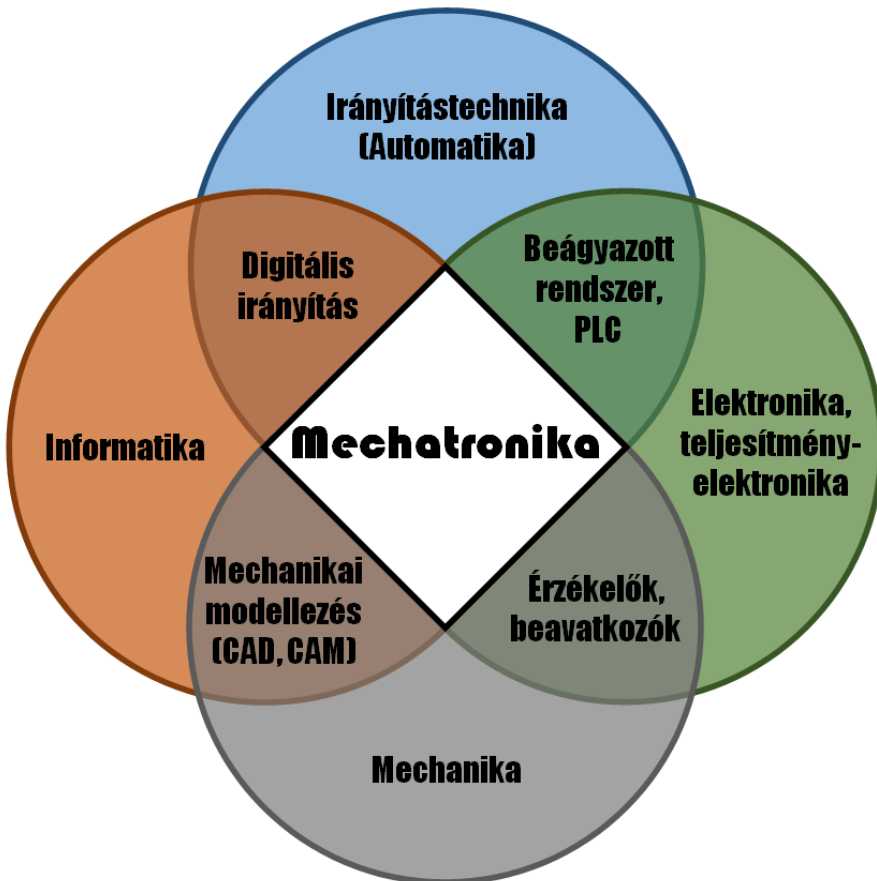


Kővári Attila – Katona József

**Arduino és Java szoftverfejlesztés,
soros kommunikáció,
digitális be- és kimenetek kezelése**

***Mechatronika a gyakorlatban
Arduino és Raspberry Pi 2 alapokon***



**Arduino és Java szoftverfejlesztés,
soros kommunikáció,
digitális be- és kimenetek kezelése**

*Mechatronika a gyakorlatban
Arduino és Raspberry Pi 2 alapokon*

@Kővári Attila – Katona József

Lektorálta: Dr. Odry Péter

Publio Kiadó
Felelős kiadó: Publio Kiadó Kft.
2015
Minden jog fenntartva!
További impresszum adatok...

Előszó

Manapság már az ipari gyártófolyamatok és általánosságban az egyes berendezések automatizálása elképzelhetetlen digitális irányítóberendezések alkalmazása nélkül. A gyártó, megmunkáló, mozgató, energiatermelő mechanikai mozgást végző gépek automatizálása terén az igazi fejlődést a digitális számítógépek, a villamos mérés-, vezérlés- és szabályozástechnika, villamos gépek és más villamos működtetésű energia átalakítók alkalmazása hozta meg. A **mechatronika (mechanika és elektronika)** szó is erre utal, vagyis egy multidiszciplináris területet jelöl, mely elsősorban a gépész, villamos és automatikai területhez kapcsolódik, azok alkalmazása a gépészeti berendezések automatizálására, akár intelligenssé tételére. Manapság mechatronikán ennél még többet értünk, mivel a gépek, berendezések egyre komplexebbé válnak, ezért mára a mechatronika már az előbb felsorolt szakterületeken kívül az információfeldolgozás, alkalmazott informatika, méréstechnika, villamos energetika, híradástechnika, távközlés szakterületek együttes alkalmazását jelenti. Mechatronikai megoldások esetén az egyes szakemberek nem kizárólag a saját szakterületükhöz kapcsolódó problémák, hanem egy több szakterületet integráló csapat tagjaként, egy közös cél optimális megoldásán kell dolgozzanak. Ezért egy mechatronikai mérnöknek azonosulnia kell a komplex rendszerben történő gondolkodás filozófiájával, és mindennel, amely e filozófia mögött van. Nem egyes részegységek, hanem a teljes rendszer optimális megoldására kell törekedjen ésszerű kompromisszumok által.

E könyv a mechatronika alapjait gyakorlati példákon keresztül megismertető sorozat egy kötete. A könyvsorozat elsősorban a mechatronika villamos és informatikai ismereteit felölelő, információ feldolgozó, érzékelő és beavatkozó egységeivel foglalkozik. A könyv gyakorlati feladatok felhasználásával segítséget kíván nyújtani a szakemberek számára a mechatronikai rendszerek felépítésének megismerésében. A kötetekben található példák olyan feladatokat, mechatronikai rendszereket ölelnek fel, amelyek olcsón, bárki számára beszerezhető eszközök felhasználásával otthoni körülmények között is összeállíthatók, tesztelhetők. A sorozat a mechatronikai rendszerek tárgyalása során előzetes ismeretekre támaszkodik elsősorban a középiskolai fizika elektromosság, mágnesesség, testek mozgása, és alapszintű programozási ismeretek területén.

Bízunk benne, hogy azok, akik végigolvassák a könyvet, és összeállítják, tesztelik a gyakorlati példákat, azok alkalmazás szintjén megismerik a mechatronikai rendszerekben rejlő lehetőségeket, elsajátítják a mechatronikai rendszerek tervezésének, kialakításának alapjait, és a tárgyalt ismeretek és gyakorlatias példák pedig jó alapot adnak az egyes mechatronikai feladatok megoldásához.

Jelen kötet az Arduino és Java szoftverfejlesztés alapjait, a soros kommunikáció, digitális be- és kimenetek kezelését egyszerű gyakorlati példákon keresztül mutatja. A fejlesztett Java alkalmazás Windows operációs rendszerű PC-n és Linux operációs rendszerű Raspberry Pi 2-ön is futtatható.

Köszönettel tartozunk a kézirat lektorának, Dr. Odry Péter szakmai tanácsaiért és módszertani javaslataiért.

A Szerzők

Komplex irányítórendszerek, SCADA

A korszerű irányítórendszerekre jellemző, hogy igen összetettek. Egy komplex irányítórendszerben több speciális vezérlőegység, számos érzékelő jele alapján a folyamat állapotát módosítani képes beavatkozók segítségével hajtja végre a kívánt feladatot. Az irányított folyamat központi felügyeletét, kezelését, a folyamat állapotának általában grafikus sémaképen történő megjelenítését, mérésadatgyűjtést és megjelenítését, eseményregisztrálást, riasztást végző úgynevezett SCADA (Supervisory Control and Data Acquisition – felügyeleti irányító és mérésadatgyűjtő) rendszerek végzik. A SCADA rendszerek folyamatosan kommunikálnak a folyamat vezérlőberendezéseivel (melyek általában ipari vezérlők vagy beágyazott rendszerek) és paramétereket állítanak be és kérdeznak le tőlük.

SCADA rendszereknek ma már szervesen kell illeszkedni egyéb termelést támogató rendszerhez és adott esetben a vállalati információs rendszerekhez is. Így a SCADA rendszerek a folyamatmegjelenítés és paraméter beállítás lehetőségén túl a mért folyamatparamétereket adatbázisban tárolják és interfészeket biztosítanak az egyéb vállalati információs rendszerek számára.

A vezérlési funkciók ipari környezetben általában programozható logikai vezérlők, PLC-k (Programmable Logic Controller) segítségével valósul meg, de számos esetben alkalmaznak úgynevezett beágyazott rendszereket, melyek központi egységét általában valamilyen mikrovezérlő adja. Az Arduino egy egyszerűen kezelhető, olcsó, beágyazott rendszerek építése szempontjából ideális platform, ezért is esett a választás ezen platform alkalmazásának bemutatására.

Az irányítóberendezések általános működésére az adott ciklusidővel, ciklikusan végrehajtott műveletsorozat jellemző. Ez a műveletsorozat a bemenetek beolvasásával kezdődik, majd a vezérlő a bemenetek és a belső állapot alapján, a vezérlő programja által meghatározott aritmetikai és logikai utasítások végrehajtása után állítja be a kimeneteit. A ciklus végén elvégzi a kommunikációs feladatokat, egyrészt a folyamatfelügyeleti rendszer és esetleg közvetlenül más vezérlők felé. Az Arduino platform programozása során is az irányítási funkciókat megvalósító utasítássorozatot egy ciklikusan futó függvényben kell definiálni.

Kisebrendszerek esetén a felügyeleti funkciók megvalósítása akár történhet egy személyi számítógépen, vagy akár egy mini számítógépen is, mint például a Raspberry Pi 2. A Raspberry Pi 2 számítógépen elsősorban Linux operációs rendszer futtatható, de a Microsoft Windows 10 is telepíthető lesz rá. Egyes SCADA funkciók Raspberry Pi 2-ön történő megvalósításához a Java platform

került kiválasztásra, elsősorban platformfüggetlensége miatt, ami lehetővé teszi, hogy a Java platformon íródott programok hasonlóan fussanak a különböző hardvereken, operációs rendszereken.

A jövőben a platformfüggetlenséget a Microsoft .NET Core 5 framework is támogatni fogja, mivel nem csak Windows, hanem Linux és Mac is támogatott operációs rendszer lesz, így komoly alternatívát jelent majd a Java platformmal szemben.

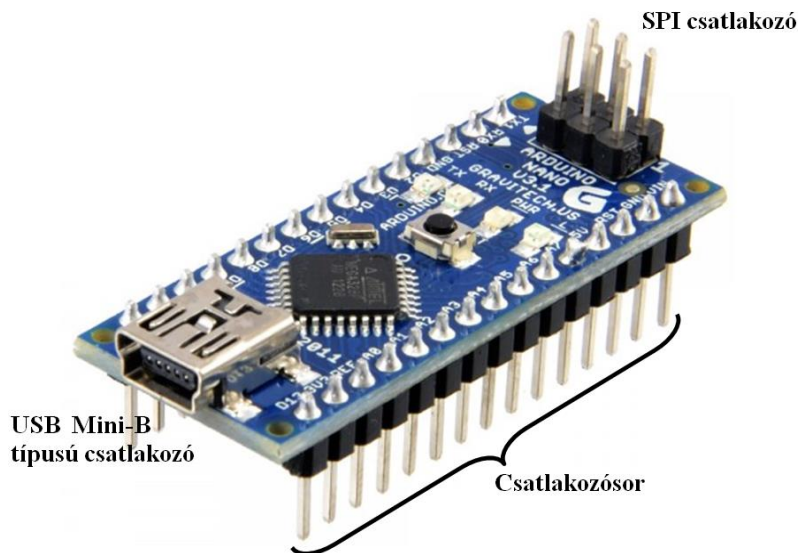
Szoftverfejlesztés Arduino platformra

Arduino programozása

Az Arduino fejlesztőkörnyezetben a programot C vagy C++ programozási nyelven lehet megírni. Az Arduino fejlesztőkörnyezet tartalmazza a nyílt forráskódú Wiring C/C++ függvénykönyvtárat, amely leegyszerűsíti, könnyebbé teszi a be- és kimeneti műveletek programozását. A Wiring projektről az alábbi weboldalon lehet részletesebben olvasni: <http://wiring.org.co/>.

A programban két fő függvényt kell definiálni, az egyik a *setup()*, amely bekapcsolás után és RESET megnyomásakor egyszer hívódik meg és kezdeti beállításokat, inicializálásokat végzi el, valamint a *loop()*, amely a folyamatosan, ciklikusan futó programrészt tartalmazza. Így a legegyszerűbb, funkció nélküli programot az alábbi ábrán látható kell definiálni. Az Arduinora készített programokat „sketch”-nek nevezik.

Az Arduino fejlesztőlap programozásához természetesen szükség van egy fejlesztőlapra. A következőkben az első kötetben bemutatásra került Arduino Nano kisméretű és olcsó fejlesztőmodulon kerül bemutatásra az Arduino fejlesztőplatformok programozása. A fejlesztőlap USB/USB Mini kábel segítségével köthető össze azzal a számítógéppel, amelyen a programfejlesztés fog történni.



1. ábra: *Arduino Nano V3.1 fejlesztőmodul*

Arduino IDE

Az Arduino programok fejlesztése az Arduino integrált fejlesztőkörnyezetben (IDE Integrated Development Environment) C/C++ nyelven történhet, mely az Arduino projekt weboldaláról www.arduino.cc letölthető a Download menüpont alatt. Windows operációs rendszert futtató számítógép esetén a Windows Installer vagy Windows ZIP fájlokat kell letölteni, az Installer verzió telepíti a fejlesztőkörnyezetet a ZIP fájl esetén ezt elegendő csak egy könyvtárba kicsomagolni. Az Arduino IDE elérhető Mac OS X vagy Linux operációs rendszerre is. A telepítés vagy kicsomagolás után az Arduino fejlesztőkörnyezet az arduino telepítési könyvtárában található arduino.exe –vel indítható. Indítás után a programkód szerkesztő ablak jelenik meg. A fejlesztőkörnyezet nyelve a File/Settings (Fájl/Beállítások) menüpontban a Szerkesztő nyelve beállításánál változtatható meg magyar nyelvre.

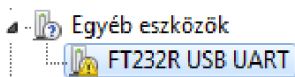


2. ábra: *Arduino fejlesztőkörnyezet kezdőablaka*

Csatlakoztassuk USB-n keresztül a fejlesztőlapot a számítógéphez. A fejlesztőlap a tápfeszültség ellátást USB kapcsolaton keresztül kapja, melyet egy zöld színű PWR jelöléssel ellátott LED (Light Emitting Diode – világító dióda) jelez. Az USB-re történő csatlakoztatás után feltehetően egy piros színű LED

folyamatosan villogni fog, mivel a legtöbb esetben egy egyszerű LED villogtató program már fel van töltve az eszközre.

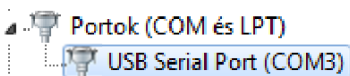
A fejlesztőlappal történő kommunikáció USB-n keresztül soros porton történik, melyhez egy virtuális soros port (VCP – Virtual Com port – virtuális soros port) illesztőprogram szükséges. Az Arduino Nano fejlesztőmodulon Future Technology Devices International Ltd. (FTDI) gyártmányú USB/soros illesztőegység található, ennek illesztőprogramja az Arduino fejlesztőkörnyezet könyvtárában a drivers alkönyvtárban megtalálható. Innét feltelepíthető a Vezérlőpult/Eszközkezelőben található Arduino, FT232, USB Serial névvel kezdődő ismeretlen eszközön dupla egérekattintással előugró ablakban az Illesztőprogram majd a Frissítést választva.



3. ábra: Eszközkezelőben megjelenő új eszköz

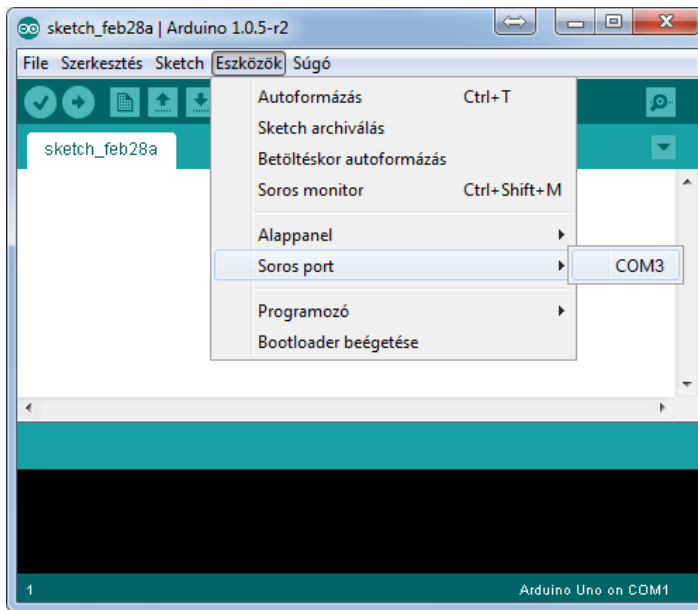
Amennyiben ez esetleg valamiért nem sikerül, úgy az FTDI illesztőegység telepíthető illesztőprogramja a gyártó honlapjáról is letölthető az alábbi linken: <http://www.ftdichip.com/Drivers/VCP.htm>. Egyes klón Arduino Nano fejlesztőmodulok a WCH Nanjing QinHeng Electronics által gyártott CH340G illesztőegységet tartalmaznak, annak illesztőprogramja az alábbi linken érhető el: <http://www.wch.cn/downloads.php?name=pro&proid=5>.

