

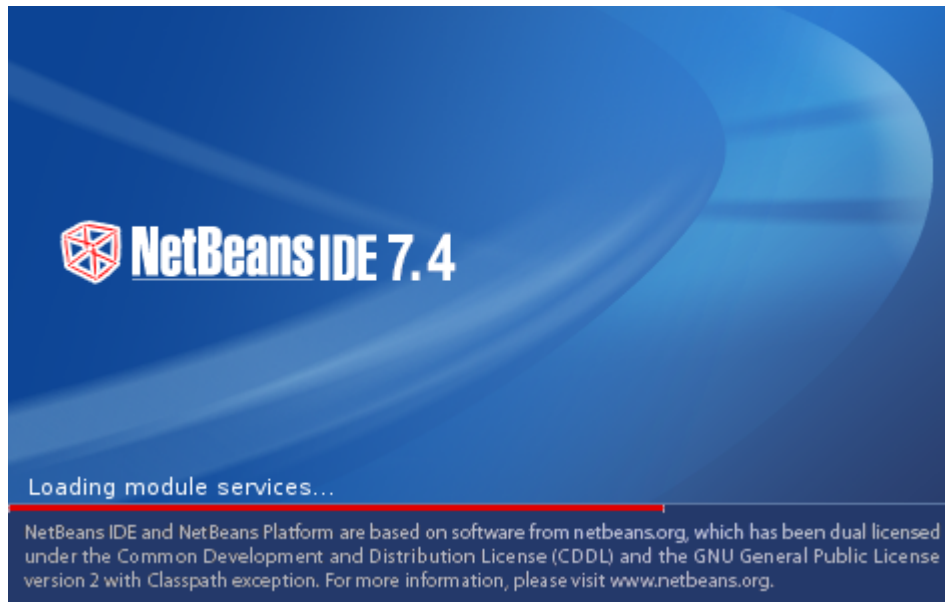
1. Bevezet

A **Java-Suli** sorozat elssorban az általam oktatott diákoknak szólt, mivel az óráimon eladott anyagok rendezetlen gyjteményét hoztam egységes formába, illetve – az elmúlt három év tapasztalata alapján – könnyen emészthet sorrendbe. A tanulóim mindegyike *többé-kevésbé* ismerte a C és a C++ nyelvet, így ezen tudás vagy tapasztalat nélkül az oldal *látogatói* számára kissé nagy falat lehet a nyelv megtanulása. Más nyelv (**Delphi**, **Visual Basic**, stb.) ismerete kissé *megkönnyítheti* a tanulást.

