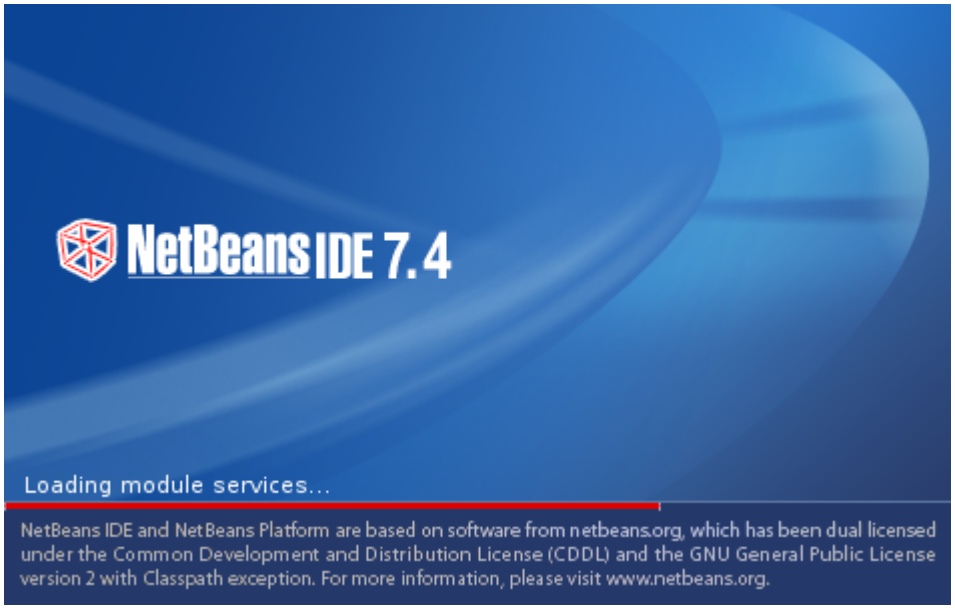
1. Bevezet

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.