Az illesztőegység sikeres telepítése után az eszközhöz rendelt virtuális soros port száma az eszköz nevének végén olvasható le.

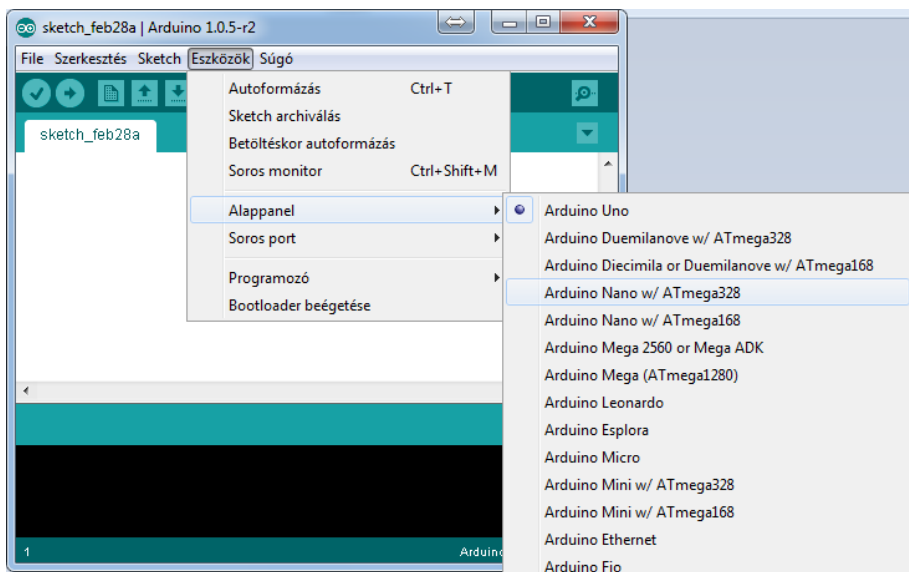


4. ábra: Arduino USB virtuális soros port eszköz

A képen látható eszköz megnevezésének a végén látható soros port száma a COM3. Ezzel a virtuális soros porton keresztül a számítógép és az Arduino modul között soros kommunikáció megvalósítható. Az Arduino fejlesztőkörnyezet ezen a soros porton keresztül, az Arduino fejlesztőmodul paramétereinek megadása után a megírt és lefordított programot fel tudja tölteni a fejlesztőlapon található mikrovezérlőre. A fejlesztőkörnyezetben ki kell választani azt a soros portot, amelyre az Arduino Nano fejlesztőlap van csatlakoztatva, jelen esetben a COM3, valamint az Alappanelelek között meg kell adni az Arduino Nano w/ ATmega328-at.

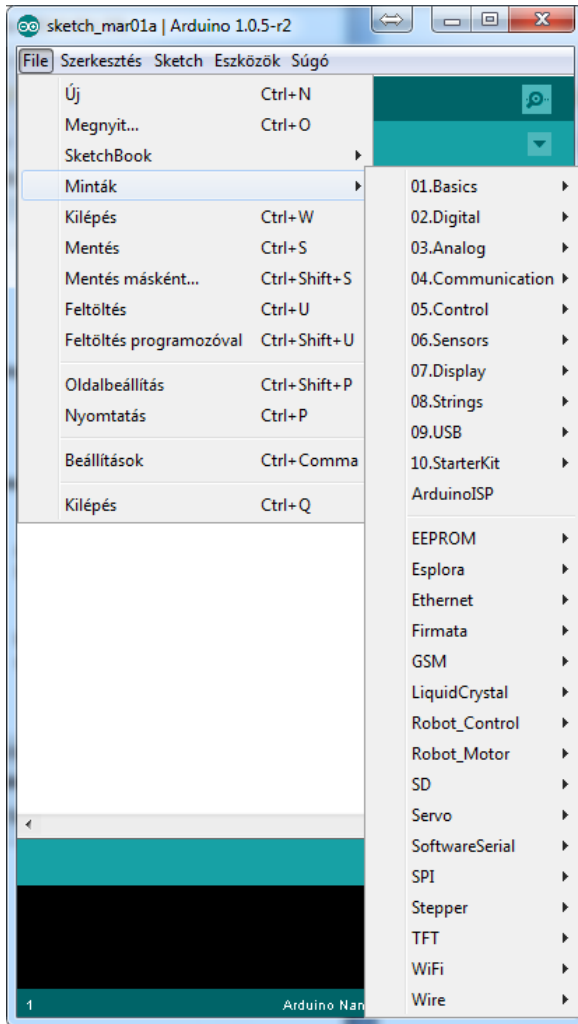


5. ábra: *Arduino fejlesztőkörnyezetben soros port beállítása*



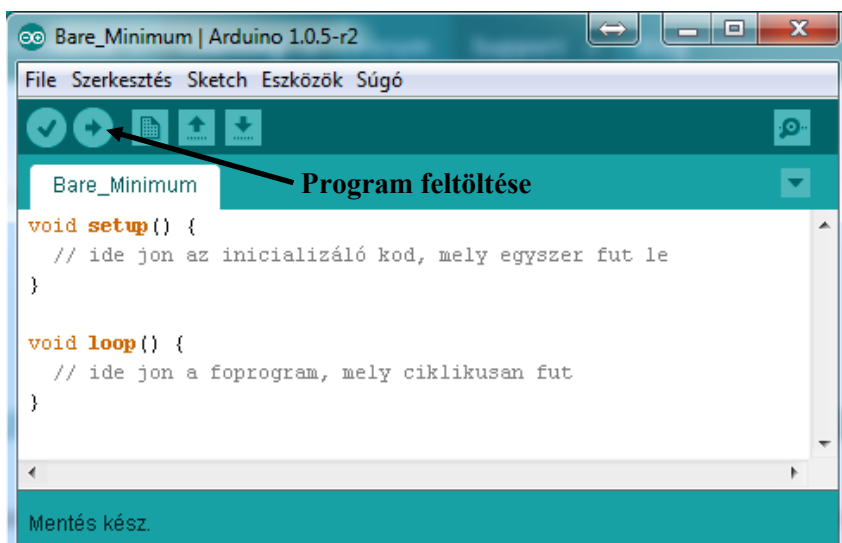
6. ábra: *Arduino fejlesztőkörnyezetben fejlesztőlap beállítása*

A beállítások után az eszköz már programozható is. A fejlesztőkörnyezet számos mintaprogramot tartalmaz, mely a File/Minták menüpontban találhatóak tématerületek szerint rendszerezve.



7. ábra: Mintaprogramok gyűjteménye az Arduino fejlesztőkörnyezetben

A fejlesztőkörnyezetben a program a „pipa” ikonra kattintva ellenőrizhető, fordítható, a mellette található „jobbra nyíl” ikonra kattintva pedig feltölthető az Arduino fejlesztőlapon található mikrovezérlőre, melyen a program a feltöltés után azonnal el is indul. Sikeres feltöltés esetén néhány másodperc múlva, az alsó részen található állapotsorban, a „Feltöltés kész.” jelenik meg. Feltöltés közben két LED az RX és TX soros kommunikációt jelző LED-ek villognak.



8. ábra: Program feltöltése az Arduino fejlesztőlapra

Szoftverfejlesztés Java platformra

Java platform

A Java platform egy olyan számítógépes környezet, amely képes a Java programozási nyelven írt programok futtatására. A Java kifejezés általában a Java programozási nyelvre vonatkozik, amit a Java platformban való használatra fejlesztettek ki. A Java platform a Java programozási nyelv és a futtató környezetet egysége.

A háttérben egy virtuális gépnek nevezett futtató program áll, mely többféle operációs rendszeren lehetővé teszi a Java programok futtatását. Ahhoz, hogy egy Java programot a Java platformon futtatni tudjunk, tehát szükségünk van egy Java virtuális gépre, és osztálykönyvtárak gyűjteményére, amiket a program felhasznál. A virtuális gépet, és az osztálykönyvtár megvalósításait a Java futtatói környezet a JRE (Java Runtime Environment) tartalmazza, ez szükséges a Java programok futtatásához.

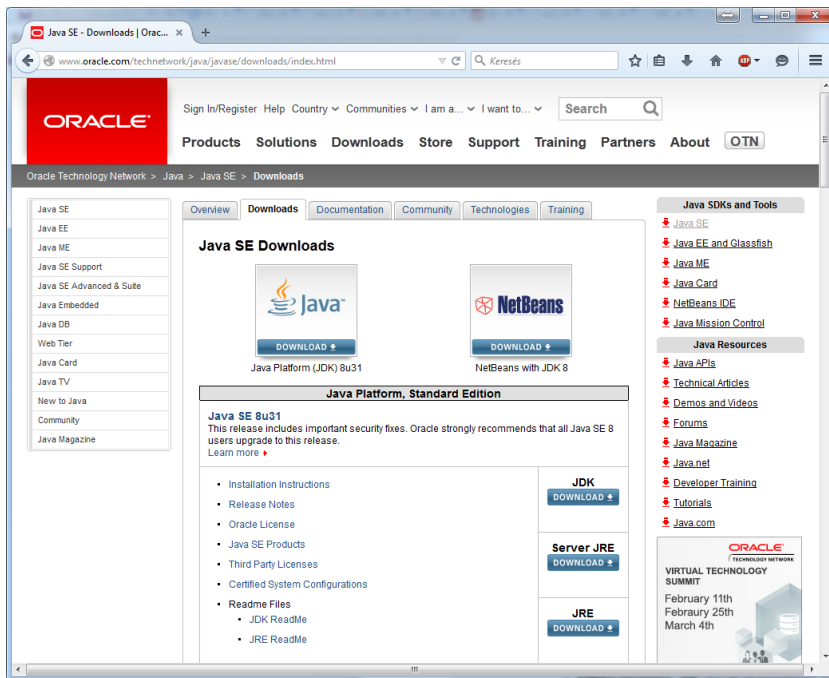
A Java virtuális gép a JVM (Java Virtual Machine) a Java bájtkódú programokat képes futtatni úgy, hogy azt futási időben fordítja át az adott operációs rendszer, gép utasításaira. Ez a köztes bájtkód tehát mindig ugyanaz, attól függetlenül, hogy milyen hardveren vagy operációs rendszeren fut. Így lényegében ugyanazt a Java programot bármilyen operációs rendszeren, környezetben lehet futtatni, amihez elérhető a Java program bájtkódú utasításait értelmező java virtuális gép. A JVM just-in-time (JIT) megvalósítása a Java program bájtkódú utasításait nem futási időben, hanem betöltéskor fordítja át, így a betöltés hosszabbá válik, de a futás gyorsabb lesz.

A Java alapú programok írásához is rendelkezésre áll olyan osztálykönyvtár gyűjtemény a JCL (Java Class Libraries) és Java API, mely igen sok használatra kész szoftverkomponenst tartalmaz úgynevezett csomagokba szervezve.

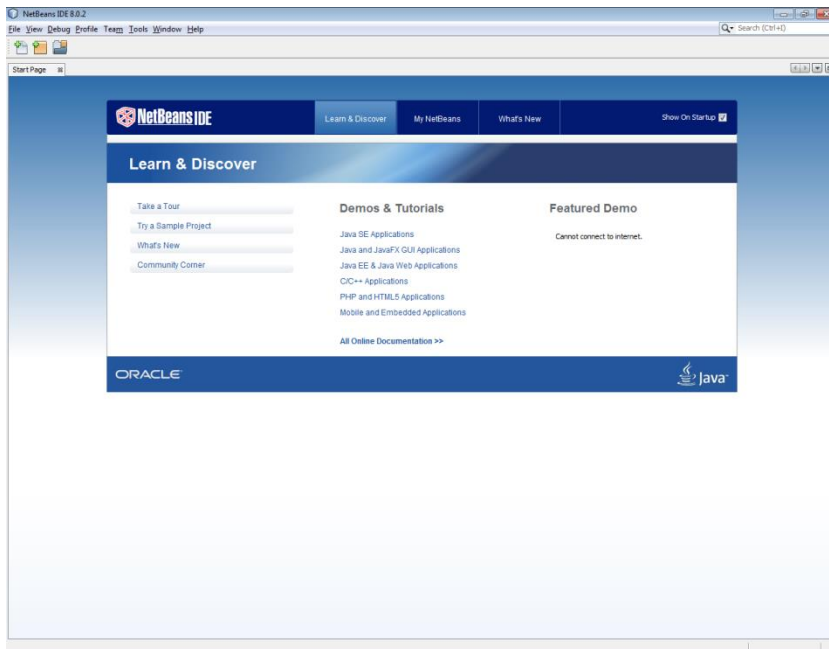
Java alapú programok fejlesztéséhez a Java fejlesztői készlet a JDK (Java Development Kit) szükséges. A JDK tartalmazza a JRE-t is, ezért a JDK telepítése esetén a JRE telepítése nem szükséges. A JRE Java bájtkódot tud futtatni, amelyet a Java programozási nyelven megírt forráskódból a JDK segítségével a Java fordító (javac - Java compiler) fordíthatunk le bájtkódra.

NetBeans IDE

A NetBeans integrált fejlesztői környezet az egyik legelterjedtebb szoftverfejlesztést támogató IDE Java programok fejlesztéséhez. A NetBeans fejlesztőkörnyezet a JDK-val együtt az Oracle oldaláról tölthető le 32 és 64 bites Mac OS X, Linux és Windows operációs rendszerekhez az alábbi linken: <http://www.oracle.com/technetwork/java/javase/downloads/index.html>.



9. ábra: *NetBeans IDE + JDK letöltése*



10. ábra: *Netbeans IDE kezdőképernyője*

Java programozás alapjai

A Java programozási nyelv is objektum orientált nyelv. Az objektum orientált nyelvek alapvető építőeleme az osztály (class), mely állapotot (adatok, változók) és viselkedéseket (metódus, függvény) foglal össze. Például egy „iskolai osztály” változója lehet az osztály létszáma, metódusa pedig az osztály érdemjegyei átlagának kiszámítása. Az objektum az osztály egy konkrét példánya, például az Arany János Általános Iskola 5.a osztálya.

Objektumorientált programozásban fontos összefüggés az öröklés, mely során a meglévő objektumokból levezetett (példányosított) újabb objektumok öröklik az alap objektumok egyes adatstruktúráit és függvényeit, ugyanakkor újabb tulajdonságokat is definiálhatnak, vagy akár öröklötet újraértelmezhetnek. Például egy „sportolókból álló iskolai osztály” leszármaztatható az „iskolai osztályból”, mely így öröklí az osztály létszáma változót és az érdemjegy átlag számító metódust. De új változót is definiálhat, például az osztály szerzett érmeinek száma, vagy új metódust is, például az érmeek alapján a legeredményesebb sportolót meghatározó függvény.

Az objektumok közötti kölcsönhatások következtében hatnak egymásra és változtatják meg állapotukat. Ezek a hatások, kommunikációk és ennek következtében létrejövő állapotváltozások adják lényegében a program lefutását. Az objektumok közötti kommunikáció üzenetek segítségével valósul meg, mely üzenetnek három fő része van: melyik objektummal, milyen műveletet, milyen adatokkal hajtson végre.

A Java alkalmazásban a C nyelvhez hasonlóan a *main* metódus (függvény) hívódik meg először úgy, hogy a Java értelmező először az adott osztály *main* metódusát hívja meg. Az alkalmazás további metódusait a *main* fogja meghívni. Java alkalmazásban tehát mindenképpen lennie kell egy alap osztálynak, ami a *main* metódust tartalmazza.

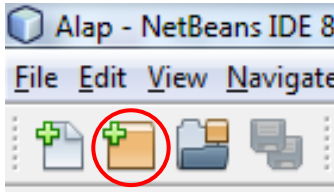
Egy „Hello” szöveget parancssorba kiíró alkalmazás programja az alábbi.