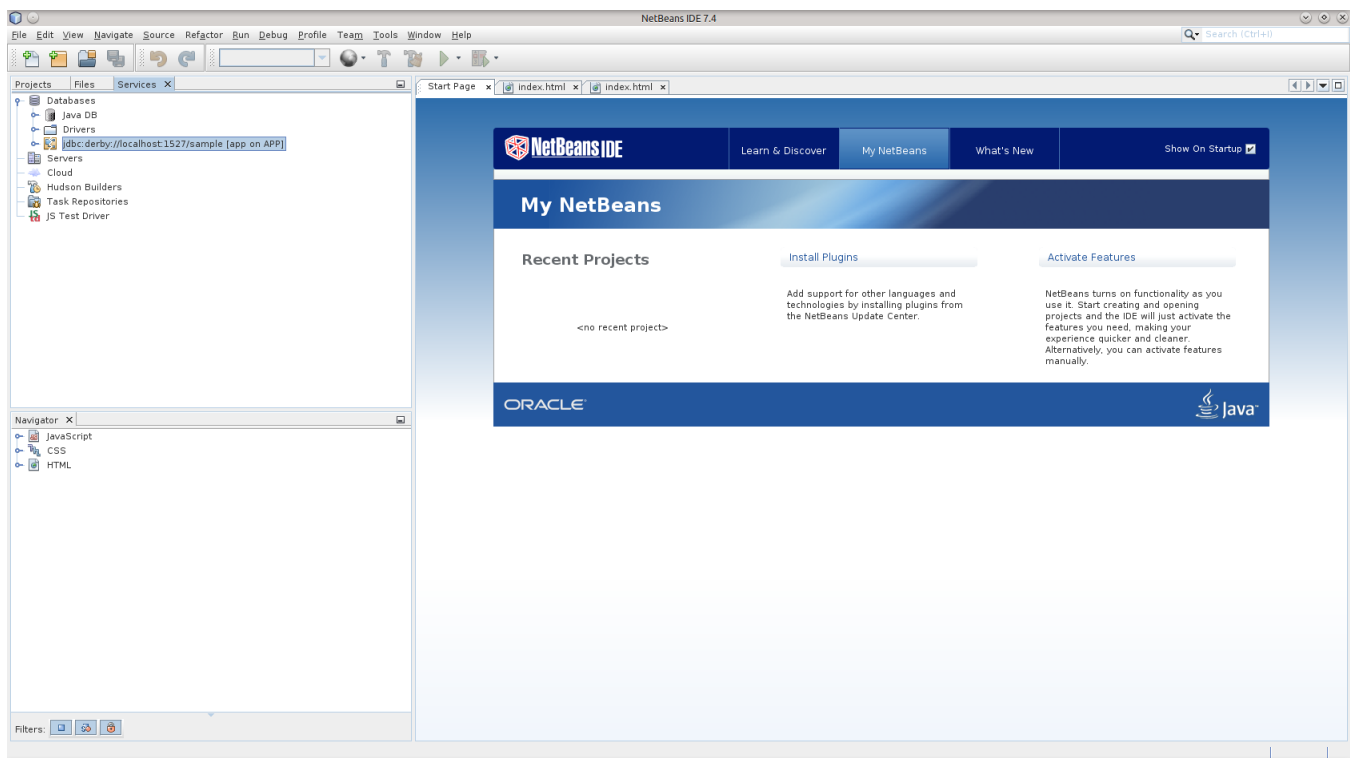
1.1. Fejlesztői környezetek

Sok olyan programozóval találkoztam, akik egy [notepad](#) összetettség vagy [vi](#) illetve [emacs](#) jelleg környezetre esküsznek, mint *egyetlen és igaz* út. Hagyjuk rájuk ezt a rigolyát... a programozók többsége *nagy* és *összetett* programot szokott használni, amelyeket integrált fejlesztői környezetnek nevezünk (*IDE*). Azért nevezik integrált környezetnek, mert (szinte) minden benne van, ami elfordulhat egy program megírása során. A legfontosabb komponens egy *szöveg szerkesztő* (illetve *editor*), amely általában képes a Java nyelv kulcsszavait és fbb szerkezeti elemeit kiemelni (*syntax highlight*). Fontos komponens a *projekt* alapú fájlszervezés, illetve a projekt *lefordítási* és *futtatási* lehetősége. Célszerű egy *verziókövet* rendszer használata, bár ennek szükségessége csak hosszadalmas csapatmunka esetén fontos. Ha használunk *grafikus felhasználói felületet* (GUI), akkor örömmel vesszük, ha az IDE tartalmaz egy *GUI szerkesztőt* is. A készülő program fejlesztői dokumentációját is (fél)automatizálhatjuk, ha az IDE megkeresi a dokumentálatlan osztályokat és metódusokat, majd "kikényszeríti" a dokumentálást.

Az *ingyenes* környezetet használó Java fejlesztők között két nagyobb "hív" csoportot találunk: az egyik csoport a [NetBeans](#) programot, a másik csoport pedig az [Eclipse](#) környezetet részesíti elnyben. A maradék "kisebbség" osztozik az összes többi program között. Jelen cikksorozatban a NetBeans környezetet fogom bemutatni a befolyásolás szándéka nélkül (!), mivel én is ezt használom, és ezen oktattam a programozást is. A NetBeans letöltését, telepítését és elindítását nem részletezem, egyik tevékenység sem triviális olyan összetettnek és nehéznek, hogy egy programozást "életcélul" választó olvasónak ez gondot okozhasson... :)