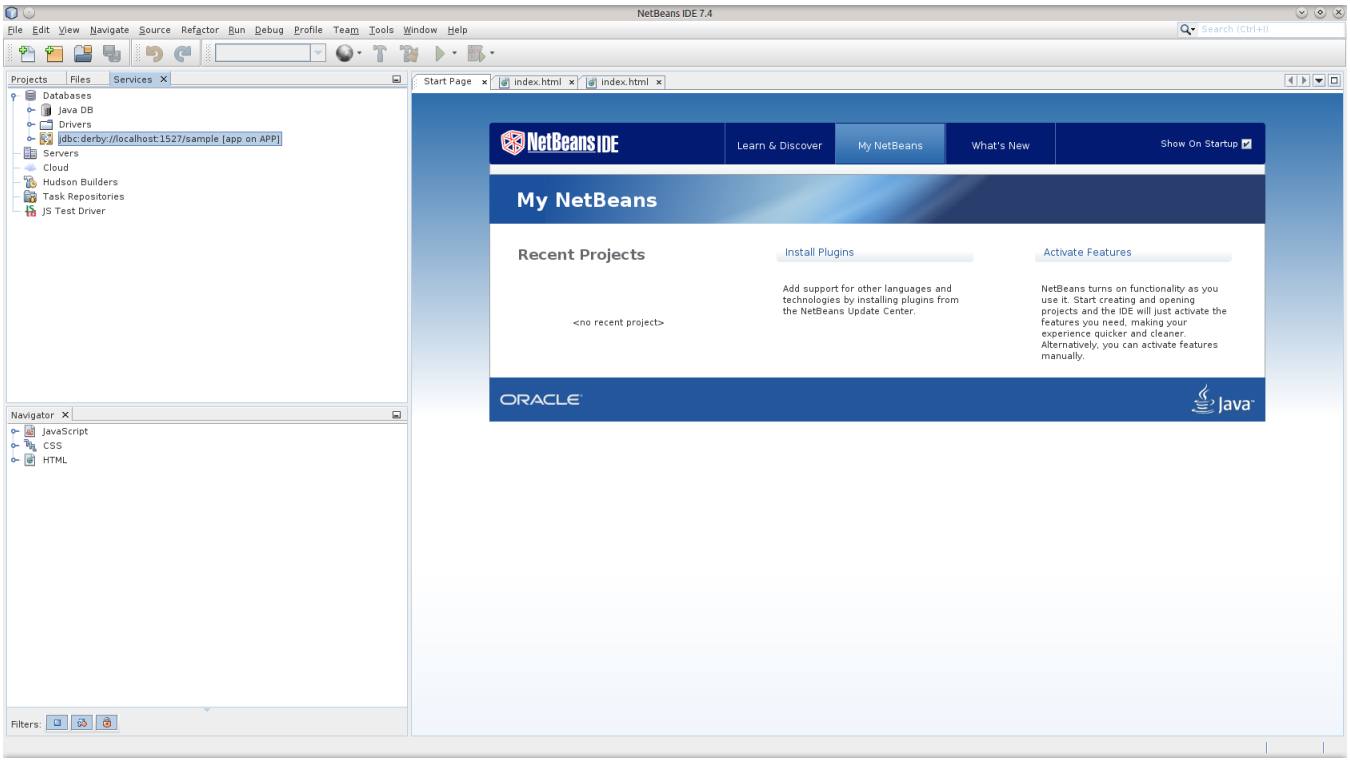
1.1. Fejleszti környezetek

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 fejleszt környezetetnek nevezünk (*IDE*). Azért nevezik integrált környezetnek, mert (szinte) minden benne van, ami elfordulhat egy program megírása során. A legfontosabb komponens egy *szöve gszerkeszt* (illetve *editor*), amely általában képes a Java nyelv kulcsszavait és fbb szerkezeti elemeit kiemelni (*sytax highlight*). Fontos komponens a *projekt* alapú fájlszervezés, illetve a projekt *lefordítási* és *futtatási* lehetsége. Célszer 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 szerkesztt* is. A készül program fejleszti dokumentációját is (fél)automatizálhatjuk, ha az IDE megkeresi a dokumentálatlan osztályokat és metódusokat, majd "kikényszeríti" a dokumentálást.

Az *ingyenes* környezetet használó Java fejlesztk 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 tnik olyan összetettnek és nehéznek, hogy egy programozást "életcélul" választó olvasónak ez gondot okozhasson... :)