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

Az *Alap* nevű osztály tartalmazza a *main()* metódust, a *main()* metódus pedig tartalmazza az *Alap* osztály végrehajtható utasításait. A *public* azt jelöli, hogy az osztály és a *main* függvény más osztálybeli objektumokból is elérhető, meghívható, *static* pedig, hogy példányosítás nélkül is meghívható. A *main()* eljárás paramétere egy *args* nevű sztring tömb, amelynek elemei a program indításakor megadott parancssori paramétereket tartalmazzák. A „hello” szöveg kiírása a *System* osztály *out* (*PrintStream* osztálytípusú statikus) mezőjén

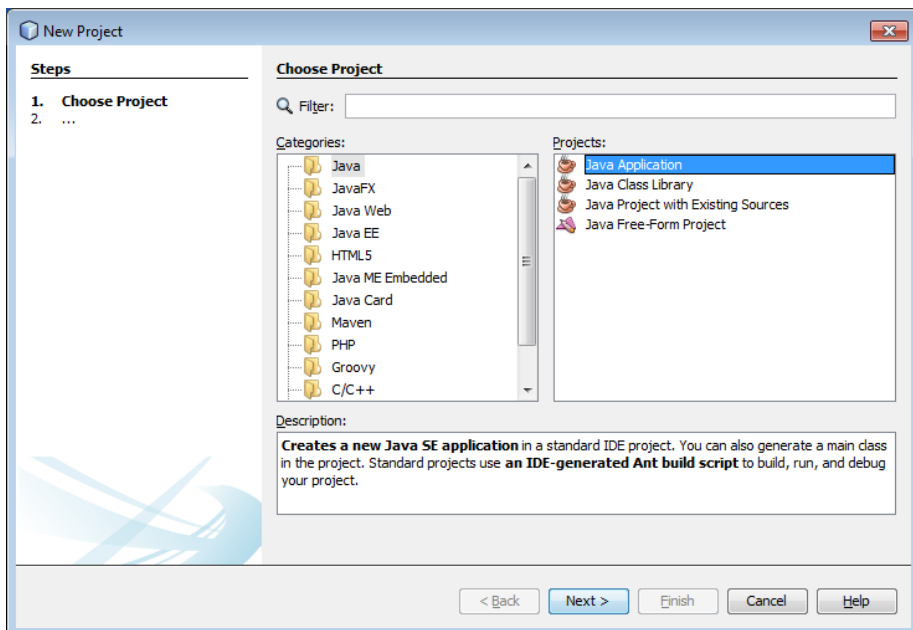
keresztül a `println()` metódus meghívása segítségével történik. A program a zöld színű háromszög ikonra történő kattintással fordítható le és indítható el.

A NetBeans IDE-ben egy új projekt létrehozásával lehet az előbb megadott egyszerű mintaprogramot lefordítani és futtatni. Új projekt a File menü New Project segítségével hozható létre, vagy a bal felső sarokban található ikonra kattintva (vagy Ctrl+Shift+N gyorsbillentyűvel).

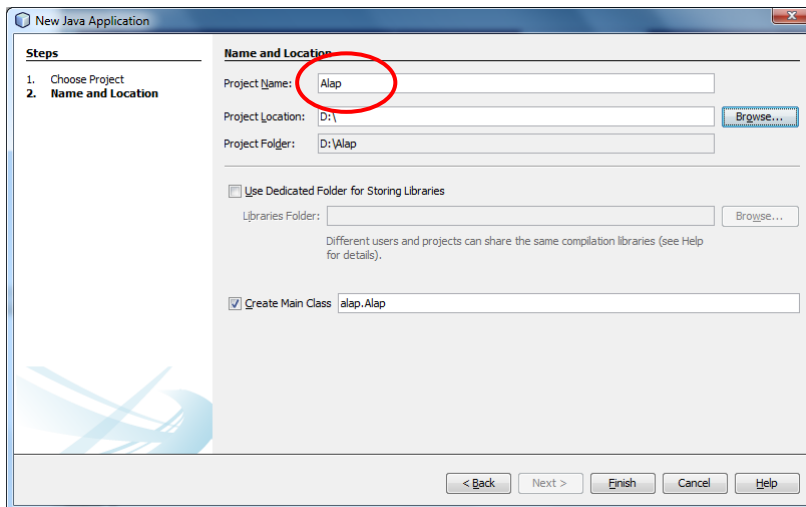


11. ábra: Új projekt létrehozása NetBeans IDE-ben

Az előugró ablakban a Java Application –t választva, majd a Next továbblépés gombra kattintva megadható a Projekt Name, vagyis a projekt neve, ami legyen Alap, és a projekt előre definiált elérési útja is módosítható, ha szükséges. A „Finish”-re kattintva létrejön a projekt.



12. ábra: Új Java alkalmazás projekt létrehozása NetBeans IDE-ben



13. ábra: Projekt nevének megadása

```

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  |  package alap;
7  |
8  |  /**
9  |   *
10 |   * @author Katona József
11 |   */
12 |  public class Alap {
13 |
14 |      /**
15 |       * @param args the command line arguments
16 |       */
17 |      public static void main(String[] args) {
18 |          // TODO code application logic here
19 |      }
20 |
21 |  }

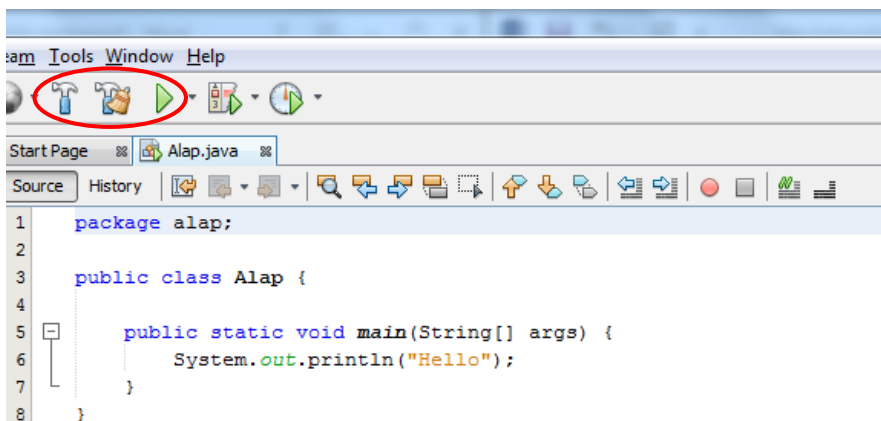
```

14. ábra: Program alapértelmezett forráskódja a projekt létrehozása után

A „Hello” szöveget parancssorba kiíró program kódszerkesztő részbe történő beírása után a program a Run menüpontban található Build Project (F11 gyorsbillentyű), Clean and Build Project (Shift+F11 gyorsbillentyű) fordítás után a Run Project (F6 gyorsbillentyű) segítségével futtatható. Ezek a funkciók ikonként is elérhetők.

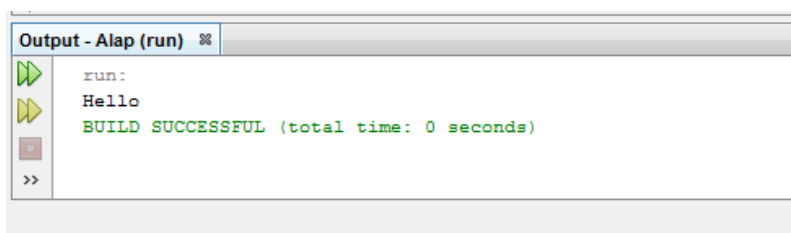


15. ábra: Build, Clean and Build és Run funkciók



16. ábra: Hello szöveget kiíró program kódja

A program fordítása és futtatása után a programszerkesztő alatti Output (Kimenet) ablakban jelenik meg a futás eredménye a „Hello” szöveg kiírása.



17. ábra: Program futásának eredménye

Amennyiben nem a *main*, hanem egy külön nem static *kiir()* metódusban van megvalósítva a „Hello” szöveg kiírása, úgy először a *new* operátorral példányosítani kell az *Alap* osztályt, majd utána kell a példány *kiir()* metódusát meghívni.

```

public class Alap {

    public void kiir(void) {          //kiir metodus
        System.out.println("Hello"); //Hello kiírása
    }

    public static void main(String[] args) { //main metódus
        Alap Hello_prog = new Alap(); //Hello_prog az Alap osztály
                                      //egy példány
        Hello_prog.kiir();             //a példány kiir metodusának meghívása
    }
}

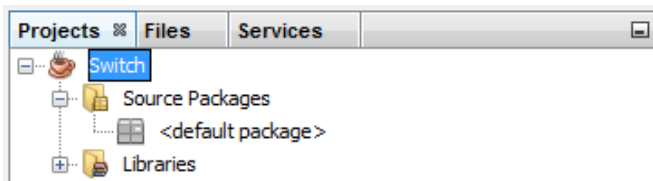
```

Grafikus felhasználói felület létrehozása Java alkalmazásban

Grafikus felhasználói felületet, a GUI-t (Graphical User Interface) a Netbeans IDE GUI szerkesztőjével, a GUI Builder segítségével egyszerűen össze lehet állítani. A GUI tartalmazza a gombokat, szövegmezőket és egyéb más grafikus elemet, melyek a felhasználó számára az átlátható egyszerű kezelést, paraméterek, állapotok jól átlátható megjelenítését valósítja meg. Grafikus felület létrehozható a Swing és AWT eszközkészlet segítségével, mindkettő az Oracle Java Foundation Classes (JFC) része, mely a Java alapú GUI komponenseket tartalmazza. A továbbiakban a Swing grafikus felhasználói felület készítésének alapjai kerülnek egy példaprogramon bemutatásra.

A bemutatásra kerülő példaprogramban egy kapcsoló állapotától függően kell egy feliratot OFF kikapcsolt vagy ON bekapcsolt értékre beállítani.

Grafikus felülettel rendelkező projektet az előző fejezetben bemutatott lépésekhez hasonlóan kell létrehozni New Project majd a Java Application-t választva. A projekt neve legyen Switch (kapcsoló), a projekt létrehozásakor a Create Main Class (fő osztály létrehozása) jelölőnégyzetből legyen kivéve a pipa, mivel az alkalmazás belépési pontját később adjuk meg.

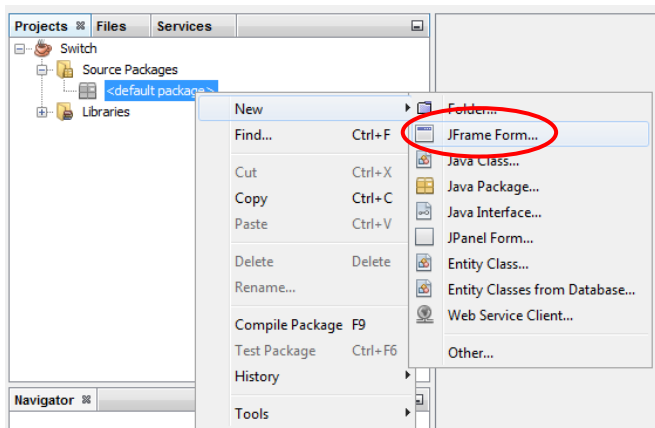


18. ábra: A Switch projekt felépítése a Projects ablakban

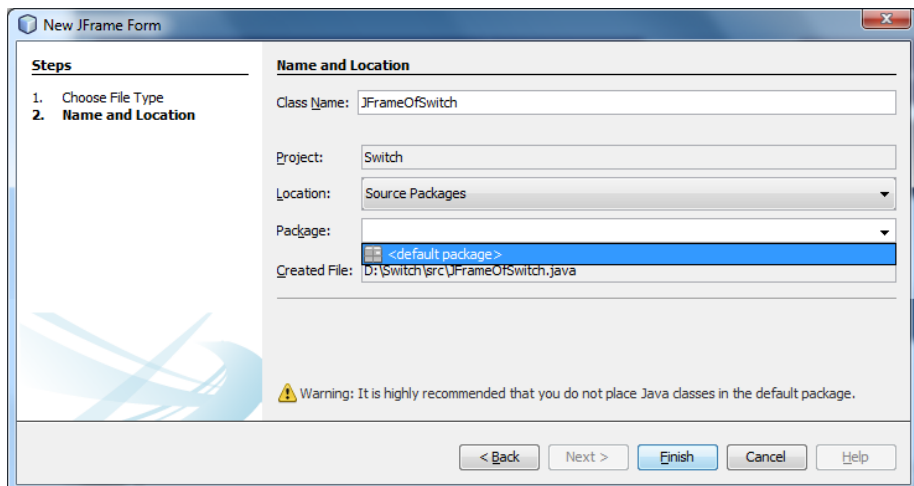
A fejlesztőkörnyezet a felhasználói felület összeállítása során egy háttér forráskódban a felhelyezett panelektől és vezérlőktől függően a szükséges kódot módosítja. Ez nagyban csökkenti a forráskód szerkesztést.

A felhasználói felület létrehozásához szükség van egy Java konténerre, egy *JFrame* objektum létrehozására, amelyre további vezérlőket (címke, beviteli mező, rádiógomb stb.) helyezhetünk. A konténer objektum létrehozásához a Projects ablakban kattintsunk jobb egérgombbal a '<default package>' (alapértelmezett csomag) nevére. A felbukkanó menüben a New almenü *JFrame* Form... menüpontra kattintva hozhatunk létre új *JFrame* típusú konténer objektumot.

A New *JFrame* Form dialógus ablakban megadhatjuk az osztály nevét, amely legyen *JFrameOfSwitch*, és hogy melyik csomagban hozzuk létre a konténert, ez legyen *switch*. A *JFrameOfSwitch* osztály a *JFrame* osztály egy leszármazottja.



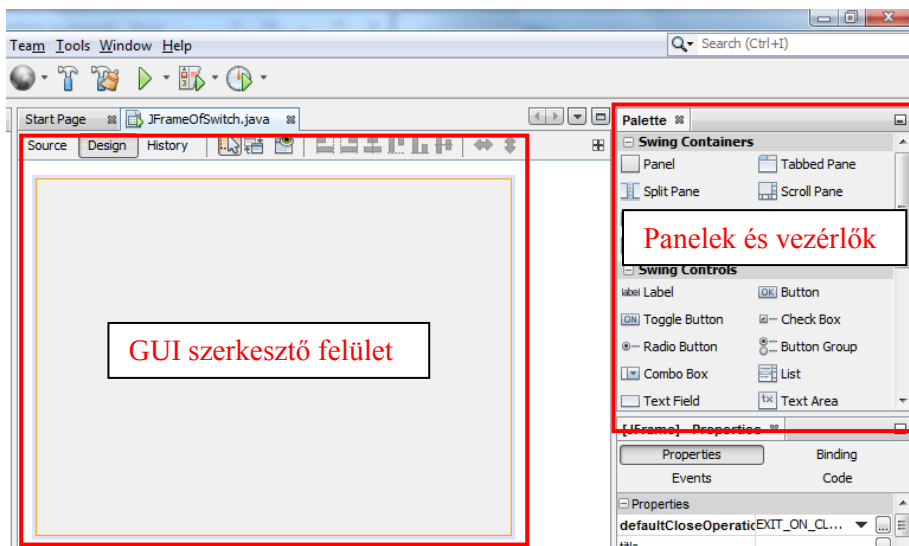
19. ábra: Új Form létrehozása



20. ábra: Új Form osztály és csomag nevének megadása

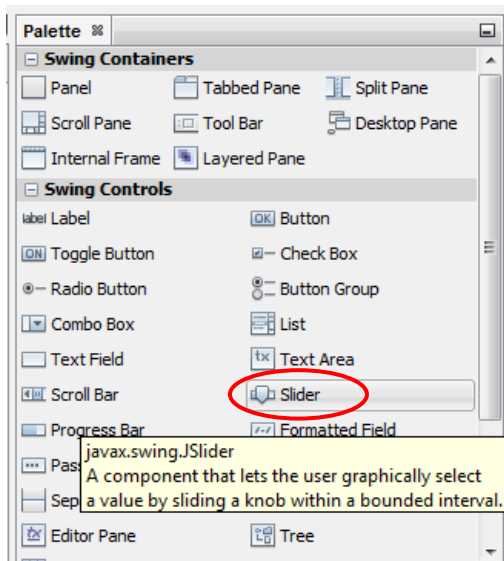
A paraméterek megadása után a Finish (befejezés) gombra kattintva, létrejön a JFrameOfSwitch nevű ablak, illetve a hozzá tartozó JFrameOfSwitch osztály a megadott switch nevű csomagon belül. Ezután a NetBeans GUI szerkesztő lesz aktív.