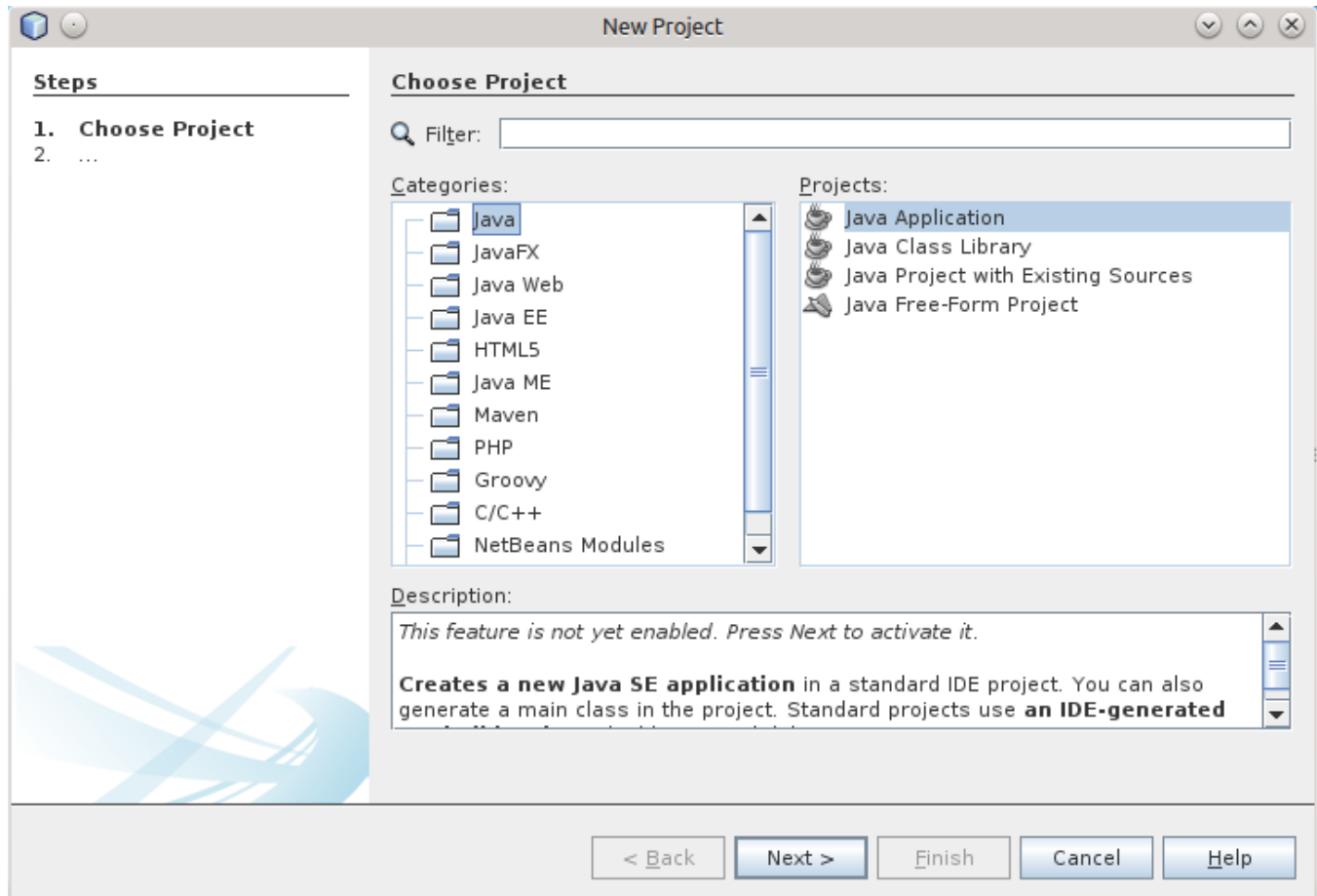


Maga a program kinézete távolról pont olyan, mint bármelyik másik fejlesztőeszköz: a felső részen egy átlagos menürendszert és egy ikonsort, az alsó rész bal oldalán egy fájlstruktúrát megjelenítő komponenst, jobb oldalt pedig a fülekkel kiválasztható szerkesztő ablakterületet találunk. Ezt természetesen testre szabhatjuk, de ezek a komponensek általában a nevezett helyen szoktak elfordulni.