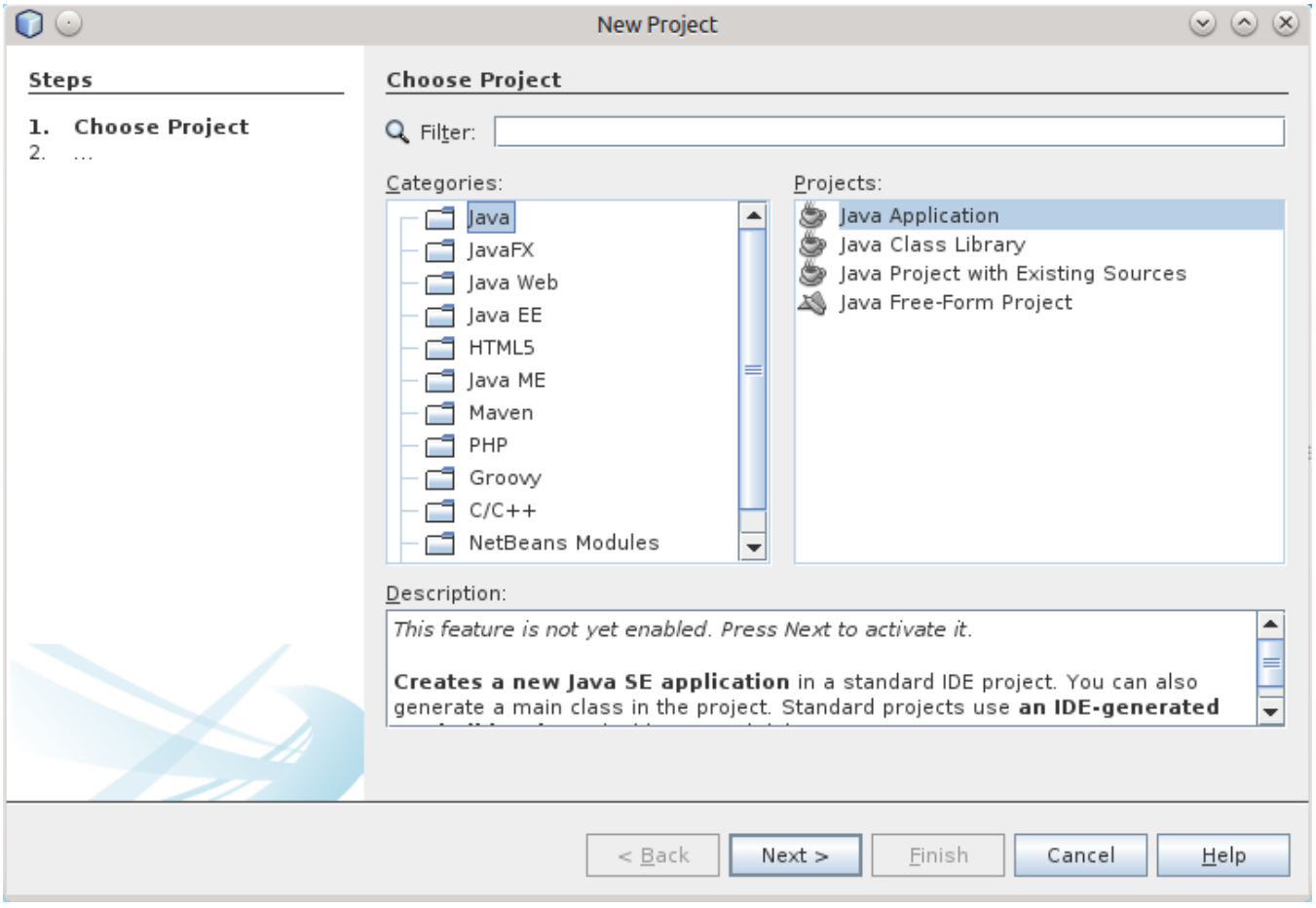


Maga a program kinézete távolról pont olyan, mint bármelyik másik fejlesztesköz: a fels részén egy átlagos menürendszert és egy ikonsort, az alsó rész bal oldalán egy fa struktúrát megjelenít komponenst, jobb oldalt pedig a fülekkel kiválasztható szerkeszt ablakterület találunk. Ezt természetesen testre szabhatjuk, de ezek a komponensek általában a nevezett helyen szoktak elfordulni.