A GUI szerkesztőben alapértelmezetten a képernyő jobb oldali részén jelennek meg az egyes panelek, amelyek segítségével különböző ablakok hozhatók létre, oszthatók fel, görgető sávok stb. A vezérlők között nyomógomb, címke, különböző gombok, szövegezők, kiválasztó, listamező stb. található.

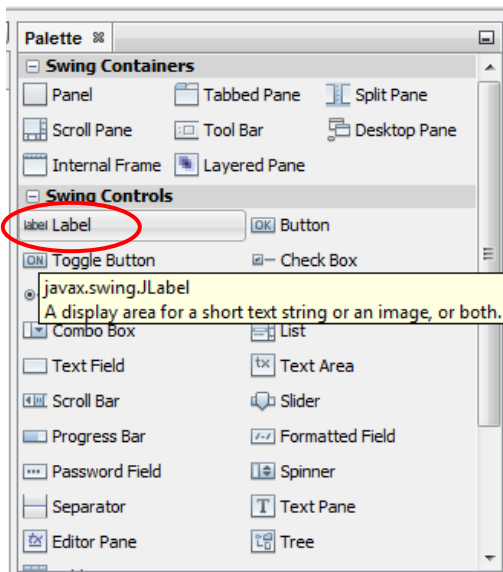


21. ábra: Bal oldalon a GUI tervező felület és jobb oldalon a panelek és vezérlők

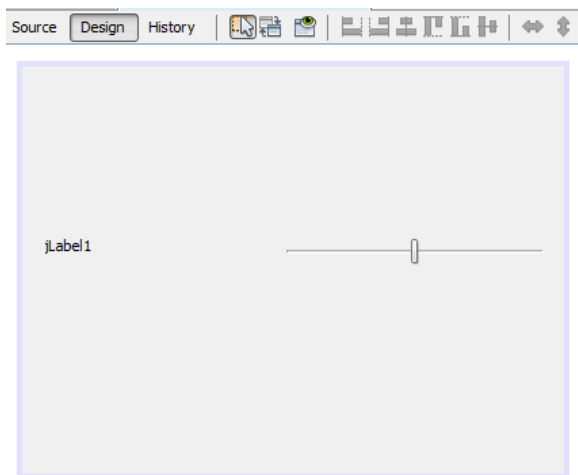
A példaprogramban a kapcsoló Slider vagyis csúszka nevű vezérlővel kerül megvalósításra, míg a feliratok Label vagyis címke segítségével kerülnek kiírásra. A csúszka a vezérlők között található, a bal egérgomb vezérlőn történő lenyomásával és nyomva tartásával áthúzható a GUI szerkesztő részre, és a gomb elengedésével lehelyezhető. Ugyanígy kell eljárni a címke esetén is.



22. ábra: Slider, csúszka típusú vezérlő kiválasztása

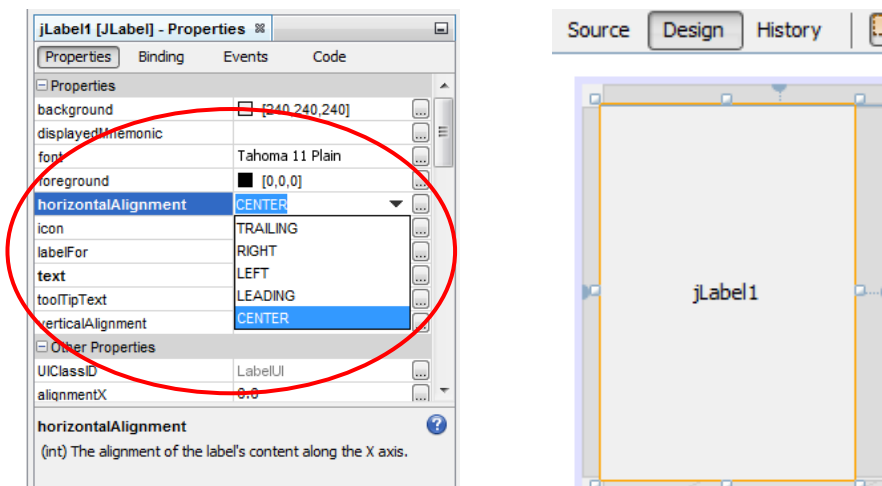


23. ábra: *Label, címke típusú vezérlő kiválasztása*



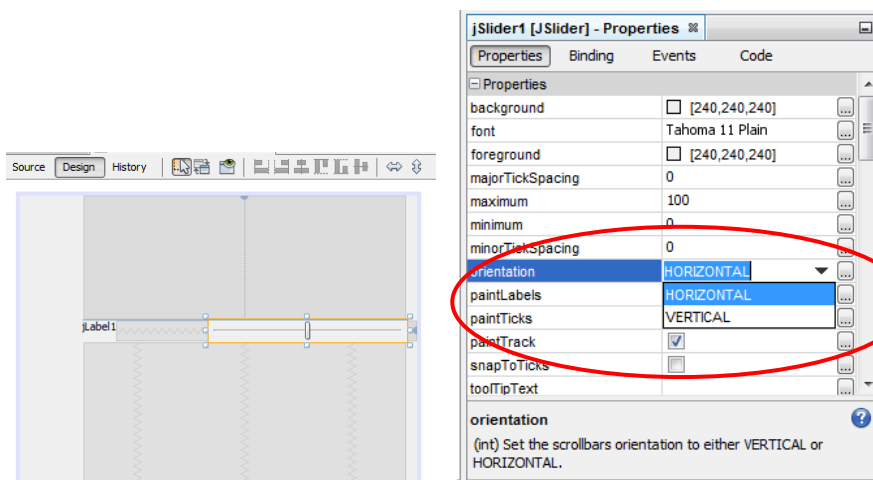
24. ábra: *Címke és csúszka elhelyezése*

A Java programban minden vezérlő egy objektum, amely rendelkezik tulajdonságokkal és képességekkel. A címke tulajdonságai a címke kiválasztása után a jobb alsó részen megjelenő a Properties (tulajdonságok) ablakban találhatóak. Amennyiben például módosítjuk a *horizontalAlignment* paramétert CENTER értékre, úgy a szöveg középre igazítva fog megjelenni a címkén.



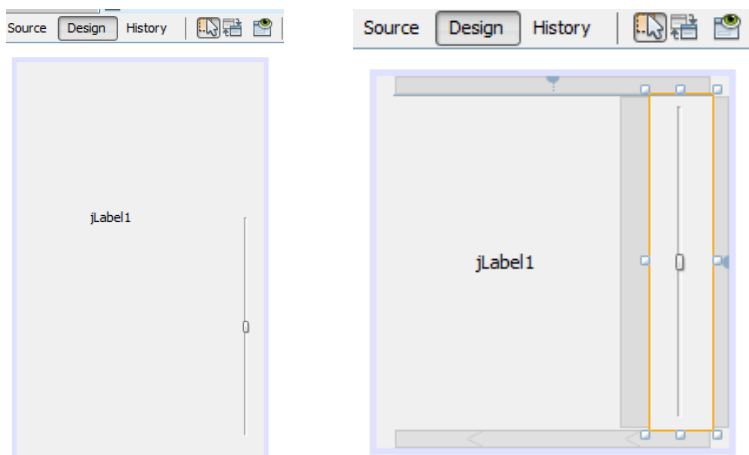
25. ábra: *jLabel1* nevű címke feliratának középre igazítása

Hasonlóképpen a csúszka függőleges helyzetűvé tétele a csúszka kiválasztása után az *orientation* paraméter VERTICAL értékre történő állításával valósítható meg. A címke és a csúszka is átméretezhető.



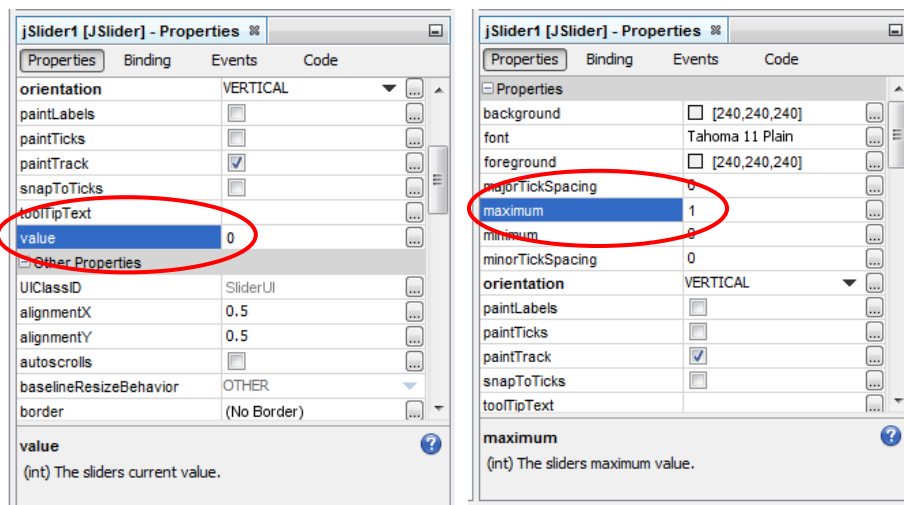
26. ábra: *jSlider1* nevű csúszka elforgatása az *orientáció* paraméter vertikálisra állításával

Az elforgatás után a csúszka áthelyezhető a csúszka kiválasztása és az egér bal gombjának lenyomása közben történő elmozdítással. Az egérgomb elengedésével a csúszka letehető.



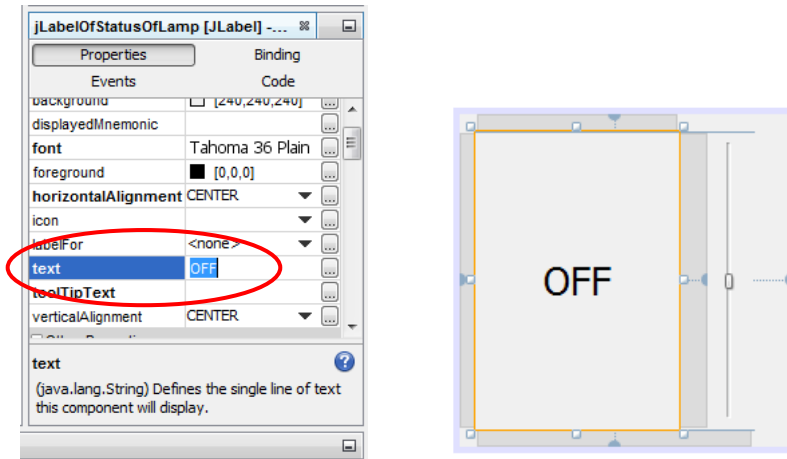
27. ábra: Csúszka elhelyezkedésének módosítása

Egy normál kapcsoló kétállapotú, ezt kellene a csúszka segítségével is megvalósítani. Tehát úgy kellene beállítani a csúszka paramétereit, hogy alapértelmezetten kikapcsolt, majd a csúszka elmozdításával bekapcsolt állapotot vegyen fel. Ezt legegyszerűbben úgy lehet megvalósítani, hogy a csúszka alapértelmezett *value* értéke 0-ra, a *maximum* értéke 1-re kerül beállításra (csúszka csak egész értékeket ad vissza).



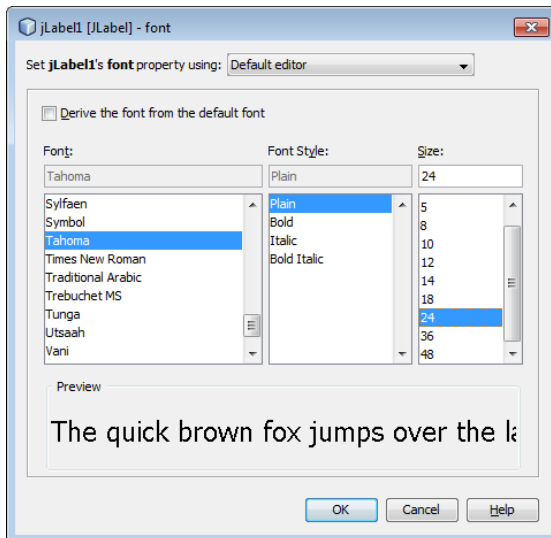
28. ábra: Csúszka maximális értékének 1-re, kezdeti értékének 0-ra állítása

A címke kezdeti értékét, a *text* paraméterét is célszerű módosítani „OFF”-ra, mivel a csúszka kezdeti értéke is 0.



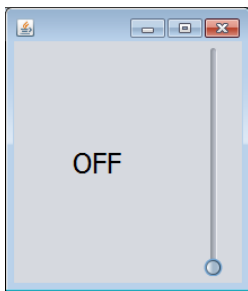
29. ábra: Címke alapértelmezett szövegének „OFF” értékre állítása

A címke betűtípusa és mérete a *font* paraméter segítségével módosítható.



30. ábra: Címke betűtípusának és betűméretének beállítása

A projekt fordítása majd futtatása után megjelenő ablakban megjelenik a címke felirata és a csúszka.

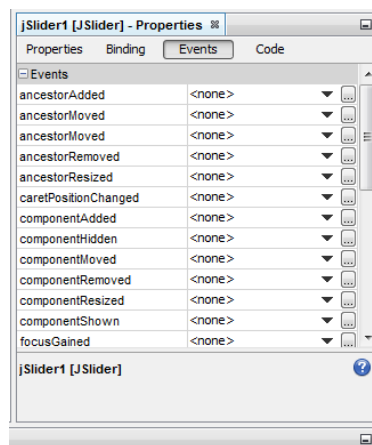
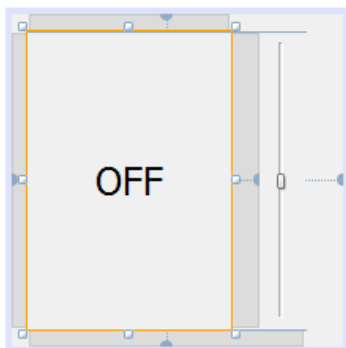


31. ábra: Java program futtatása

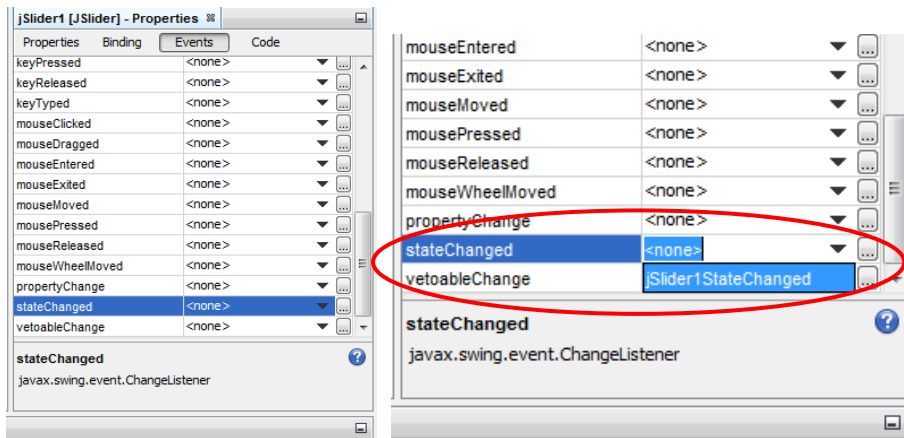
Eseménykezelés Java alkalmazásban

Ahhoz, hogy a csúszkán történt változáshoz valamilyen funkciót lehessen rendelni, szükség van a csúszka változásának, mint eseménynek a kezelésére. Ehhez egy eseményfigyelőt kell definiálni, amely a csúszka megváltozásakor kiváltott eseményre hívódik meg. Az esemény hatására tehát egy függvény hívható meg, amely lekérdezheti a csúszka helyzetét és akár ez alapján módosíthatja a címke feliratát is.

A következőkben egy egyszerű eseménykezelés kerül bemutatásra, amely a csúszka megváltozásakor hívódik meg és a csúszka alsó helyzetében „OFF” ki, felső helyzetében „ON” be szövegre változtatja a címke szövegmezőjét. Jelöljük ki a csúszka vezérlőt, majd a Properties tulajdonságok között kattintsunk az Events események nevű fülre. A vezérlő különböző állapotainak megváltozása eltérő eseményeket generál, válasszuk ki a *statusChanged*, a státusz megváltozott eseményt, és a *jSlider1StatusChanged* eseménykezelő függvényt, amely az esemény bekövetkezésekor fog meghívódni.

