override
    public CharSequence getInsertPrefix() {
        return text;
    }
```

Végezetül hozza létre a fennmaradó metódusok dummy implementációit. Tehát a createDocumentationTask() és a createToolTipTask() metódusok esetében adjuk vissza a null értéket. Ezután az instantSubstitution() esetében adja vissza a false értéket, a getSortPriority() esetében pedig a 0 értéket. Végül ürítse ki a defaultAction és a processKeyEvent metódusokat .

6. Kattintson a jobb egérgombbal a modulra, és válassza a Run parancsot. Az IDE új példánya elindul, és telepíti a modult. Nyisson meg egy HTML-fájlt az IDE-ben. Írjon be valamit, és nyomja le a Ctrl-Space billentyűt. Most már a következőt kell látnia:

Ha a fenti listában megnyomja az Entert, semmi sem történik. Ez azért van, mert még nem definiáltuk a defaultAction() metódust. Ezt a következő szakaszban fogjuk megtenni. Figyeljük meg azt is, hogy a lista nem szűkül, miközben gépelünk. Ez azért van, mert még nem hoztunk létre szűrőt. A szűrő érzékelni fogja, hogy mit gépelünk, és ennek megfelelően fogja módosítani a listában lévő bejegyzéseket. A szűrőt egy későbbi szakaszban fogjuk létrehozni.

Az Action implementálása

Ebben a szakaszban megadjuk, hogy mi történik, amikor a felhasználó megnyomja az Enter billentyűt vagy az egérrel a kódkiegészítő mezőben lévő bejegyzés fölé kattint.

1. Töltse ki a defaultAction() metódust a következőképpen:

```
@Override
public void defaultAction(JTextComponent jtc) {
    try {
        StyledDocument doc = (StyledDocument) jtc.getDocument();
        doc.insertString(caretOffset, text, null);
        //This statement will close the code completion box:
        Completion.get().hideAll();
    } catch (BadLocationException ex) {
        Exceptions.printStackTrace(ex);
    }
}
```

2. Telepítse ismét a modult. Figyelje meg, hogy amikor megnyomja az Enter billentyűt vagy az egérrel a kódkiegészítő mezőben lévő bejegyzés fölé kattint, a kiválasztott szöveg a kurzorral a HTML-fájlba kerül. A kódkiegészítő mező meghívása előtt begépelt szöveg azonban nem kerül eltávolításra. Az alábbiakban a "V"-t el kell távolítani, mivel a kódkiegészítő mezőben a "Vietnam" volt kiválasztva:

```
<p>Good morning VVietnam
```

A következő szakaszban olyan funkciót adunk hozzá, amely érzékeli a beírt karakterek számát, és eltávolítja őket, amikor a kiválasztott országot beillesztjük a dokumentumba.

A szűrő implementálása

Ebben a szakaszban engedélyezzük, hogy a kódkiegészítő mező szűküljön, miközben a felhasználó gépel. Így amikor a felhasználó beírja a 'hel' betűt, csak az ezekkel a betűkkel kezdődő szavak jelennek meg a kódkiegészítő mezőben. A szűrőt a CountriesCompletionProvider osztályban definiáljuk.

1. A CountriesCompletionProvider osztályban írja át az AsyncCompletionTask() metódust az alábbi félkövérrel kiemelt utasításokkal:

```
return new AsyncCompletionTask(new AsyncCompletionQuery() {

    @Override
    protected void query(CompletionResultSet completionResultSet, Document document, int caretOffset) {

        String filter = null;
        int startOffset = caretOffset - 1;

        try {
            final StyledDocument bDoc = (StyledDocument) document;
            final int lineStartOffset = getRowFirstNonWhite(bDoc, caretOffset);
            final char[] line = bDoc.getText(lineStartOffset, caretOffset - lineStartOffset).toCharArray();
            final int whiteOffset = indexOfWhite(line);
            filter = new String(line, whiteOffset + 1, line.length - whiteOffset - 1);
            if (whiteOffset > 0) {
                startOffset = lineStartOffset + whiteOffset + 1;
            } else {
                startOffset = lineStartOffset;
            }
        } catch (BadLocationException ex) {
            Exceptions.printStackTrace(ex);
        }