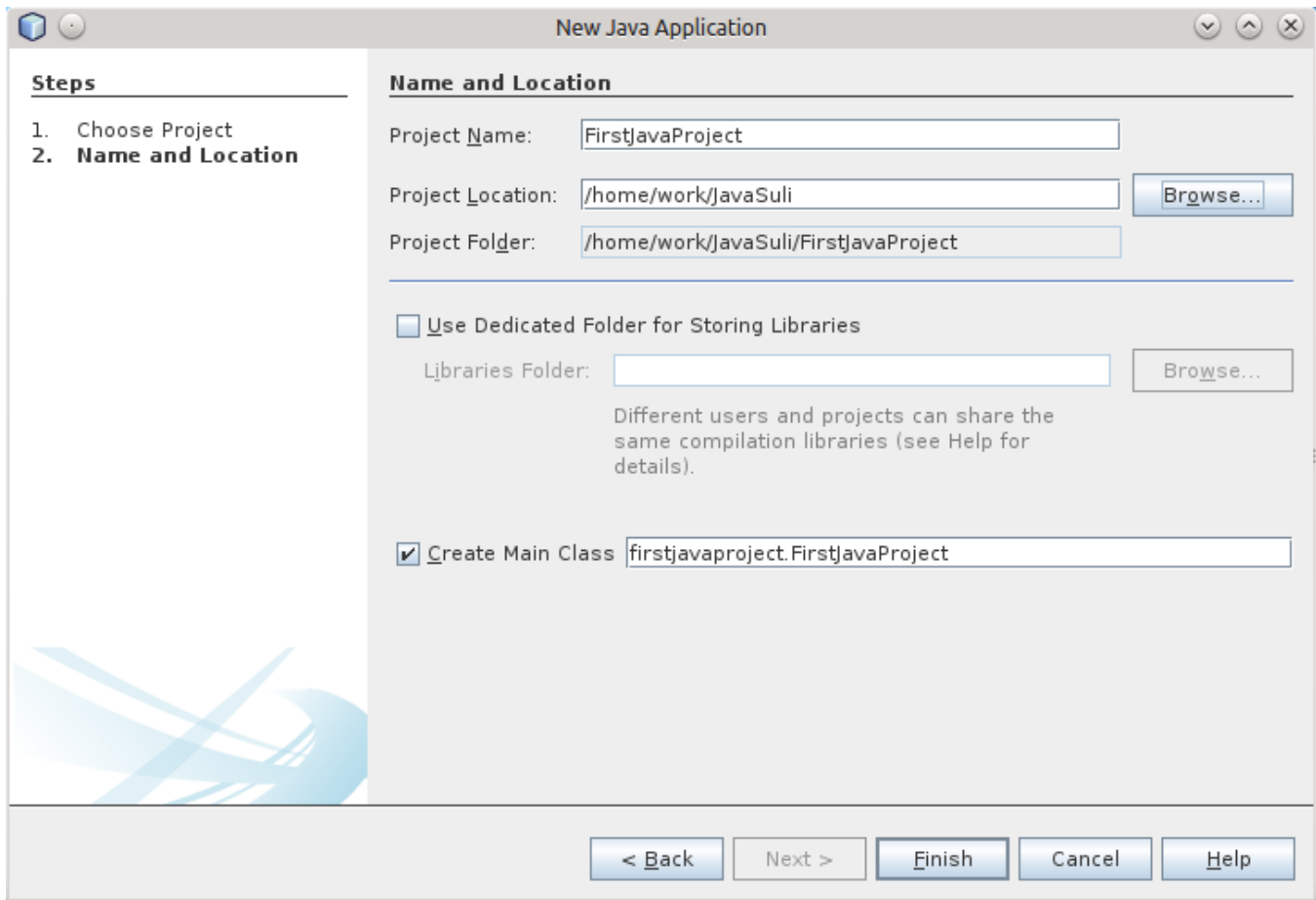


Els dolgunk, hogy létrehozzunk egy új *projektet*. Bár a fejlesztő környezetekkel képesek vagyunk egy-egy állományt külön is szerkeszteni, fordítani és futtatni: ezt lehetőleg *kerüljük el*. A projekt gondolkodás fontos része a fejlesztői munkának, ugyanis ez magát a teljes programfejlesztési utat tartalmazhatja a tervezéstől a kész programcsomag elkészítéséig. Tekintsük a projektet úgy, mint egy gyjtt, ahol mindent megtalálunk, ami a programunkkal kapcsolatos.

NetBeans esetén új projektet a *File -> New project...* menüpont segítségével tudunk létrehozni, amely elcsalogat egy dialógus ablakot.

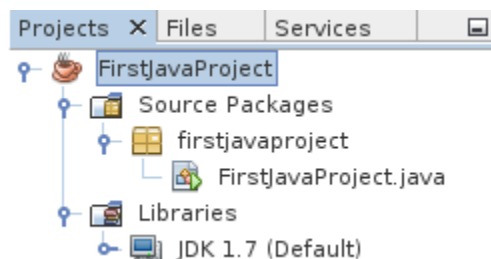


Új projekt esetén meg kell adnunk a projekt típusát, hiszen a Java eléggé univerzális nyelv lett: a mobiltelefontól kezdve egészen a nagy kiszolgálókig képesek vagyunk programot írni. A Sun által kitalált alaptípusok okán alapvetően három f irányzat közül választhatunk. Tudunk a *Java ME (Mobile Edition)* használatával mobil eszközökre programot írni (ám ez időközben kissé elavult nyelvjárás lett az Android miatt). Leggyakrabban a *Java SE (Standard Edition)* jellegű programokat szoktunk készíteni, ezek közé tartoznak a felhasználói programok, a Java Applet-ek és a legtöbb szerver oldali Java program. A *Java EE (Enterprise Edition)* olyan nagyvállalati technológiákat is hordoz, amelyekkel kényelmesen tudunk stabil, adatvédelem központú és jól skálázható programokat alkotni. A három f kategórián belül tudunk több altípust kiválasztani, ehhez szükséges, hogy a fejlesztő környezetben az adott típusok telepítve legyenek, vagyis ezek fejlesztését támogassa az IDE. Nekünk most a Java típuson belül a *Java Application* altípus kell.