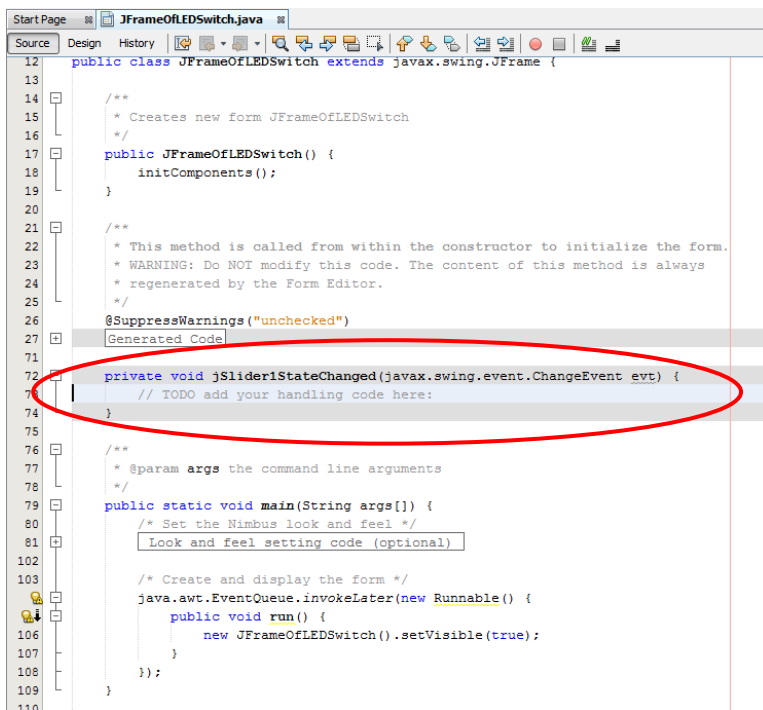


32. ábra: Csúszka vezérlőhöz rendelt események listája



33. ábra: Csúszka státuszának megváltozása (*statusChanged*) eseményhez (*Events*) eseménykezelő függvény megadása (*Slider1StateChanged*)

Az eseménykezelő kiválasztás után a programkód szerkesztőben a *jSlider1StatusChanged* függvény törzse automatikusan beillesztésre kerül a forráskódba.



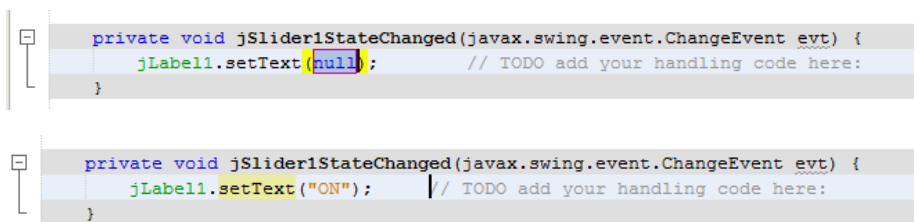
34. ábra: *jSlider1StatusChanged* eseménykezelő függvény a kódszerkesztőben

A csúszka állapotának megváltozása eseménykor a *jLabel1* nevű címke szövegmezőjét kívánjuk megváltoztatni a *jSlider1* csúszka állapotának függvényében. A *jLabel1* címke neve után írt „.” karakter leütésekor – az IDE környezetbe beépített IntelliSense-nek köszönhetően – megjelennek a *jLabel1* címke tulajdonságai és képességei, amelyek között a *setText()* metódus alkalmas a címke szövegének megváltoztatására, amely paraméterül egy String típusú szöveget vár.

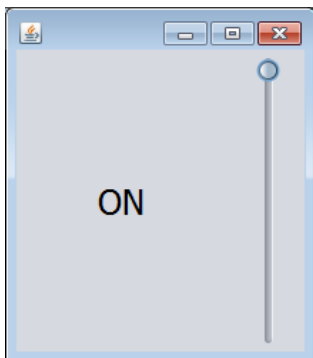


35. ábra: *jLabel1* címke metódusai

A *setText()* kiválasztása után a „null” paraméter helyén kell megadni a kívánt szöveget.



A program lefordítása és futtatása után, ha a csúszka állapota megváltozik, akkor a címke felirata „ON”-ra változik.

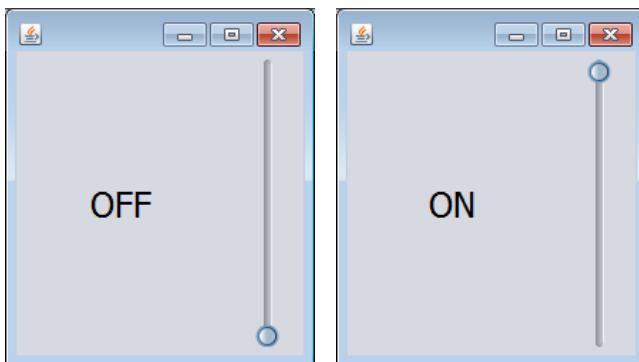


36. ábra: Címke feliratának „ON”-ra módosulása a csúszka állapotváltozásakor

A programot egy if-else feltételes vezérlési szerkezettel és a `JSlider1` értékének `getValue()` metódussal történő lekérdezésével a csúszka helyzetétől függően lehet beállítani a címke feliratát. Ha a csúszka értéke 0, akkor „OFF”, ha 1 akkor „ON” értékre.

```
private void jSlider1StateChanged(javax.swing.event.ChangeEvent evt) {
    if (jSlider1.getValue() == 1) jLabel1.setText("ON");
    else jLabel1.setText("OFF");
}
```

A program fordítása és futtatása után csúszka helyzetétől függően lesz „OFF” vagy „ON” a címke felirata.



37. ábra: Címke feliratának a csúszka helyzetétől függő módosulása

Soros kommunikáció

A számítástechnikában a soros port egy fizikai soros kommunikációs interfészt jelöl, melyen keresztül soros módon, vagyis bitenként történik az adatok átvitele. Ez az adatátviteli mód az iparban is elterjedten alkalmazott.

Mivel az Ethernet, a FireWire vagy épp az USB is soros adatfolyamként továbbítja az adatokat, ezért a soros porton legtöbbször az RS-232 szabvány szerinti kommunikációt értik. A számítógépek esetén az UART integrált áramkör végzi az adatok aszinkron soros adatfolyammá történő átalakítását, de ezek az áramköri egységek ma már a mikrovezérlőkben is integráltan megtalálhatóak.

Az Arduino Nano fejlesztőlapon található ATmega328 mikrovezérlőben is integráltan megtalálható az UART egység, melynek jelszintje 5V. A legtöbb mai számítógép már hagyományos RS-232 soros portot nem tartalmaz, csak USB-t, ezért a számítógéppel történő kommunikáció létrehozása céljából egy USB-UART átalakítóra van szükség. Az Arduino Nano ezt az átalakítót tartalmazza, mely a legtöbb esetben a korábban már említett FTDI gyártmányú.

Soros kommunikáció kezelése Arduino alkalmazásban

Arduino fejlesztőlapok esetén a soros kommunikáció a 0-ás és 1-es digitális be- és kimeneteken keresztül valósul meg, ezért ha soros kommunikáció használatban van, úgy ezen kivezetéseket nem szokás más célra felhasználni. A soros kommunikáció kezelése a *Serial* függvénykönyvtár segítségével lehetséges. Az inicializálást végző *setup()* függvényben szokás a soros kapcsolatot megnyitni és a kommunikáció sebességét beállítani a *Serial.begin()* függvény segítségével. Adatok soros vonalon történő olvasására és írására, állapotának lekérdezésére több függvény is alkalmazható. A legtöbbet használt függvények olvasásra a *Serial.read()*, írásra a *Serial.print()*, *Serial.write()*, a kiolvasható adatok mennyiségének lekérdezésére pedig a *Serial.available()*.

A soros kommunikáció kezelése egy egyszerű visszhang (echo) program segítségével könnyen bemutatható. A *setup()* függvényben kerül a soros kapcsolat megnyitásra és a sebesség 9600 bps (bit per secundum – bit per másodperc) értékre beállításra. A *loop()* függvényben a *Serial.available()* függvény segítségével kerül megvizsgálásra, hogy érkezett-e adat a soros kapcsolaton, ha igen, akkor az a *Serial.read()* függvény segítségével beolvasásra kerül és a *Serial.write()* függvénnyel pedig visszaküldésre a soros kapcsolaton keresztül visszaküldés.

```

//Soros kommunikacio - echo program

void setup() //inicializalas
{
    Serial.begin(9600); //soros kapcsolat megnyitása,
                        //kapcsolat sebességeinek megadása: 9600 bps
}

void loop() //cilikusan futo program
{
    byte data;
    if (Serial.available()) //ha van olvashato adat a soros vonalon
    {
        data = Serial.read(); //adat beolvasasa
        Serial.write(data); //beolvasott adat kuldese
    }
}

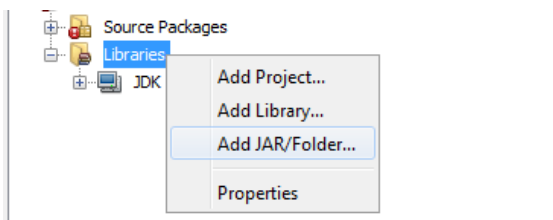
```

Soros kommunikáció kezelése Java alkalmazásban

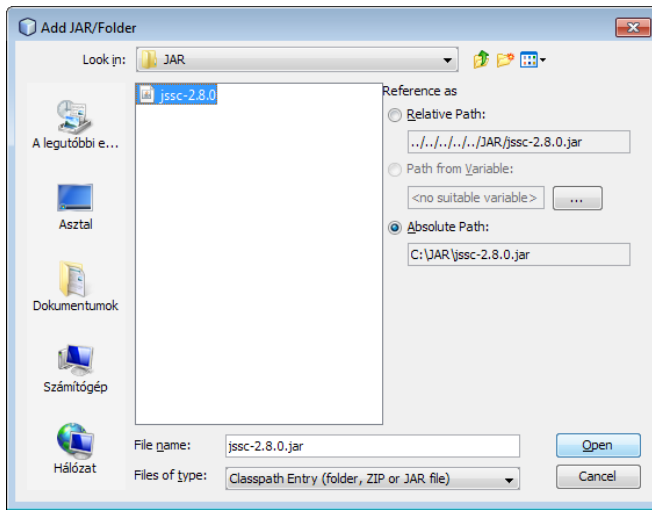
A Java Communication API, mely többek között a soros port kezelését is biztosítja, csak Solaris és Linux operációs rendszereket támogatja. Windows és Linux operációs rendszerek támogatása esetén a JSSC (Java Simple Serial Connector – Java egyszerű soros csatlakozó) vagy RXTX könyvtárakat alkalmazzák. A következőkben a soros kommunikáció kezelése a JSSC, Windows, Linux, Solaris és Mac OS X operációs rendszereket támogató Java könyvtár alkalmazása kerül bemutatásra egy egyszerű példán keresztül.

A JSSC könyvtár a projekt honlapján érhető el: <http://code.google.com/p/java-simple-serial-connector/>. A következő mintapélda a soros kommunikáció alapvető kezelését mutatja be a JSSC könyvtár felhasználásával egy visszhang alkalmazás megvalósításán keresztül.

Hozzunk létre egy új SerialEcho nevű projektet. A jssc-X.X.X.jar fájl (jelenleg jssc-2.8.0.jar) letöltése után a NetBeans fejlesztőkörnyezetbe a létrehozott projekt Libraries (könyvtárak) eleméhez adjuk.



38. ábra: Függvénykönyvtár hozzáadása a projekthez



39. ábra: JSSC JAR függvénykönyvtár kiválasztása

Az import segítségével lehet a JSSC függvénykönyvtár elemeit használni. A `SerialPort` és a hozzá kapcsolódó hibakezelés végző `SerialPortException` osztályok importálása után az osztályok a program számára használhatóak lesznek. Az import sorok forráskódba történő beírása után a sárga aláhúzás azt jelzi, hogy a forráskódban még ezen osztályok nincsenek használatban.

```

1  package serialecho;
2
3  import jssc.SerialPort;
4  import jssc.SerialPortException;
5
6  public class SerialEcho {
7
8      public static void main(String[] args) {
9
10     }
11 }

```

40. ábra: JSSC függvénykönyvtár importálása

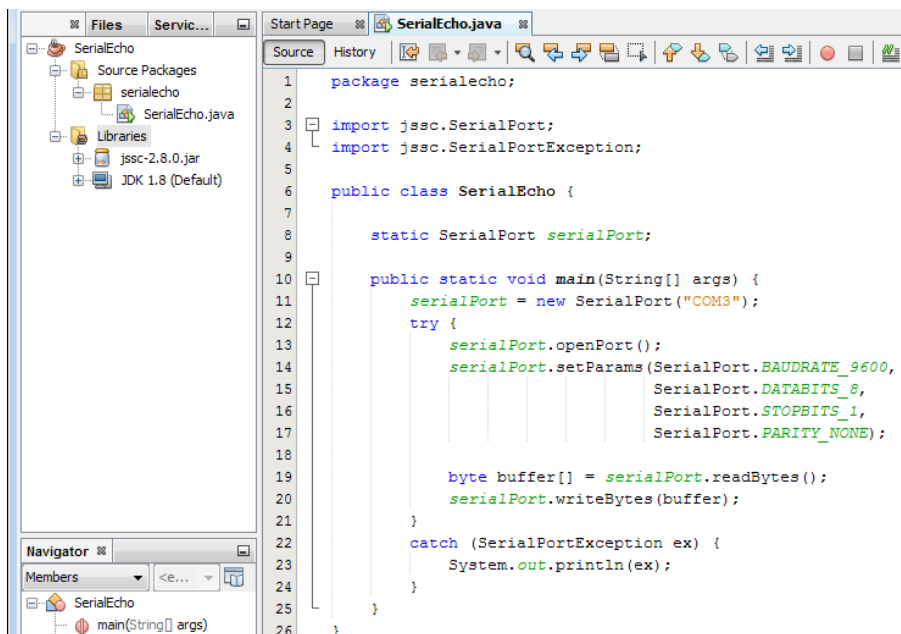
A soros kommunikáció kezeléséhez olvasni és írni kell tudni a soros portot, melyhez az többek között a `readBytes()`, illetve a `writeBytes()` vagy `writeString()` függvények használhatóak. A soros port kezeléséhez első lépésben létre kell hozni a soros portot kezelő `serialPort` objektumot, majd az `openPort()` metódussal lehet megnyitni a portot és a `setParams()` függvénnyel pedig a kommunikáció paramétereit lehet megadni.

A soros port kezelése közben fellépő hibák az úgynevezett kivételkezeléssel kerülnek feldolgozásra. A kivételkezelés segítségével kezelni lehet a futási időben bekövetkező hibákat. A kivételek kiváltására és kezelésére használt kulcsszavak a try és a catch. Azokat az utasításokat, amelyek kivételt küldhetnek az úgynevezett try (próbáld) blokkba kell elhelyeznünk. Amennyiben a try blokkban kivétel keletkezik azt a catch (elkapni) segítségével észlelhetjük és megadhatjuk, hogy milyen utasítások hajtsódjanak végre a kivétel fellépésekor. Amennyiben a try blokkban nem váltódik ki kivétel, úgy a catch ág nem fog lefutni. A try blokkhoz akár több catch ág is tartozhat, amelyek különböző hibák fellépésekor hajtsódnak végre.

Az alábbi egyszerű példaprogram *main()* függvényében a *serialPort* objektum *new* kulcsszóval kerül létrehozásra, majd az *openPort()* metódussal megnyitásra és a soros kommunikáció paraméterei a *setParams()* metódussal kerültek beállításra:

- sebesség 9600 baud
- adatbitek száma 8
- stop bitek száma 1
- paritás nincs

A soros porton beérkezett adatokat a *readBytes()* függvénnyel egy buffer változóba beolvassa majd a *writeBytes()* függvénnyel kiírja a soros portra és a program kilép.



41. ábra: Egyszerű soros port kezelés JSSC könyvtár segítségével

A soros port olvasása történhet eseményvezérléssel is, amikor egy esemény kiváltása lehet maga a soros porton érkező adat. Ehhez az eseményhez is egy eseménykezelő függvény rendelhető, mely az esemény bekövetkezésekor hívódik meg. A soros porton érkező események észleléséhez és feldolgozásához a `SerialPortEvent` és `SerialPortEventListener` osztályok importálása szükséges.