        //Iterate through the available locales
        //and assign each country display name
        //to a CompletionResultSet:
        Locale[] locales = Locale.getAvailableLocales();
        for (int i = 0; i < locales.length; i++) {
            final Locale locale = locales[i];
            final String country = locale.getDisplayCountry();
            //Here we test whether the country starts with the filter defined above:
            if (!country.equals("") && country.startsWith(filter)) {
                //Here we include the start offset, so that we'll be able to figure out
                //the number of characters that we'll need to remove:
                completionResultSet.addItem(new CountriesCompletionItem(country, startOffset, caretOffset));
            }
        }
        completionResultSet.finish();

    }
```

```
}, jtc);
```

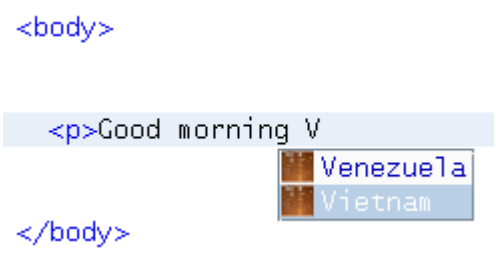
2. A CountriesCompletionProvider végére adjuk hozzá a következő két metódust:

```
static int getRowFirstNonWhite(StyledDocument doc, int offset)
throws BadLocationException {
    Element lineElement = doc.getParagraphElement(offset);
    int start = lineElement.getStartOffset();
    while (start + 1 < lineElement.getEndOffset()) {
        try {
            if (doc.getText(start, 1).charAt(0) != ' ') {
                break;
            }
        } catch (BadLocationException ex) {
            throw (BadLocationException)new BadLocationException(
                    "calling getText(" + start + ", " + (start + 1) +
                    ") on doc of length: " + doc.getLength(), start
                    ).initCause(ex);
        }
        start++;
    }
    return start;
}


static int indexOfWhite(char[] line){
    int i = line.length;
    while(--i > -1){
        final char c = line[i];
        if(Character.isWhitespace(c)){
            return i;
        }
    }
    return -1;
}
```

3. Módosítsa a CountriesCompletionItem konstruktorát úgy, hogy megkapja a start offsetet. Ezután módosítsa a defaultActiont úgy, hogy a start offsetet használja a kiválasztott ország beillesztésekor eltávolítandó karakterek meghatározásához. Az alábbiakban a következő utasításokat kell hozzáadni: private int dotOffset;

```
public CountriesCompletionItem(String text, int dotOffset, int caretOffset) {
    this.text = text;
    this.dotOffset = dotOffset;
    this.caretOffset = caretOffset;
}

@Override
public void defaultAction(JTextComponent component) {
    try {
        StyledDocument doc = (StyledDocument) component.getDocument();
        //Here we remove the characters starting at the start offset
        //and ending at the point where the caret is currently found:
        doc.remove(dotOffset, caretOffset-dotOffset);
        doc.insertString(dotOffset, text, null);
        Completion.get().hideAll();
    } catch (BadLocationException ex) {
        Exceptions.printStackTrace(ex);
    }
}