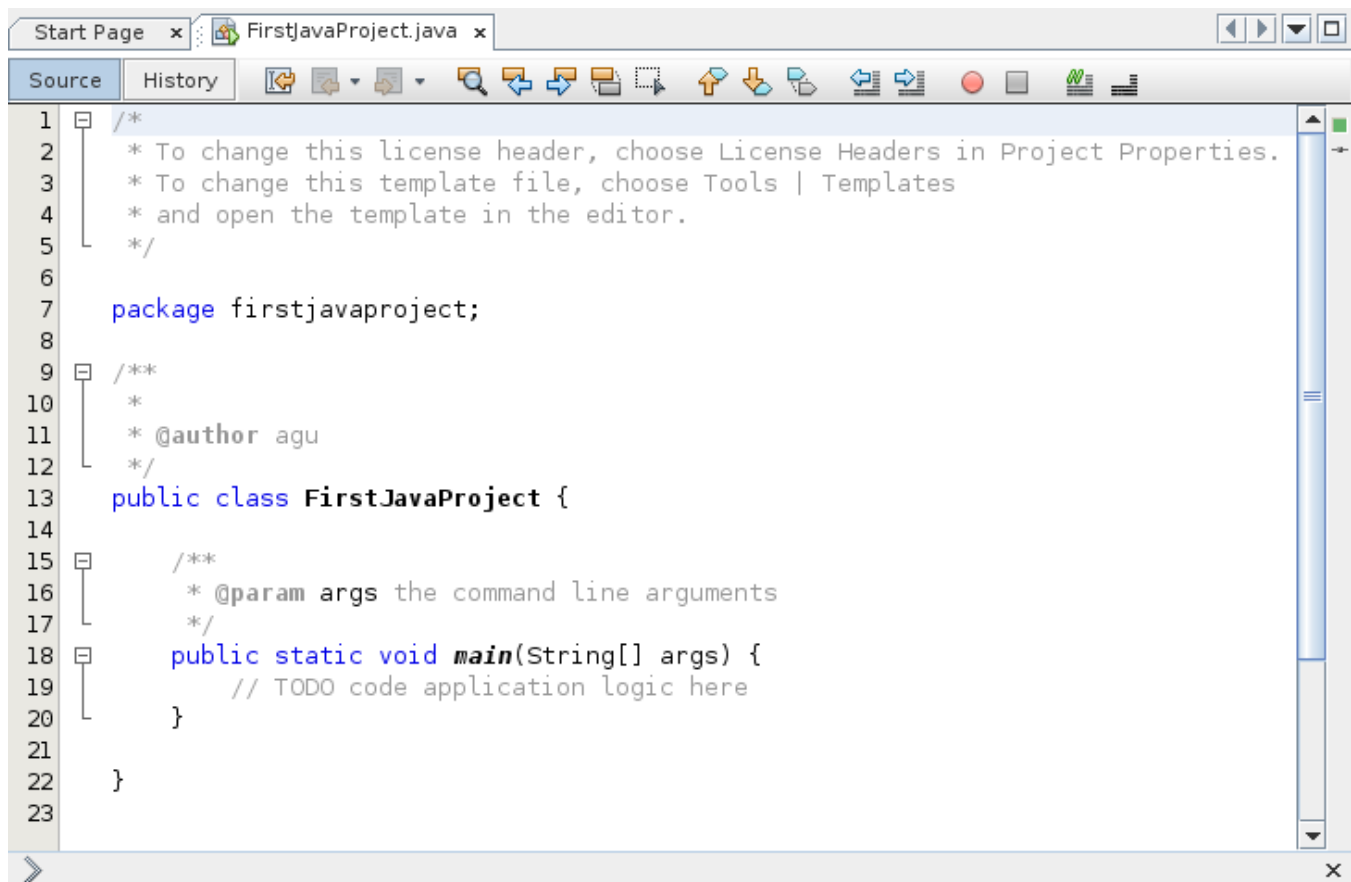
A projektnek nevet kell adni, amely – bár tartalmazhat ékezetes betűket is – lehetőleg ékezetmentes legyen, írjuk egybe a szavakat, és minden szó első betűje nagybetű legyen. Ezt a stílust a Java nyelv tervezői alakították ki, vétek lenne ettől eltérni. Célszerű egy külön könyvtárat szánni a projekteknek, ez akár lehet a saját könyvtárunkban belül a Java is. Egyéb fejlesztőkönyvtárak kérdezhetnek még pár információt, mint például egy leírást, vagy akár verziókövető rendszert is elérési útvárat. Nekünk elég ennyi információ, így a *Finish* gombra kattintva elkészül egy mintából a *FirstJavaProject* projektünk.

A környezet általában megnyitja a projekt f állományát (*FirstJavaProject.java*), amely egy üres és különösebb funkciók nélküli osztály (az osztályokra később visszatérünk). Elsőként azonban nézzük meg a bal felső részén megtalálható fájlstruktúrát, s ennek hasznát.



Alapvetően a kettő lenyitható ág érdekes, az egyik a *Source Packages*, a másik pedig a *Libraries*, természetesen egy összetettebb projekt esetén lesz még néhány egyéb ág is ennek a fácskának, de jelen esetben elégedjünk meg ezzel a kettővel, amelyek közül az első ágat fogjuk használni, ahol a forrásállományaink kapnak helyet. A Java csomagokra osztja a forrásállományokat, amelyeket tekinthetünk könyvtáraknak vagy mappáknak is. A NetBeans mindig a projekt kisbetsített nevét adja annak a csomagnak, amelyben a program kap helyet. A második ághoz a programunk által használt osztálykönyvtárak és egyéb könyvtárakra történik hivatkozás, de erről jóval később tesztek majd említést.