A soros porton adat érkező eseményt a `SerialDataProcess` osztály dolgozza fel, amely a `JSSC SerialPortEventListener` osztály egy megvalósítása. Az eseménykezelőt az `addEventListener()` a `SerialDataProcess` objektumon keresztül hívja meg az adat érkező esemény bekövetkezésekor, melyet az `RXFLAG` jelez. Az eseményre az objektumon belül a `serialEvent()` hívódik meg. Ebben kell megvalósítani a soros port olvasását a `readBytes()`, írását a `writeBytes()` metódusokkal. A visszhang program kódja:

```
//Soros kommunikacio - Java echo program
package serialecho;
//JSSC importok
import jssc.SerialPort;
import jssc.SerialPortException;
import jssc.SerialPortEvent;
import jssc.SerialPortEventListener;

public class SerialEcho {
    static SerialPort serialPort; //SerialPort objektum

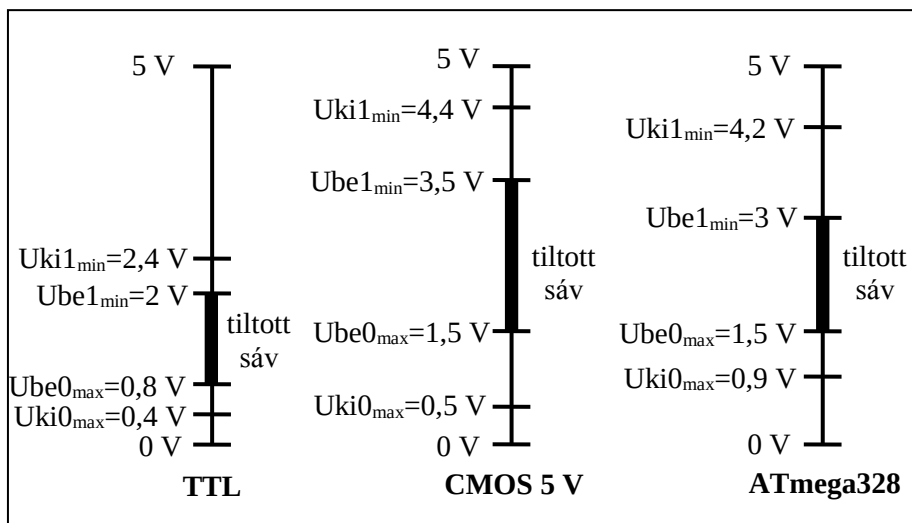
    public static void main(String[] args) {
        serialPort = new SerialPort("COM3"); //SerialPort objektumpéldány
        try {
            serialPort.openPort(); //soros port megnyitása
            //soros kommunikáció paramétereinek beállítása
            serialPort.setParams(SerialPort.BAUDRATE_9600,
                                SerialPort.DATABITS_8,
                                SerialPort.STOPBITS_1,
                                SerialPort.PARITY_NONE);
            //eseménykezelő hozzárendelése az adat érkező eseményhez
            serialPort.addEventListener(new SerialDataProcess(),
                                       SerialPort.MASK_RXFLAG);
        } //hibakezelés, hiba parancssorba kiírása
        catch (SerialPortException ex) {
            System.out.println(ex);
        }
    }
    //eseménykezelő osztály megvalósítása
    static class SerialDataProcess implements SerialPortEventListener {
        public void serialEvent(SerialPortEvent event) {
            try { //eseménykezelő függvényben soros port olvasása, írása
                byte buffer[] = serialPort.readBytes();
                serialPort.writeBytes(buffer);
            } //hibakezelés, hiba parancssorba kiírása
            catch (SerialPortException ex) {
                System.out.println(ex);
            }
        }
    }
}
```

Digitális be- és kimenetek és kezelésük

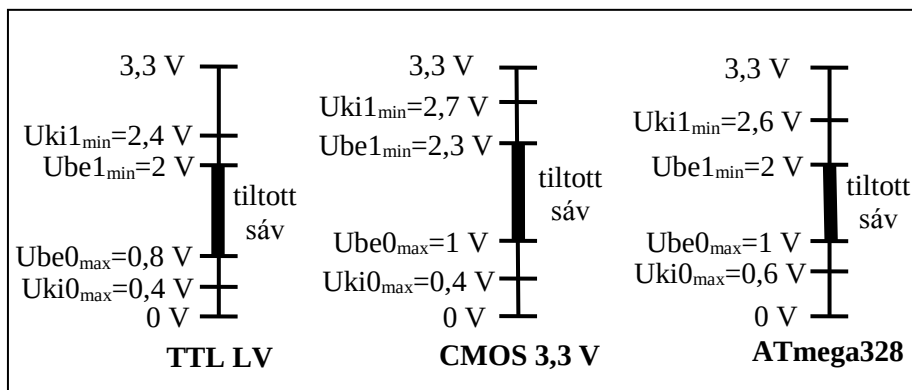
Általánosságban egy rendszer be- és kimenetein keresztül tartja a külvilággal a kapcsolatot, bemenetein kap információkat, azokat feldolgozza, és a kimenetén továbbítja. Egy összetett rendszert egyes részegységeket reprezentáló rendszerek és azok egymással való kapcsolata adja. A kapcsolatokban az egyik rendszer kimenete egy másik rendszer bemenete. Például egy számítógép esetén a billentyűzet által kiadott jelsorozat a számítógép számára bemenet, míg a nyomtatóra küldött kódsorozat a számítógép számára kimenet, míg a nyomtató számára bemenet. A mai irányító berendezések szinte kivétel nélkül digitálisak, vagyis digitális kétállapotú jeleket, azokból felépülő kódokat fogadnak, dolgoznak fel és állítanak elő. Ezért a digitális, logikai-szintű be- és kimenetek és feldolgozó egységek a digitális rendszerek legalapvetőbb építőelemei. A digitális rendszerekben két logikai szintet alkalmaznak, az alacsony és a magas, szokás igaz és hamis vagy 0 és 1 szerint megadni. A digitális rendszerekben a logikai alacsony és magas szintekhez bizonyos feszültségtartományokat rendelnek. Az alacsony szint a 0 V közeli feszültségnek felel meg, a magas szint pedig beágyazott rendszerek esetén a legtöbb esetben 5 V vagy 3,3 V közeli feszültség. Az iparban alkalmazott vezérlőrendszerek esetén a jobb zajtűrés, nagyobb teljesítmény miatt nagyobb feszültség szinteket alkalmaznak, általában 24 V-ot. Az egyes eltérő logikai feszültség szintekkel rendelkező digitális egységek között jelszint illesztést kell alkalmazni. A következőkben a digitális be- és kimenetek kezelése kerül bemutatásra Arduino fejlesztőlap segítségével.

Digitális be- és kimenetek jellemzői

Az Arduino fejlesztőlapok többségén az Atmel mikrovezérlője található, legtöbb fejlesztőlapokon az Atmel ATmega328 típusú mikrovezérlője. A digitális jelszintek tartományai a legelterjedtebben alkalmazott TTL (Transistor-transistor logic – Tranzisztor-tranzisztor logika), CMOS (Complementary metal-oxide-semiconductor – Komplementáris fém-oxid félvezető) technológiák és az ATmega328 mikrovezérlő esetén az alábbi ábrán látható 5 V és 3,3 V tápfeszültség esetén. Hagyományos TTL technológia tápfeszültsége fix 5 V, ezért a TTL jelszintek csak 5 V tápfeszültség esetén értelmezettek. Az egyes technológiák esetén a magas és alacsony logikai értékekhez feszültségtartományokat rendelnek. A be- és kimenetek esetén ezekhez a logikai szintekhez rendelt a feszültségtartományok eltérőek. A különböző terhelési viszonyok miatt a kimenetre szűkebb tartományt írnak elő, mint a bemenetre. A logikai szintek esetén a magas jelszint minimuma és az alacsony jelszint maximuma között van egy tiltott sáv, amelyet az egyértelmű működés, jelszint azonosítás miatt a digitális jel nagysága nem vehet fel.



42. ábra: TTL, CMOS technológiák és az ATmega328 mikrovezérlő digitális jelszintjei 5 V tápfeszültség esetén



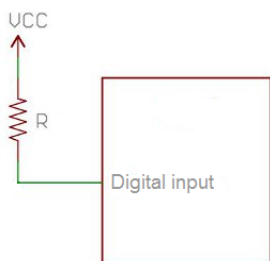
43. ábra: TTL LV, CMOS technológiák és az ATmega328 mikrovezérlő digitális jelszintjei 3,3 V tápfeszültség esetén

Látható, hogy a bemeneti jel esetén TTL jelszintet figyelembe véve 1,2 V feszültségkülönbség van a magas szint minimális és az alacsony szint maximális értéke között. Arduino ATmega328 mikrovezérlője esetén ez a különbség 1,5 V, kicsit nagyobb. Másrészt az is megfigyelhető, hogy az egyes technológiák jeltartományai is eltérőek, amennyiben az egyik technológiával felépített digitális rendszert össze kell kapcsolni a másikkal, úgy jelszint illesztést kell beiktatni. Egyszerűbb esetekben, azonos tápfeszültségű áramkörök esetén a

jelszint/jeltartomány illesztés elhagyható, de ez a rendszer zajjal szembeni hibatűrősségét csökkenti.

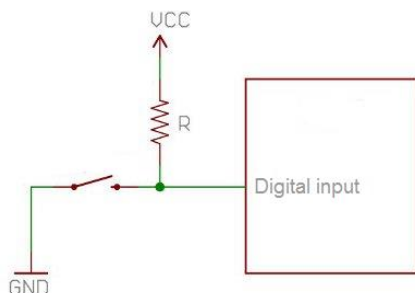
A kimeneti áram tekintetében az Arduino ATmega328 mikrovezérlője esetén arra kell ügyelni, hogy az egyes portok kimeneti árama összesen ne haladja meg a 100 mA-t és ajánlatos az egyes digitális kimenetek esetében a maximális 20 mA kimeneti áramot nem túllépni. Ez azt jelenti, hogy 5 V tápfeszültség esetén a kimenetre és a tápfeszültség vagy a föld közé ne kössünk $R = 5 \text{ V} / 20 \text{ mA} = 250 \text{ } \Omega$ értéknél kisebb ellenállást, legalább 470 Ω vagy 1 k Ω ellenállás alkalmazása célszerű.

A digitális bemeneteket nem szabad bekötetlenül hagyni, mivel a beolvasott állapot a külső zajtól függően változhat, nem fix érték. Ezért a nem állandó jelleggel bekötött bemeneteket fix feszültség szintre, földre (GND) vagy tápfeszültségre (VCC) szokás kötni egy ellenálláson keresztül. Amennyiben a digitális bemenetet magas szintre, tápfeszültségre, kötik egy ellenálláson keresztül, akkor ezt az ellenállást felhúzó ellenállásnak nevezik.



44. ábra: *Digitális bemenet fix tápfeszültségre kötése felhúzó ellenálláson keresztül*

A felhúzó ellenállás egyik tipikus alkalmazása, ha a bemenetre egy kapcsoló csatlakozik. Egészen addig, amíg a kapcsoló nincs bekapcsolva, addig a bemenet fix magas szintet kap a felhúzó ellenálláson keresztül. Amint a kapcsolót bekapcsoljuk, a bemeneti jelszint közel nullára csökken, vagyis a logikai érték alacsony lesz.



45. ábra: *Kapcsoló illesztése digitális bemenethez*

Arduino digitális be- és kimenetek kezelése

Az Arduino fejlesztőlapok digitális be- és kimeneteinek kezelése programkódból viszonylag egyszerű. Az egyes digitális be- és kimenetek mindegyike a *setup()* függvényben konfigurálható akár bemenetként akár kimenetként a *pinMode()* függvény segítségével.

2-es kivezetés digitális bemenet: `pinMode(2, INPUT);`

2-es kivezetés digitális kimenet: `pinMode(2, OUTPUT);`

Alapértelmezetten minden digitális be- és kimenet bemenetként van konfigurálva. Digitális bemenet igen nagy bemeneti ellenállású, közel 100 M Ω . Ennek következtében nagyon kicsi áram elegendő a bemeneti feszültség szint megváltozásához, tehát a bekötetlen digitális bemenet logikai értéke zajtól erősen függ. A mikrovezérlő 20 k Ω és 50 k Ω közötti értékű beépített felhúzó ellenállást is tartalmaz, melyet szintén a *pinMode* segítségével lehet konfigurálni, természetesen ebben az esetben az adott kivezetés bemenetként funkcionál.

2-es kivezetés digitális bemenetként történő konfigurálása felhúzó ellenállás alkalmazásával: `pinMode(2, INPUT_PULLUP);`

A D13-as kivezetésre egy sorosan kötött ellenállás és LED csatlakozik, ezért bemenetként felhúzó ellenállással történő konfigurálása esetén a bemeneti jelszint 1,7 V körüli lesz, nem magas szint. Ezt a kivezetést digitális bemenetként történő használat esetén célszerű a bemenet és föld közé bekötött külső ellenállás segítségével fix alacsony értékre kötni.

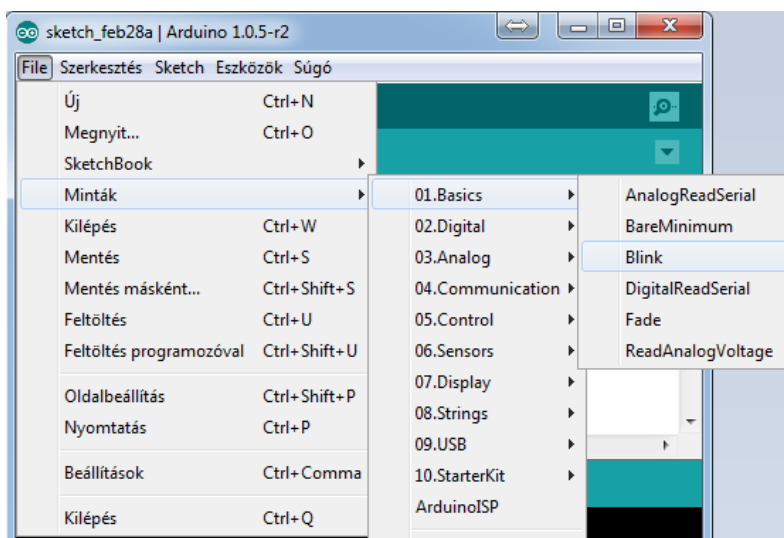
Digitális kimenetként történő konfigurálás esetén érdemes különösen odafigyelni arra, hogy túl nagy kimenő áram, esetleg rövidzár ne fordulhasson elő, mert az az adott kimenetet tönkretetheti.

A *loop()* függvényben a *digitalRead()* és *digitalWrite()* függvények segítségével olvasható vagy állítható be a digitális be- vagy kimenet értéke a HIGH vagy LOW logikai boolean érték.

2-es kivezetés digitális bemeneti értékének beolvasása: `digitalRead(2);`

2-es kivezetés digitális kimeneti értékének beállítása: `digitalWrite(2, HIGH);`

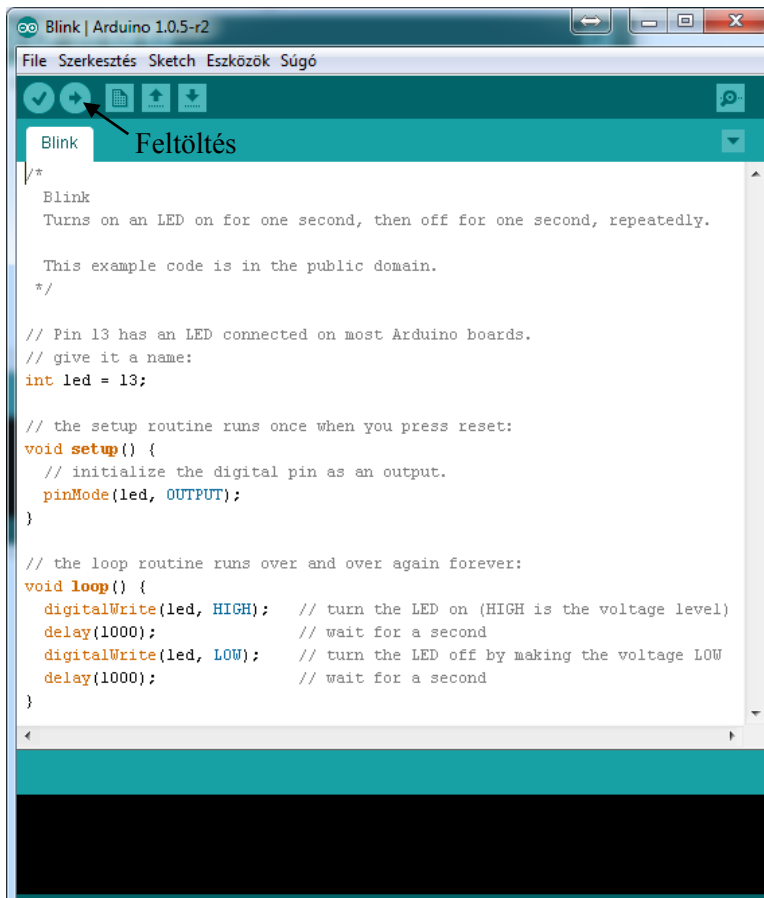
A digitális kimenetek programban történő kezelésére jó példa a mintaprogramok között található LED villogtató algoritmus. A File/Minták menüpontban található 01.Basics (alapok) almenüben található Blink-re (villog) kattintva hívható be a program kódja.