...
...
...
```

4. Telepítse ismét a modult, és vegye észre, hogy ezúttal a szavak listája szűkül, miközben gépel...



...és amikor megnyomja az Entert, a beírt karaktereket eltávolítja, és a kódkiegészítő mezőben kiválasztott országgal helyettesíti.

Az eszköztár és a dokumentációs feladat implementálása

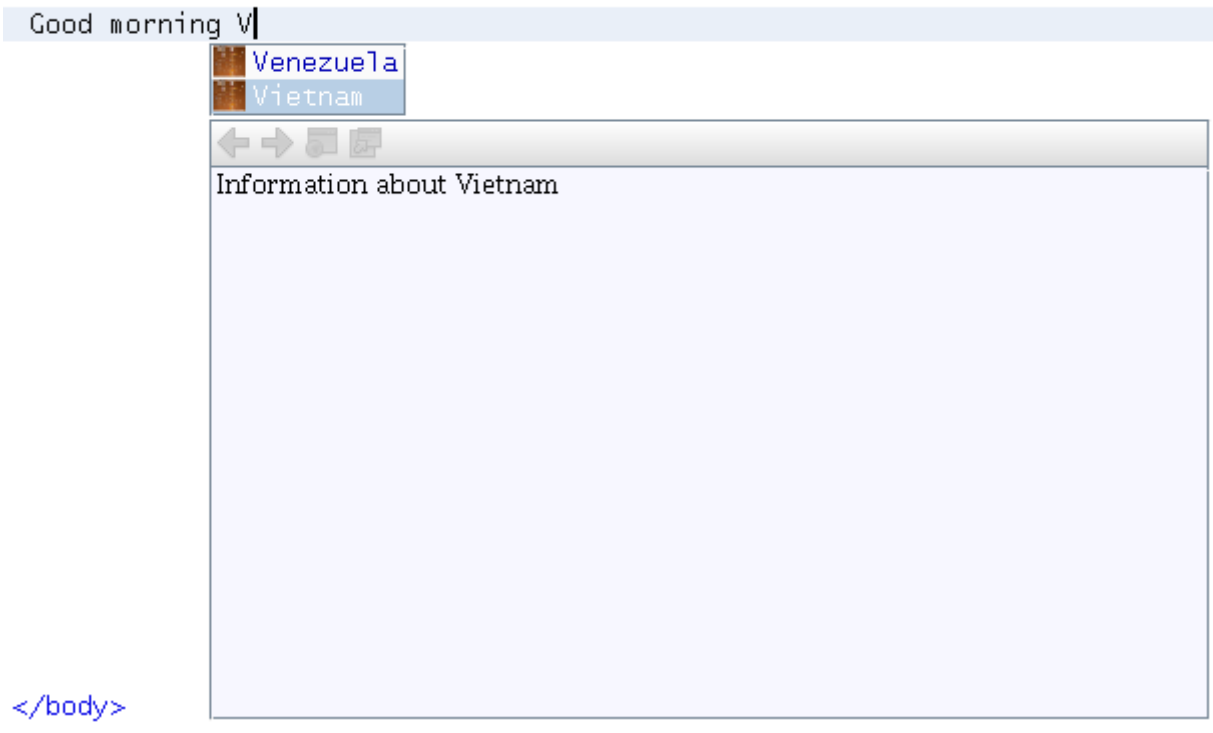
Az alábbiakban leírtak szerint néhány opcionális funkció is hozzáadható.

1. Opcionálisan implementálhatja a createToolTipTask metódust a CountriesCompletionItemben , ezzel, ha a Ctrl-P billentyűt lenyomja:

Íme a kód, amely a fenti képernyőképen látható eredményt fogja elérni:

```
@Override
public CompletionTask createToolTipTask() {
    return new AsyncCompletionTask(new AsyncCompletionQuery() {
        @Override
        protected void query(CompletionResultSet completionResultSet, Document document, int i) {
            JToolTip toolTip = new JToolTip();
            toolTip.setTipText("Press Enter to insert \"" + text + "\"");
            completionResultSet.setToolTip(toolTip);
            completionResultSet.finish();
        }
    });
}
```

2. Opcionálisan dokumentációt is megadhat a kódkiegészítő mezőben szereplő bejegyzésekhez: A dokumentációs mezőt így használhatja, ha az CountriesCompletionItem osztályban implementálja a createDocumentationTask metódust:



A fenti kódban a CountriesCompletionDocumentation osztályra való hivatkozás a következőképpen valósítható meg:

```
public class CountriesCompletionDocumentation implements CompletionDocumentation {

    private CountriesCompletionItem item;

    public CountriesCompletionDocumentation(CountriesCompletionItem item) {
        this.item = item;
    }

    @Override
    public String getText() {
```

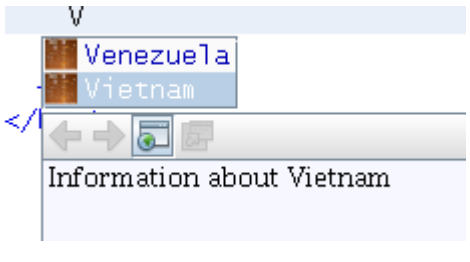
```
            return "Information about " + item.text;
        }