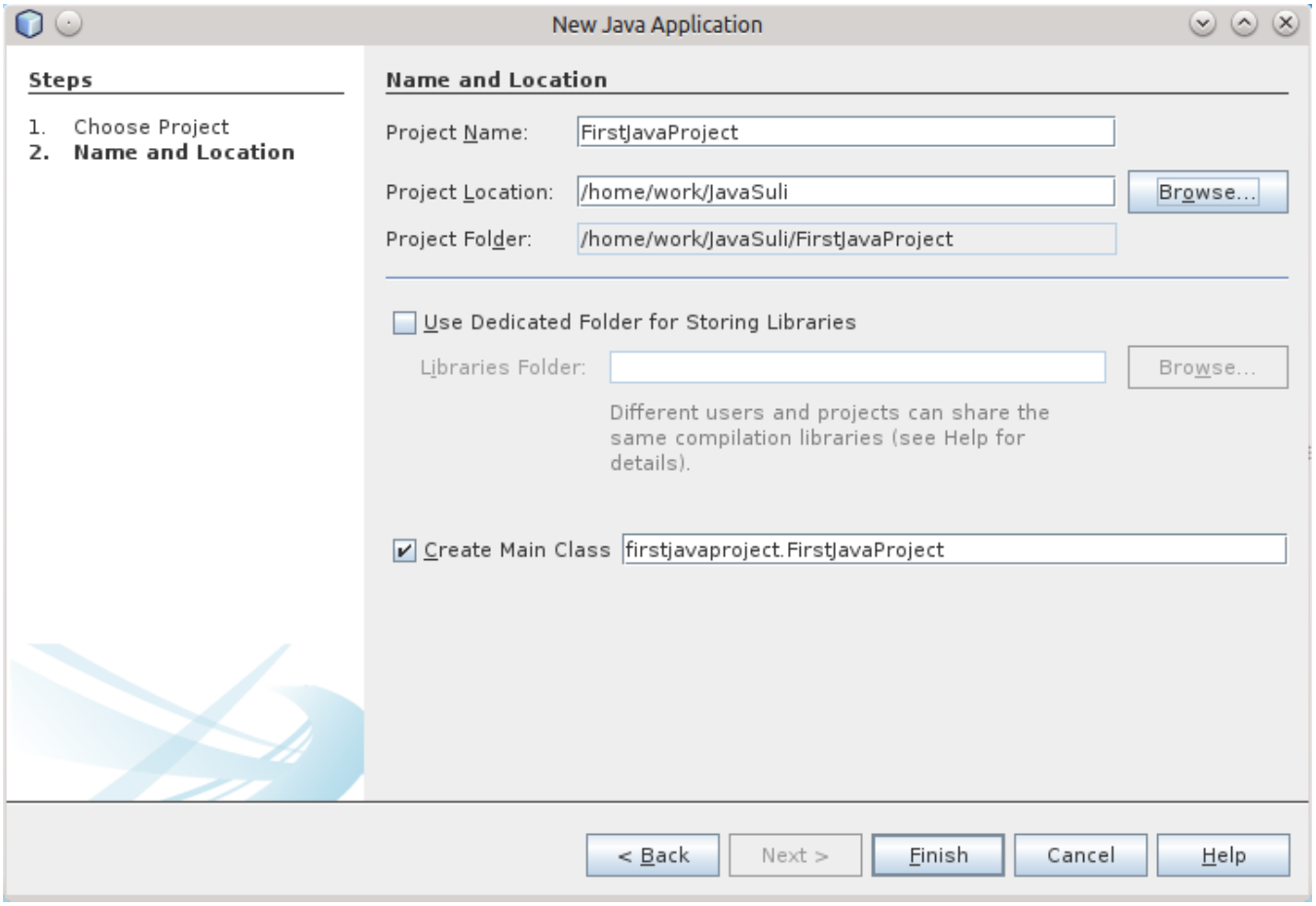


Els dolgunk, hogy létrehozzunk egy új *projektet.* Bár a fejleszt környezetekkel képesek vagyunk egy-egy állományt külön is szerkeszteni, fordítani és futtatni: ezt lehetleg *kerüljük el.* A projekt gondolkodás fontos része a fejleszti munkának, ugyanis ez magát a teljes programfejlesztési utat tartalmazhatja a tervezéstl 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.