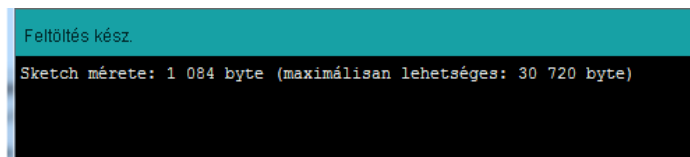
46. ábra: *Blink (villogó) mintaprogram betöltése*

A Blink program kódján látható, hogy C nyelven íródott, a programba tett kommenteket „/” után vagy „/*” „*/” közé kell elhelyezni. Látható a `setup()` és `loop()` függvények definiálása is.

A program a „jobbra nyíl” Feltöltés ikon-ra történő kattintással fordítható le és tölthető fel az eszközre (amely egyébként sok esetben az újonnan vásárolt eszközre már fel van töltve, ezért villog folyamatosan egy piros LED).



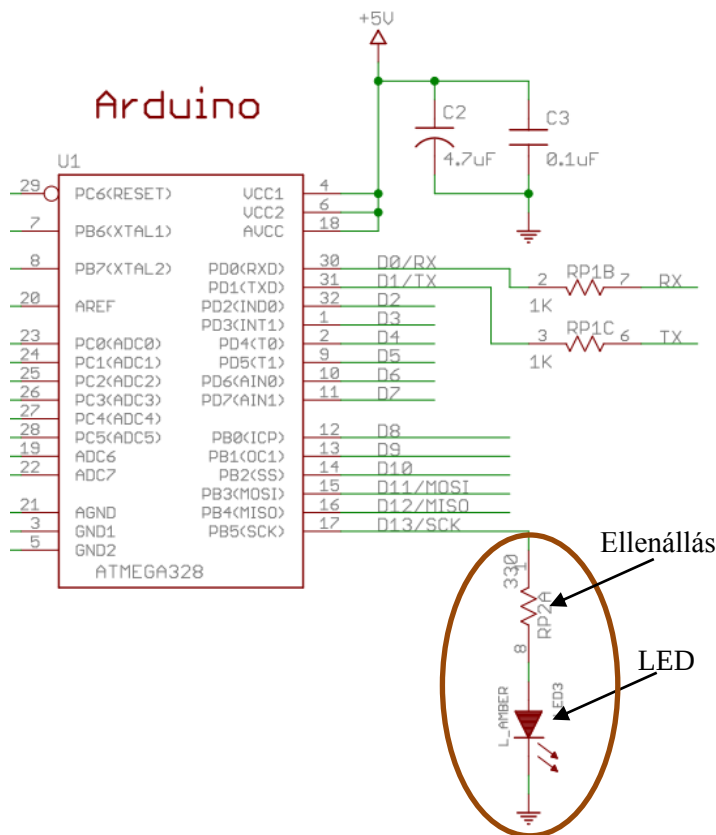
47. ábra: *Blink (villogó) mintaprogram*



48. ábra: *Blink mintaprogram sikeres feltöltése*

A LED villogtató mintaprogramban a *led* változóban kerül megadásra, hogy a fejlesztőlap 13-as digitális kimenetére (D13) van kötve a LED világító dióda. A *setup()* inicializáló függvényben a többfunkciós 13-as kivezetést a *pinMode(led, OUTPUT)* függvény segítségével digitális bemenetként kerül konfigurálásra. A *loop()* függvényben a *digitalWrite* parancs segítségével a kimenet magas (HIGH) majd *delay* függvény segítségével ms-ban megadott,

1000 ms nagyságú késleltetés letelte után alacsony (LOW) értékre kerül beállításra. Ennek következtében a 13-as digitális kimenet 1 mp időtartamra magas és 1 mp időtartamra alacsony. A 13-as kivezetésre az Arduino fejlesztőlapon közvetlenül bekötésre került egy piros színű LED és vele sorba kapcsolt, a LED áramának értékét meghatározó 330 Ω ellenállás az ábrán látható kapcsolási rajz részlet szerint.



49. ábra: *Arduino Nano fejlesztőlapon D13 kivezetésére kötött LED és áramkorlátozó ellenállás*

Magas szintű kimenet esetén a D13 kivezetésen közel 5 V feszültség jelenik meg, melynek következtében az ellenállás és LED soros kapcsolásán áram fog keresztül folyni (közel 10 mA nagyságú), ennek következtében a LED világít. Ha a kimenet alacsony szintű, úgy a D13 kivezetésen közel 0 V feszültség jelenik meg, a LED árama 0-ra csökken, a LED nem világít.

A mintaprogramban a *delay* függvény paraméterének módosításával a villogás lassítható vagy gyorsítható.

Komplex Arduino mintaprogram digitális be- és kimenetek és soros kommunikáció kezelésére

A következőkben egy összetett program kerül bemutatásra, mely egy távfelügyeleti számítógéptől soros kapcsolaton kapott adatok alapján be- vagy kikapcsolja az Arduino alapú beágyazott vezérlő D13 digitális kimenetre kötött LED-et és a vezérlő pedig a D12 digitális bemenet állapotának megfelelő adatot visszaküldi a felügyeleti számítógép felé.

A digitális kimenet beállítását a felügyeleti számítógép a „#DO_ON_” parancs segítségével tudja a D13 digitális kimenetet magas szintre, a „#DO_OFF” parancs segítségével pedig alacsony szintre beállítani. Mindkét parancs a „#” karakterrel kezdődik és utána fix 6 karaktert tartalmaz, ez egyszerűsíti a kapott parancs feldolgozását, értelmezését.

A D12 digitális bemenet állapotától függően az Arduino beágyazott vezérlő a „#DI_ON_” állapotjelzőt küldi a felügyeleti számítógép felé, ha a bemenet állapota magas szintű, és a „#DI_OFF” állapotjelzőt, ha alacsony szintű. Az állapotjelző szintén „#” karakterrel kezdődik és fix 6 karaktert tartalmaz.

A programban négy globális változó került definiálásra. A *digital_in* a D12 digitális bemenetről beolvasott, a *digital_out* a felügyeleti számítógép által küldött parancs alapján a D13 digitális kimenetre beállítandó logikai értékeket tartalmazza. A *serial_string* karaktertömbbe kerülnek beolvasásra a soros kapcsolaton érkező karakterek, a *command_received* logikai magas értéke jelzi, hogy a soros kapcsolaton a felügyeleti számítógép felől parancs érkezett.

Az inicializálást végző *setup()* függvényben a soros port megnyitását és a D12 kivezetés digitális bemenetként, a D13 kivezetés digitális kimenetként történő beállítását tartalmazza a korábbiakban már tárgyalt függvények felhasználásával.

```
//digitális be- és kimenet változója
boolean digital_in=LOW, digital_out=LOW;
char serial_string[8]; //soros porton érkező parancs karakterek tömbje
boolean command_received; //parancs érkezett

void setup() //inicializálás
{
  Serial.begin(9600); //soros port megnyitása, 9600 bps sebességgel
  pinMode(12, INPUT); //12 kivezetés digitális bemenet
  pinMode(13, OUTPUT); //13 kivezetés digitális kimenet
}
```

A *loop()* ciklikusan futó függvényben a vezérlők általános működésének lépései kerültek megvalósításra. Első lépésben a bemenet beolvasása történik, mely jelen esetben a 12-es, és az érték a *digital_in* változóba kerül. Aztán az egyes műveletvégzések következnek, mely jelen feladat egyszerűsége miatt

nem tartalmaz utasításokat. Harmadik lépésben a kimenetek beállítása történik, jelen esetben a 13-as kimenet a *digital_out* változó értékének függvényében. Negyedik lépésben a folyamatfelügyeleti számítógéppel történő kommunikáció kerül megvalósításra. A sorosan érkező adatok olvasását a *read_serial()* függvény végzi, mely visszatérési értéként a *command_received* változóban megadja, hogy érkezett-e parancs a soros porton a felügyeleti számítógép felől. A *command_received* változó az *if* feltételes utasításban megvizsgálásra kerül, és ha értéke magas (igaz), akkor további *if* feltételes utasításokban lévő karaktertömböket összehasonlító *strcmp()* függvénnyel megállapításra kerül, hogy kimenet értékét magas vagy alacsony szintre beállító parancs érkezett-e és ennek megfelelően állítja be a *digital_out* változó értékét. A kommunikációs blokk utolsó lépésében a *send_serial()* függvény hívódik meg, amely a felügyeleti számítógép felé küld el információkat, jelen esetben a 12-es bemenet logikai állapotát.

A ciklus következő lefutása előtt 100 ms nagyságú várakozás van beiktatva, ezzel a *loop()* függvény másodpercenként kb. 10-szer fut le.

```
void loop() //ciklikusan futó program
{
  //1. bemenetek beolvasása a bemeneti változókba
  digital_in = digitalRead(12); //12-es bemenet értékeinek beolvasása

  //2. bemenetek és belső változók alapján
  //felhasználói program futtatása (logikai, aritmetikai műveletek)
  //kimeneti változók beállítása

  //3. kimenetek beállítása
  digitalWrite(13,digital_out); //13-as kimenet beállítása

  //4. kommunikáció más vezérlőegységekkel, folyamatfelügyeleti
  //számítógéppel
  command_received = read_serial(); //soros port olvasása
  if (command_received) //ha érkezett parancs
  {
    //digitális kimenet bekapcsolása (HIGH) DO_ON_ paranccsal
    if (strcmp(serial_string,"#DO_ON_") == 0) digital_out = HIGH;
    //digitális kimenet kikapcsolása (LOW) DO_OFF paranccsal
    if (strcmp(serial_string,"#DO_OFF") == 0) digital_out = LOW;
  }
  send_serial(); //soros porton információk küldése

  delay(100); //rovid várakozás a ciklus újraindítása előtt
}
```

A *read_serial()* függvényben a *Serial.read()* függvény segítségével karakterenként kerül beolvasásra a soros porton érkező adat. Ahhoz, hogy értelmezhető parancs érkezzen, az első karakternek „#”-nek kell lennie, amely a *serial_string* karaktertömb első elemébe kerül beolvasásra. Egy *if* utasítás segítségével kerül megvizsgálásra, hogy ez „#”-e, és ha igen, akkor *command_rec* változó igaz értéket kap, amellyel majd a függvény vissza fog

térni. A küldött parancs további 6 karaktere egy *for* ciklus vezérlő utasítással kerül beolvasásra a *serial_string* karaktertömbbe. A karaktertömbbe beolvasott karaktersorozat a „\0” karakterrel kerül lezárásra, hogy a karaktertömb sztringként kezelhető legyen. Amennyiben az első beolvasott karakter nem „#” úgy a soros kapcsolaton eddig beolvasott karakterek mind kiolvasásra kerülnek, hogy a következő adat olvasás újonnan érkező karakterekkel kezdődjön. Az adatok kiolvasása egy *while* feltételes ciklusvezérlő utasítás segítségével. a *Searial.read()* függvény -1 visszatérési értékéig tart, mely azt jelzi, hogy már nincs kiolvasható adat. A függvény a végén a *command_rec* változó értékével tér vissza, mely kezdeti értéke hamis, ha parancs érkezik, akkor kerül igaz értékre beállításra.

```
boolean read_serial() //soros porton erkezo adat olvasasa
{
    boolean command_rec = false;

    serial_string[0] = Serial.read(); //egy karakter olvasasa

    //amennyiben erkezett parancs kezdete karakter,
    //akkor a parancs 6 karakterenek beolvasasa
    if (serial_string[0] == '#')
    {
        command_rec = true; //parancs erkezett valtozo igaz erteku
        for (int i=1; i<7; i++) //parancs tolabbi karakterei
            serial_string[i] = Serial.read(); //karakterek olvasasa
        serial_string[7] = '\0'; //sztring lezaro karakter
    }
    else
        while(Serial.read() != -1); //ha nem parancs jott, akkor
        //az erkezett adatok torlese

    //visszateres azzal, hogy erkezett-e parancs
    return command_rec;
}
```

A *send_serial()* függvény a *digital_in* logikai értékétől függően egy *if* feltételes utasítás segítségével kerül kiértékelésre, és a magas vagy alacsony értékétől függő állapot kerül elküldésre a folyamatfelügyeleti számítógép felé.

```
void send_serial() //soros porton informaciok kuldesa
{
    //digitalis bemenet magas erteke eseten DI_ON_ allapot kuldesa
    if (digital_in) Serial.print("#DI_ON_");
    //digitalis bemenet alacsony erteke eseten DI_OFF allapot kuldesa
    else Serial.print("#DI_OFF");
}
```

A teljes program forráskódja:

```

boolean digital_in=LOW, digital_out=LOW; //digitalis be- es kimenet változoja
char serial_string[8]; //soros porton érkezo parancs karakterek tombje
boolean command_received; //parancs érkezett
String s;

void setup() //inicializálás
{
  Serial.begin(9600); //soros port megnyitása, 9600 bps sebesség
  pinMode(12, INPUT); //12 kivezetés digitális bemenet
  pinMode(13, OUTPUT); //13 kivezetés digitális kimenet
}

void loop() //ciklikusan futó program
{
  //1. bemenetek beolvasása a bemeneti változókba
  digital_in = digitalRead(12); //12-es bemenet értékének beolvasása

  //2. bemenetek és belső változók alapján
  //felhasználói program futtatása (logikai, aritmetikai műveletek)
  //kimeneti változók beállítása

  //3. kimenetek beállítása
  digitalWrite(13,digital_out); //13-as kimenet beállítása

  //4. kommunikáció más vezérleegysegekkel, folyamatfelügyeleti szamitogeppel
  command_received = read_serial(); //soros port olvasása
  if (command_received) //ha érkezett parancs
  {
    //digitális kimenet bekapcsolása (HIGH) DO_ON_ parancssal
    if (strcmp(serial_string,"#DO_ON_") == 0) digital_out = HIGH;
    //digitális kimenet kikapcsolása (LOW) DO_OFF parancssal
    if (strcmp(serial_string,"#DO_OFF") == 0) digital_out = LOW;
  }
  send_serial(); //soros porton információk küldése

  delay(100); //rovid várakozás a ciklus újratekzdése előtt
}

boolean read_serial() //soros porton érkezo adat olvasása
{
  boolean command_rec = false;

  serial_string[0] = Serial.read(); //egy bajt/karakter olvasása

  //amennyiben érkezett parancs kezdete karakter,
  //akkor a parancs 6 karakterének beolvasása
  if (serial_string[0] == '#')
  {
    command_rec = true; //parancs érkezett változó igaz értéku
    for (int i=1; i<7; i++) //parancs további karakterei
      serial_string[i] = Serial.read(); //karakterek olvasása
    serial_string[7] = '\0'; //sztring lezáró karakter
  }
  else
    while(Serial.read() != -1); //ha nem parancs jött, akkor
    //az érkezett adatok törlese

  //visszatérés azzal, hogy érkezett-e parancs
  return command_rec;
}

void send_serial() //soros porton információk küldése
{
  //digitális bemenet magas értéke esetén DI_ON információ küldése
  if (digital_in) Serial.print("#DI_ON_");
  //digitális bemenet alacsony értéke esetén DI_OFF információ küldése
  else Serial.print("#DI_OFF");
}

```

Komplex Java mintaprogram grafikus felület és soros kommunikáció kezelésére

Ebben a fejezetben egy grafikus felhasználói felületű, Java alkalmazás kerül ismertetésre. Az alkalmazás egy Arduino alapú beágyazott vezérlővel soros kapcsolaton keresztül kommunikál, és a vezérlő D13 digitális kimenetét egy grafikus vezérlő elem állapotától függően alacsony vagy magas szintre beállítja, és a D12 digitális bemenet állapotát pedig a grafikus felületen visszajelzi. Ez az előző fejezetben ismertetett, a soros kapcsolaton küldött parancsok segítségével valósul meg.