        @Override
        public URL getURL() {
            return null;
        }

        @Override
        public CompletionDocumentation resolveLink(String string) {
            return null;
        }

        @Override
        public Action getGotoSourceAction() {
            return null;
        }

    }
```

A fenti kódban a getURL() végrehajtásával engedélyezheti az URL gombot, ahogy az alább látható:

Amikor a felhasználó az URL gombra kattint, az IDE-ben beállított böngésző megnyílik, és megjeleníti a megadott URL által megadott tartalmat. Gratulálunk, most már elkészült a kódkiegészítő integrációs modul egyszerű implementációjával.

NetBeans kódsablon modul bemutató

Ez a bemutató azt mutatja be, hogyan hozhatunk létre egy NetBeans modult, amely kódsablonokat biztosít. A felhasználó az Options ablakban változtatásokat végezhet, akár a meglévő kódsablonok testreszabása, akár újak hozzáadása érdekében:



Ennek a funkciónak a biztosításához egyáltalán nem kell Java kódot használnia. Ahogyan ebben a bemutatóban is látható, csak egy XML-fájlt kell biztosítania, amely meghatározza a kódsablonokat, a modul layer.xml fájljában történő regisztrálással.

A modulprojekt beállítása

Mielőtt elkezdené a modul írását, meg kell győződnie arról, hogy a projektje helyesen van beállítva. Az IDE biztosít egy varázslót, amely beállítja a modulhoz szükséges összes alapfájlt.

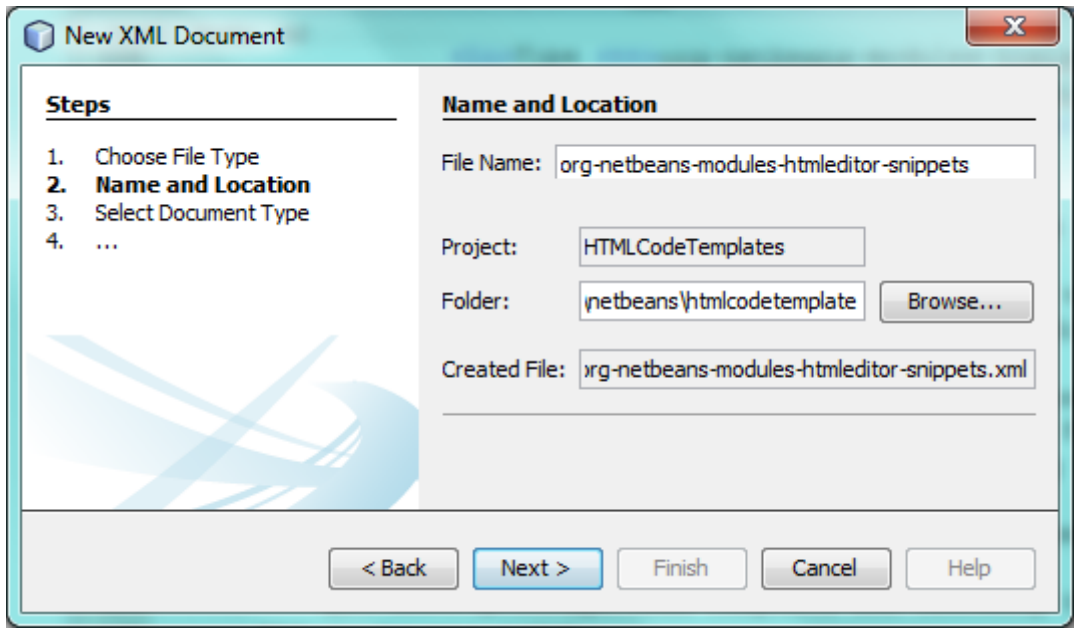
1. Válassza a New > New Project (Ctrl+Shift+N) lehetőséget. A Categories alatt válassza a NetBeans Modules menüpontot. A Projects alatt válassza a Module lehetőséget. Kattintson a Next gombra.
2. A Name and Location panel Project Name mezőjébe írja be a HTMLCodeTemplates szöveget. A Project Location a számítógépen lévő bármely könyvtárra módosítható. Kattintson a Next gombra.
3. Az Basic Module Configuration panelen írja be az org.netbeans.htmlcodetemplate kódnevet a Kódnévbázisba. Kattintson a Finish gombra.

Az IDE létrehozza a HTMLCodeTemplates projektet. A projekt tartalmazza az összes forrást és a projekt metaadatait, például a projekt Ant-építési szkriptjét. A projekt megnyílik az IDE-ben. Logikai felépítését a Projects ablakban (Ctrl-1), fájlszerkezetét pedig a Files ablakban (Ctrl-2) tekintheti meg.

A kódsablonok létrehozása

Ebben a szakaszban hozza létre a HTML-fájlok kódsablonjait.

1. Kattintson a jobb egérgombbal az org.netbeans.modules.htmlcodetemplate node-ra, majd válassza az New > Other > XML > XML Document parancsot. Kattintson a Next gombra.
2. Írja be az org-netbeans-modules-htmleditor-snippets nevet, és a "Folder"-ben keresse meg a "htmlcondetemplate" mappát:



Kattintson a Next, majd a Finish gombra.

3. Cserélje ki az org-netbeans-modules-htmleditor-snippets.xml fájl alapértelmezett tartalmát a következőkre:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE codetemplates PUBLIC "-//NetBeans//DTD Editor Code Templates settings 1.0//EN" "https://netbeans.org/dtds/EditorCodeTemplates-1_
<codetemplates>
    <codetemplate abbreviation="snf" xml:space="preserve">
        <code><![CDATA[Select "New File" (Ctrl-N) in the File menu]]></code>
        <description><![CDATA[Code template for selecting "New File".]]></description>
    </codetemplate>
    <codetemplate abbreviation="pe" xml:space="preserve">
        <code><![CDATA[Press Enter.]]></code>
        <description><![CDATA[Code template for clicking the "Enter" key.]]></description>
    </codetemplate>
</codetemplates>
```