A fejlesztő eszköz legfontosabb része az a terület, ahol a programot szerkesztjük. Általában minden környezet esetén valamilyen füles rendszer biztosítja, hogy az éppen betöltött projekt több állományát is szerkeszthessük kevés átkapcsolási idővesztéssel. A szerkesztő általában kiemeli a kulcsszavakat, más színnel mutatja a szöveges és a karakteres literálokat, illetve a sok egyéb kiemelési tudnak.



```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6
7  package firstjavaproject;
8
9  /**
10   *
11   * @author agu
12   */
13  public class FirstJavaProject {
14
15      /**
16       * @param args the command line arguments
17       */
18      public static void main(String[] args) {
19          // TODO code application logic here
20      }
21
22  }
23
```

Els programunk a klasszikus "Hello World!" szöveget fogja kiírni a konzol kimenetére, ehhez egy kissé át kell írunk a fájl tartalmát:

Java

```
package firstjavaproject;

/**
 * Hello World program
 *
 * @author auth.gabor
 */
public class FirstJavaProject
{

    public static void main(String[] args)
    {
        System.out.println("Hello World!");
    }
}
```

A projekt lefordítása – noha csak egyetlen forrásállományt tartalmaz – a *Run -> Run Main Project* menüponttal vagy az *F6* gomb megnyomásával történik. Ekkor a környezet megnézi a projekt forrásállományait, és lefordítja azokat, amelyek az elz fordítás óta változtak. Ha a fordítás sikeres volt, akkor a kész projektet le is futtatja. Minderről az eddig üres alsó panelen kapunk információt:

Result

```
run:
Hello World!
BUILD SUCCESSFUL (total time: 0 seconds)
```

Ezzel elkészült az els Java projektünk, dljünk hátra és szemléljük elégedetten a világot, mint *Java programozó*... 😊

1.2. Kódolási stílusok

Alapveten a Java önmagában is egy érdekes stílust használ, amelynek alapjai a változók, osztályok, állományok, konstansok és metódusok elnevezésében rejlik. Az összes különálló szó, amely egy osztály, egy állomány nevében szerepel nagybetűvel kezdődik:

```
TextBox
MyFileFilter
MainWindow
AbstractToolkit
```

Ezen túlmenően a változók és metódusok elnevezése teljesen hasonló, viszont az első szó kisbetűvel kezdődik:

```
setText()
stringBuffer
loadedFile
getAccountId()
```

Az előre definiált konstansok neve esetén pedig minden betű nagybetű, a szavak elválasztása aláhúzással történik:

```
MAX_CHAR
STORE_ID
LOADED_FILE
MIN_LENGTH
```

Érdemes minden körülmény esetén ezekhez a konvencióhoz ragaszkodni, mivel ez a Java része, bár más esetben is le fog fordulni a programunk, csak esetleg a későbbiekben fogjuk megbánni, ha mások is bedolgoznak majd a programunkba; vagy az általunk készített osztályt mások is használni szeretnék.

Fontos újdonság, hogy a Java forrást akár Unicode kódlappal is írhatjuk, így a használt azonosítók már tetszőleges betűket tartalmazhatnak, nem kell ékezetelleníteni a magyar változóneveket sem. Bár ez a külföldi kollégákat esetleg érzékenyen érinti, ezért kerüljük ezt a megoldást. Gondoljunk bele, hogy mennyi nehézséget okozna, ha egy kínai karakterekkel telezsúfolt programot kellene továbbfejleszteni... :)

A Java saját stílusán kívül még szokás két forrásprogram stílust megkülönböztetni, amelyeknek természetesen vannak variációi. Ezek már nem annyira a Java részei, ezekkel találkozhatunk a rokon nyelvekben is.