A grafikus felületen egy csúszka állapotának megfelelően kerül a vezérlő digitális kimenete beállításra, valamint egy címkén megjelenő felirat a vezérlő által, a digitális bemenet állapotáról küldött információ alapján, kerül kiírásra. Az alkalmazásban a vezérlő digitális kimenetének beállítása a csúszka állapotváltozásának eseményére meghívódó, a soros kommunikációs portra a csúszka állapotától függően küldött „#DO_ON_” vagy „#DO_OFF” parancs segítségével valósul meg. A soros porton a vezérlőtől érkező „#DI_ON_” vagy „#DI_OFF” állapotjelzőtől függően kerül beállításra a címke felirata.

A grafikus felületű alkalmazás létrehozása, és az eseménykezelés megvalósítása a Java programozás alapjai, valamint a soros port kezelése pedig Soros kommunikáció fejezetben már ismertetve lettek. Az ott leírtaknak megfelelően létre kell hozni egy új Java projektet, amelynek a neve legyen LEDSwitch, benne egy JFrame konténert, melynek neve legyen JFrameOfLEDSwitch.

A program indulásakor a *JFrameOfLEDSwitch* példánynak létrejöttkor hívódik meg a *SerialPortInitialization()* függvény is, mely a soros port megnyitását és paraméterezését tartalmazza a soros kommunikáció kezelése fejezetben leírtaknak megfelelően.

```
public JFrameOfLEDSwitch() {  
    initComponents();  
    SerialPortInitialization();  
}
```

```

//soros port megnyitása, beállítása
private void SerialPortInitialization() {
    try {
        serialPort.openPort(); //soros port megnyitása
        //soros port paramétereinek beállítása
        serialPort.setParams(SerialPort.BAUDRATE_9600,
            SerialPort.DATABITS_8,
            SerialPort.STOPBITS_1,
            SerialPort.PARITY_NONE);

        //soros porton érkező adat esetén létrejövő esemény
        //a SerialDataProcess osztály objektumpéldányát hívja meg
        serialPort.addEventListener(new SerialDataProcess(),
            SerialPort.MASK_RXCHAR);
    }
    catch (SerialPortException ex) { //hibakezelés
        System.out.println("Hiba:" + ex);
    }
}

```

A GUI szerkesztőben elhelyezett csúszka és címke paraméterezését a korábbi példában már bemutatott módon kell elvégezni. A csúszka *stateChanged* állapotváltozásakor létrejövő esemény hatására a *jSlider1StateChanged()* eseménykezelő függvény hívódik meg, melyben kell megvalósítani az állapottól függő parancs küldését. A csúszka állapota a *getValue()* metódus segítségével történik, és ez alapján kerül a soros porton elküldésre a *writeString()* metódussal vagy a „#DO_ON_” vagy a „#DO_OFF” parancs.

```

//csúszka állapotváltozás eseményt kezelő függvény
private void jSlider1StateChanged(javax.swing.event.ChangeEvent evt) {
    try {
        if(this.jSlider1.getValue()==1) {
            serialPort.writeString("#DO_ON_");
        }
        if(this.jSlider1.getValue()==0) {
            serialPort.writeString("#DO_OFF");
        }
    }
    catch (SerialPortException ex) { //hibakezelés
        System.out.println("Hiba:" + ex);
    }
}

```

A soros porton érkező adat feldolgozása, a Soros kommunikáció fejezetben ismertetett, adat érkezett esemény hatására a *SerialDataProcess* osztályban történik. A címke felirata a *setText()* és a felirat színe a *setForeground()* metódusokkal kerül beállításra a soros porton beolvasott állapotjelző „#DI_ON_” vagy „#DI_OFF” értékétől függően.

```

//soros port adatkezelését eseményvezérelten megvalósító osztály
class SerialDataProcess implements SerialPortEventListener {
    //eseménykezelő függvény
    public void serialEvent(SerialPortEvent event) {
        try {
            //beolvasott adat
            String readed = new
                String(serialPort.readBytes(event.getEventValue()));

            //ha "#DI_ON_" adat érkezett, akkor a címke felirata
            //"ON" értékre kerül beállításra és piros színre
            if("#DI_ON_".equals(readed)) {
                jLabel1.setForeground(Color.RED);
                jLabel1.setText("ON");
            }

            //ha "#DI_OFF" adat érkezett, akkor a címke felirata
            //"ONFF" értékre kerül beállításra és fekete színre
            if("#DI_OFF".equals(readed)) {
                jLabel1.setForeground(Color.BLACK);
                jLabel1.setText("OFF");
            }
        }

        catch (SerialPortException ex) { //hibakezelés
            System.out.println("Hiba:" + ex);
        }
    }
}

```

A teljes program forráskódja (fejlesztőkörnyezet automatikusan generált kódrészlete nélkül):

```

package ledswitch;

import java.awt.Color;
import jssc.SerialPort;
import jssc.SerialPortEvent;
import jssc.SerialPortEventListener;
import jssc.SerialPortException;

public class JFrameOfLEDSwitch extends javax.swing.JFrame {

    //soros port kezelő osztály objektumpéldánya
    static SerialPort serialPort = new SerialPort("/dev/ttyUSB0");

    public JFrameOfLEDSwitch() {
        initComponents();
        SerialPortInitialization();
    }

    //soros port inicializálása
    private void SerialPortInitialization() {
        try {
            serialPort.openPort(); //soros port megnyitása
            serialPort.setParams(SerialPort.BAUDRATE_9600, //soros port
                                //paramétereinek beállítása
                                SerialPort.DATABITS_8,
                                SerialPort.STOPBITS_1,
                                SerialPort.PARITY_NONE);
            //soros porton érkező adat esetén létrejövő esemény
            //a SerialDataProcess osztály objektumpéldányát hívja meg
            serialPort.addEventListener(new SerialDataProcess(),
                                        SerialPort.MASK_RXCHAR);
        }
        catch (SerialPortException ex) { //hibakezelés
            System.out.println("Hiba:" + ex);
        }
    }

    //soros port adatkezelését eseményvezérelten megvalósító osztály
    class SerialDataProcess implements SerialPortEventListener {
        public void serialEvent(SerialPortEvent event) {
            //eseménykezelő függvény
            try {
                String readed = new
                    String(serialPort.readBytes(event.getEventValue()));

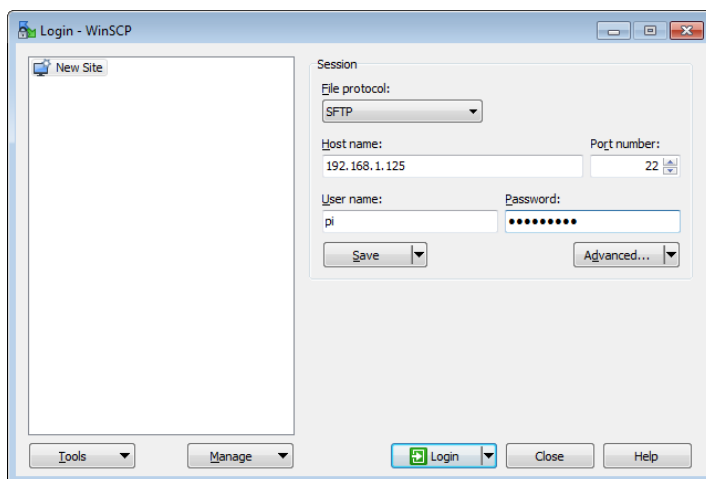
                if("#DI_ON_".equals(readed)) {
                    jLabel1.setForeground(Color.RED);
                    jLabel1.setText("ON");
                }
                if("#DI_OFF".equals(readed)) {
                    jLabel1.setForeground(Color.BLACK);
                    jLabel1.setText("OFF");
                }
            }
            catch (SerialPortException ex) { //hibakezelés
                System.out.println("Hiba:" + ex);
            }
        }
    }
}

```

```
//csúszka állapotváltozásának eseményére meghívódó függvény
private void jSlider1StateChanged(javax.swing.event.ChangeEvent
    evt) {
    try {
        if(this.jSlider1.getValue()==1) {
            serialPort.writeString("#DO_ON_");
        }
        if(this.jSlider1.getValue()==0) {
            serialPort.writeString("#DO_OFF");
        }
    }
    catch (SerialPortException ex) { //hibakezelés
        System.out.println("Hiba:" + ex);
    }
}
```

Java alkalmazás futtatása Raspberry Pi 2-ön

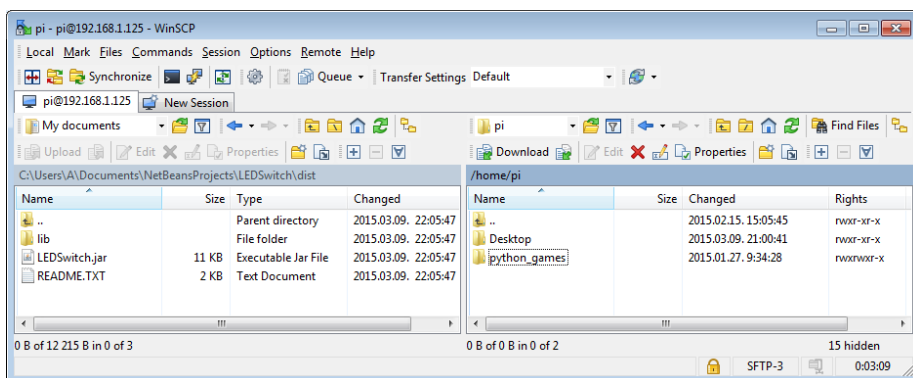
Az előzőekben bemutatott Java alkalmazás fejlesztése és futtatása Windows operációs rendszer alatt történt. A Java alkalmazás Linux operációs rendszeren is futtatható, de ott figyelembe kell venni, hogy a Windows operációs rendszer alatt a portra „COMX”, ahol X egy szám, néven kell hivatkozni. Azonban Linux operációs rendszeren „/dev/ttyYYYX” névvel, jelen esetben ez „/dev/ttyUSB0”. A Java alkalmazás Raspberry Pi 2-ön, Linux operációs rendszer alatt történő futtatásához a port nevének módosítása és fordítás után a NetBeans Dokumentumok/NetBeansProjects/LEDSwitch nevű projekt könyvtárában, a dist alkönyvtárban található fájlokat csak át kell másolni a Raspberry Pi 2-re és ott futtatni. Jelen esetben a LEDSwitch.jar és a lib könyvtár tartalmát (jssc-2.8.0.jar) kell átmásolni a Raspberry Pi 2-re. A fájlok másolásához a WinSCP program használható, mely a www.winscp.net oldalról tölthető le. A program indítása után a Host name (hoszt név) mezőbe a Raspberry Pi 2 IP címét kell megadni, User name (felhasználónév) pi, a Password (jelszó) raspberry. Login gombra kattintva megtörténik a bejelentkezés.



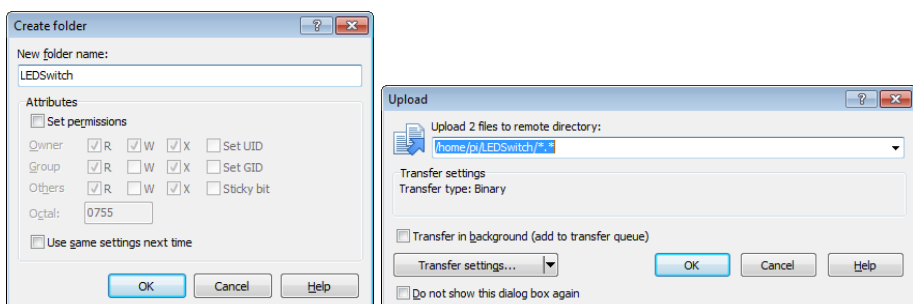
50. ábra: WinSCP programmal bejelentkezés a Raspberry Pi 2-re

A bejelentkezés után a pi nevű felhasználó mappája jelenik meg a kettéosztott képernyő jobb oldalán, a bal oldalon pedig annak a gép, amelyről a bejelentkezés történt, jelen esetben a Windows számítógép. A szükséges fájlok a korábban megadott Dokumentumok\NetBeansProjects\LEDSwitch projekt könyvtárban találhatóak, mely a C:\Users\aktuaális_felhasználó útvonalon érhető el. A Raspberry Pi 2-ön hozzunk létre egy LEDSwitch könyvtárat (Create

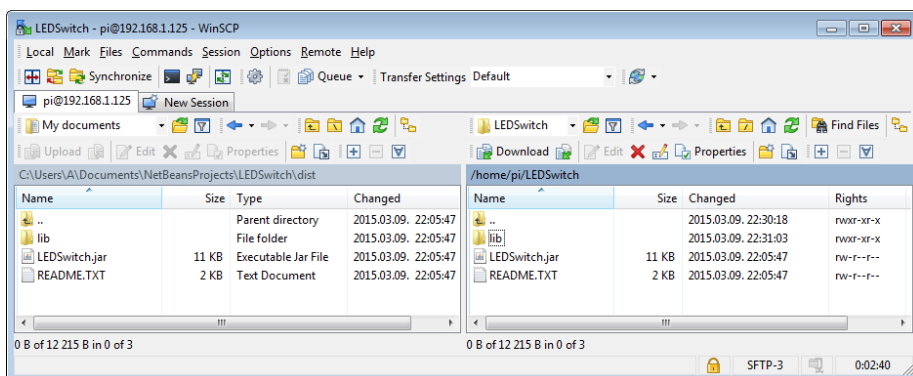
folder) az F7 gyorsbillentyű segítségével, majd a dist könyvtár tartalmát másoljuk át (Upload) az F5 gyorsbillentyűvel.



51. ábra: LEDSwitch projekt könyvtár dist alkönyvtára és a pi felhasználó könyvtára

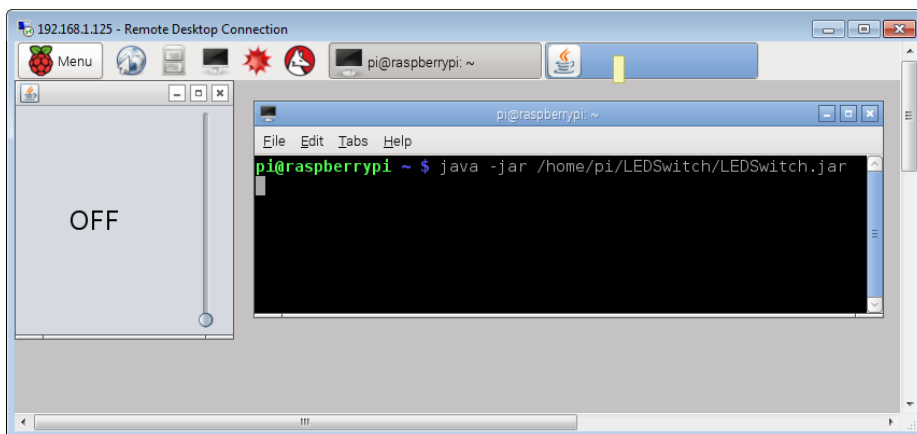


52. ábra: Új könyvtár létrehozása és a dist könyvtár tartalmának feltöltése a Raspberry Pi 2-re



53. ábra: Raspberry Pi 2 LEDSwitch könyvtárába felmásolt fájlok

A Java alkalmazás futtatásához távoli asztal segítségével be kell jelentkezni a Raspberry Pi 2-re, majd egy LXTerminal parancssoros alkalmazásban az alábbi parancsot kiadni: `java -jar /home/pi/LEDSwitch/LEDSwitch.jar`. Amennyiben az Arduino Nano USB porton csatlakozik a Raspberry Pi 2-höz, úgy az alkalmazás működni is fog, mivel az eszközkezelő alapértelmezetten telepítve volt.



54. ábra: *LEDSwitch Java alkalmazás futtatása Raspberry Pi 2-ön*

Felhasznált irodalom

- Arduino projekt weboldala: www.arduino.cc (2015.02.01)
- Raspberry Pi projekt weboldala: www.raspberrypi.org (2015.02.01)