Önnek most már két kódsablonja van. Az első az "snf" és a bővítő billentyű beírása után jelenik meg, míg a második a "pe" beírását követeli meg a bővítő billentyű megnyomása előtt.

A kódsablonok deklarálása és regisztrálása

A kódsablonok a layer.xml fájlban vannak regisztrálva annak a MIME-típusnak a számára, amelyre vonatkoznak.

1. Kattintson a jobb egérgombbal az org.netbeans.modules.htmlcodetemplate node-ra, és válassza az New > Other > Module Development > XML Layer lehetőséget. Kattintson a Next, majd a Finish gombra.
2. Cserélje ki a layer.xml fájl tartalmát a következőkre:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE filesystem PUBLIC "-//NetBeans//DTD Filesystem 1.2//EN" "https://netbeans.org/dtds/filesystem-1_2.dtd">
<filesystem>
    <folder name="Editors">
        <folder name="text">
            <folder name="html">
                <folder name="CodeTemplates">
                    <file name="org-netbeans-modules-htmleditor-snippets.xml" url="org-netbeans-modules-htmleditor-snippets.xml"/>
                </folder>
            </folder>
        </folder>
    </folder>
</filesystem>
```

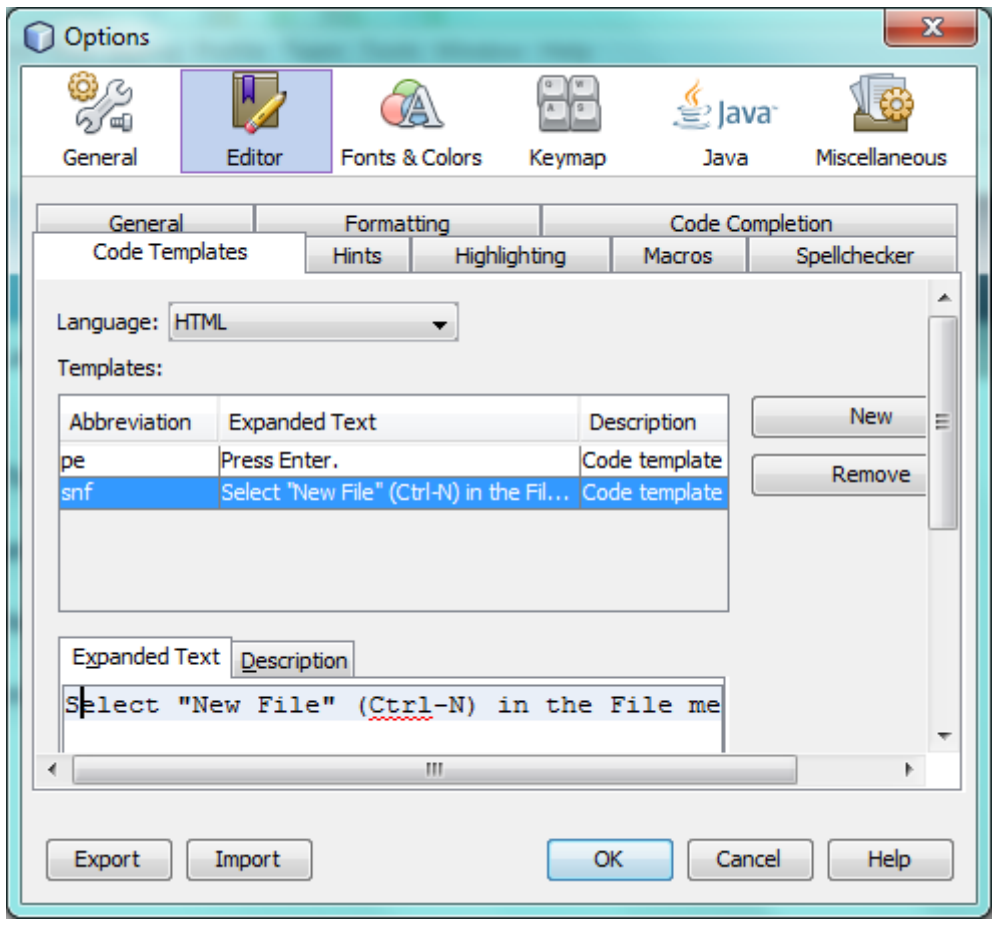
A kódsablonok létrehozása és telepítése

Most a telepítésre és a terjesztésre kell gondolnunk. Az alábbi első részben telepítjük a kódsablonokat, majd létrehozunk egy NBM fájlt és megvizsgáljuk a terjesztési csatornákat.