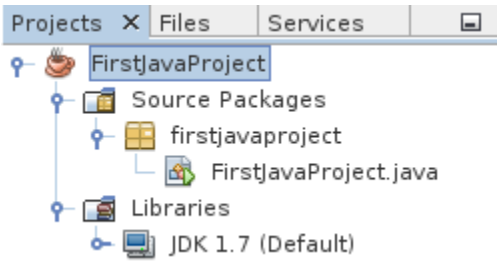


Új projekt esetén meg kell adnunk a projekt típusát, hiszen a Java eléggé univerzális nyelv lett: a mobiltelefontól kezdve egészen a nagy kiszolgálókig képesek vagyunk programot írni. A Sun által kitalált alaptípusok okán alapveten három f irányzat közül választhatunk. Tudunk a *Java ME* (*Mobile Edition*) használatával mobil eszközökre programot írni (ám ez idközben kissé elavult nyelvjárás lett az Android miatt). Leggyakrabban a *Java SE* (*Standard Edition*) jelleg 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 fejleszt környezetben az adott típusok telepítve legyenek, vagyis ezek fejlesztését támogassa az IDE. Nekünk most a *Java* típuson belül a *Java Application* altípus kell.



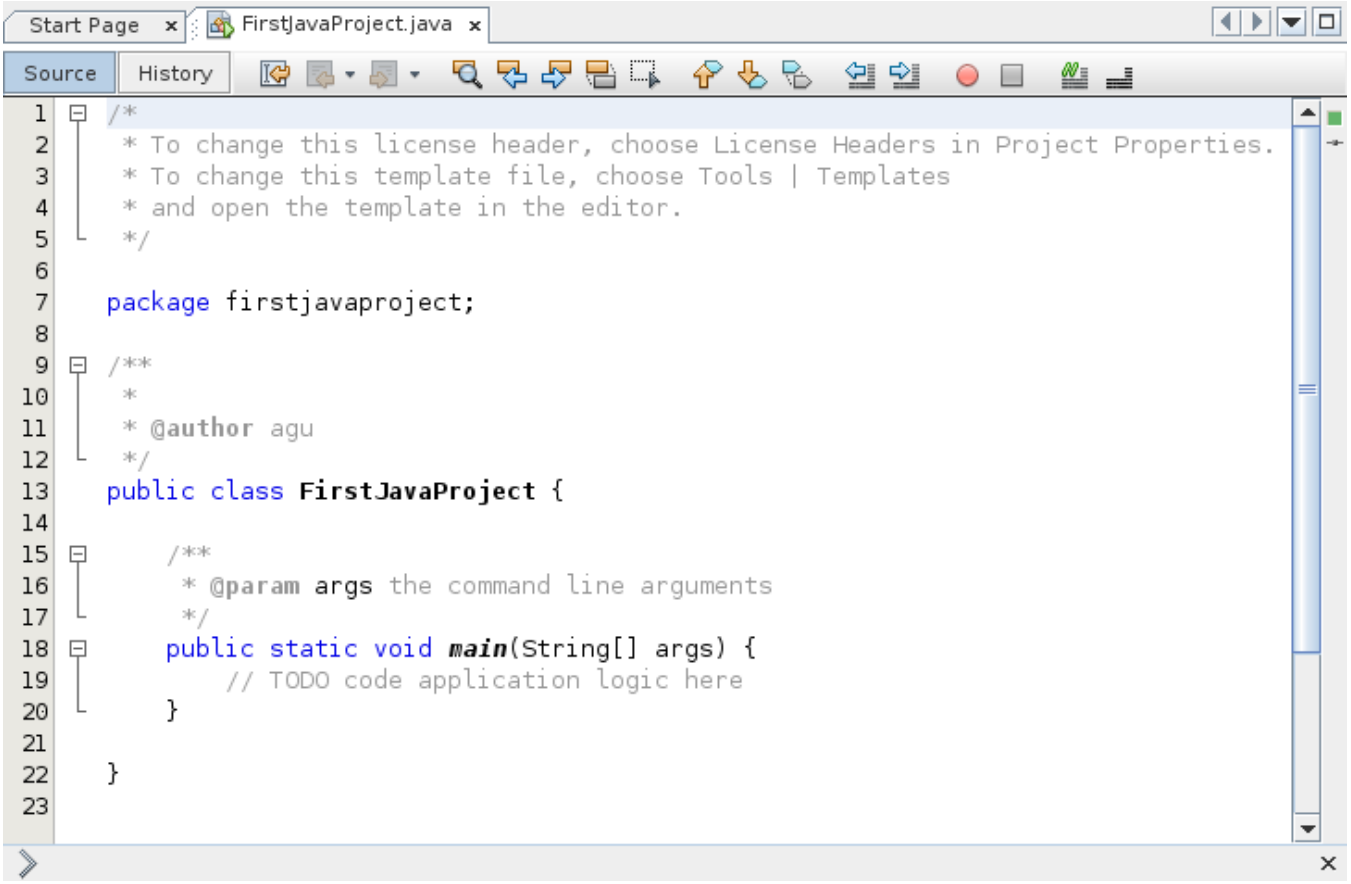
A projektnek nevet kell adni, amely – bár tartalmazhat ékezetes betket is – lehetleg ékezetmentes legyen, írjuk egybe a szavakat, és minden szó els betje nagybet legyen. Ezt a stílust a Java nyelv tervezi alakították ki, vétek lenne ettl eltérni. Célszer egy külön könyvtárat szánni a projekteknek, ez akár lehet a saját könyvtáron belül a *Java* is. Egyéb fejlesztkörnyezetek kérdezhetnek még pár információt, mint például egy leírást, vagy akár verziókövet rendszer elérési útját. 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 késbb visszatérünk). Elször azonban nézzük meg a bal fels részén megtalálható fa struktúrát, s ennek hasznát.