Elérési utak

Az egyik stílus igazából a külső könyvtárak (illetve Java nyelven: csomagok) felhasználásával kapcsolatos. Megtehetjük, hogy a program elején a teljes csomagra való hivatkozást megemlíti az `import` kulcsszóval, ekkor a fordító értelemszerűen hozzápróbálja a megtalált osztályokat és egyéb hivatkozásokat az `import` listához. Ennek hátránya, hogy elvagyázatlan tervezéskor elfordulhat két csomagban azonos osztálynév vagy konstansnév (jellemzően egy saját és egy beépített osztály neve ütközik), és ekkor az első "találat" eredménye lesz a programban, ezért általában csak hosszadalmas keresgélés árán találhatjuk meg a hiba okát:

SajatOsztaly.java

```
import java.util.*;
import javax.swing.*;

public class SajatOsztaly extends JFrame
{
    private List buttons = null;
    private JButton okButton = null;

    public SajatOsztaly()
    {
        buttons = new Vector( );
        buttons.add(new JButton("Egy"));
        buttons.add(new JButton("Kett"));
        okButton = new JButton("OK" );
    }
}
```

A fenti hibát kiküszöbölhetjük, ha minden egyes csomagra a hivatkozást a program forrásszövegében ejtjük meg, így közvetlenül megadhatjuk a megfelelő csomag elérési útját. Ennek hátránya, hogy sokkal többet kell gépelni, viszont sok aljas rejtőkód hibát kikerülhetünk ezzel:

SajatOsztaly.java

```
public class SajatOsztaly extends javax.swing.JFrame
{
    private java.util.List buttons = null;
    private javax.swing.JButton okButton = null;

    public SajatOsztaly()
    {
        buttons = new java.util.Vector( );
        buttons.add(new javax.swing.JButton("Egy"));
        buttons.add(new javax.swing.JButton("Kett"));
        okButton = new javax.swing.JButton( "OK" );
    }
}
```

A kett út között használhatjuk az "arany középutat" is: név szerint importáljuk a csomagokat, mivel így nagyon ritka lesz a névátfedés:

SajatOsztaly.java

```
import java.util.List;
import java.util.Vector;
import javax.swing.JButton;
import javax.swing.JFrame;

public class SajatOsztaly extends JFrame
{
    private List buttons = null;
    private JButton okButton = null;

    public SajatOsztaly()
    {
        buttons = new Vector( );
        buttons.add(new JButton("Egy"));
        buttons.add(new JButton("Kett"));
        okButton = new JButton( "OK" );
    }
}
```

Igazítás

A második stílusváltozat különféle alkalmazása már az olvashatóságon is sokat ront vagy javít. A megfelelő helyekre tett új sorokból és szóközökből álló formázásban vehetjük észre ezt a stílust, mint például amely az én szememnek jól olvasható:

SajatOsztaly.java

```
import java.util.*;
import java.awt.event.*;
import javax.swing.*;

public class SajatOsztaly extends JFrame implements ActionListener
{
    private List buttons = null;
    private JButton okButton = null;

    public SajatOsztaly()
    {
        buttons = new Vector( );
        buttons.add(new JButton("Egy"));
        buttons.add(new JButton("Kett"));
        okButton = new JButton( "OK" );
    }
}
```

Természetesen nem biztos, hogy mindenki számára ez jelenti az olvasható stílust, mivel ezen kívül bármilyen formázás lehetséges egészen a következ – szerintem olvashatatlan és átláthatatlan stílusig, mint a következ:

SajatOsztaly.java

```
import java.util.*;
import javax.swing.*;
import java.awt.event.*;
public class SajatOsztaly extends JFrame implements ActionListener {
    private List buttons = null;
    private JButton okButton = null;
    public SajatOsztaly() {
        buttons = new Vector( );buttons.add(new JButton("Egy"));
        buttons.add(new JButton("Kett"));okButton = new JButton( "OK" );
    }
}
```

Ez vitathatatlanul tömörebb formát ad, a fordító még örül is esetleg a kevés szóköznek és egyéb jeleknek, viszont (számomra) megnehezíti a program működésének követését, illetve struktúrájának átlátását: ez pedig a program bvíthettségét és karbantartását is megnehezíti. A fordítóprogram számára igazából teljesen mindegy, hogy egy vagy nyolc szóközt kell kihagyni a megfelel helyen, viszont hiba esetén egy kicsit nehezebb az okát megkeresni.