A kódsablonok kipróbálása

Telepítse és próbálja ki a kódsablonokat az alábbi lépéseket követve.

1. A Projects ablakban kattintson a jobb egérgombbal a HTMLCodeTemplates projektre, és válassza a Run lehetőséget. A modul elkészül és települ a célplatformra. A célplatform megnyílik, így kipróbálhatja az új modult. Az alapértelmezett célplatform a fejlesztői IDE aktuális példánya által használt telepítés.
2. Nézze meg a Options ablakot, és vegye észre a két kódsablont:



3. Nyisson meg egy HTML-fájlt, és írja be az "snf" szót. Nyomja meg a Tab billentyűt, és vegye észre, hogy a rövidítés kibővül. Ugyanígy járjon el a "pe" rövidítéssel is.

Megosztható bináris modul létrehozása

Az NBM fájl a modul bináris változata, amely a kódsablonokat biztosítja. Az alábbiakban egy menüpont segítségével létrehozzuk az NBM fájlt.

1. A Projects ablakban kattintson a jobb egérgombbal a HTMLCodeTemplates projektre, és válassza az Create NBM parancsot. Az NBM fájl létrejön, és a Files ablakban (Ctrl-2) megtekinthető.
2. Tegye elérhetővé a modult mások számára például a Plugin Portálon keresztül.
3. A címzett a modult az IDE Plugin Manager segítségével telepítheti. A főmenüben az Tools > Plugins menüpontot kell választaniuk. Küldje el nekünk visszajelzését!

> Fordította: Kulman Ferdinánd és Kovács Kornél