Alapveten a kett lenyitható ág érdekes, az egyik a *Source Packages*, a másik pedig a *Libraries*, természetesen egy összetettebb projekt esetén lesz még néhány egyéb ága is ennek a fácskának, de jelen esetben elégedjünk meg ezzel a kettvel, amelyek közül az els ágat fogjuk használni, ahol a forrás állományaink 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 fprogram kap helyet. A második ágban a programunk által használt osztálykönyvtárak és egyéb könyvtárakra történik hivatkozás, de errl jóval késbb teszek majd említést.

A fejleszt 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 idveszteséggel. A szerkesztk általában kiemelik a kulcsszavakat, más színnel mutatják a szöveges és a karakteres literálokat, illetve a sok egyéb kiemelést tudnak.



Els programunk a klasszikus *"Hello World!"* szöveget fogja kiírni a konzol kimenetére, ehhez egy kissé át kell írnunk 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. Minderrl 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 nagybetvel kezddik:

```
TextBox
MyFileFilter
MainWindow
AbstractToolKit
```

Ezen túlmenen a változók és metódusok elnevezése teljesen hasonló, viszont az els szó kisbetvel kezddik:

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

Az elre definiált konstansok neve esetén pedig minden bet nagybet, 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 késbbiekben 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 tetszleges betket tartalmazhatnak, nem kell ékezetteleníteni 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 küls könyvtárak (illetve Java nyelven: csomagok) felhasználásával kapcsolatos. Megtehetjük, hogy a program elején a teljes csomagra való hivatkozást megemlítjük az *import* kulcsszóval, ekkor a fordító értelemszeren hozzápróbálgatja a megtalált osztályokat és egyéb hivatkozásokat az import listához. Ennek hátránya, hogy elvigyázatlan tervezéskor elfordulhat két csomagban azonos osztálynév vagy konstansnév (jellemzen egy saját és egy beépített osztály neve ütközik), és ekkor az els "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 megfelel csomag elérési útját. Ennek hátránya, hogy sokkal többet kell gépelni, viszont sok aljas rejtzköd 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 megfelel helyekre tett új sorokból és szóközökbl 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 mködésének követését, illetve struktúrájának átlátását: ez pedig a program bvíthetségét 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.