



Napfogyatkozás szabadon

Eclipse

© Kiskapu Kft. Minden jog fenntartva

Az Eclipse az egyik, ha nem „a” legnépszerűbb nyílt forrású, platformfüggetlen integrált fejlesztőkörnyezet, amely a több száz elérhető bővítmény segítségével szinte az összes ma használatos fejlesztési technológiát támogatja. A rendszer már a 3.1-es változatszámnál tart, legfőbb ideje tehát, hogy a Linuxvilág hasábjain is bemutatkozzék.

A projekt célja igazából nem pusztán egy integrált fejlesztőkörnyezet (*IDE – Integrated Development Environment*) létrehozása volt, hanem egy olyan keretrendszer megalkotása, amely a hozzá kapcsolódó bővítmények révén képes bármilyen programozó igény kielégítésére. S bár az *Eclipse* hivatalosan csak „keretrendszer”, a gyakorlatban egy teljes értékű környezetről van szó, amely képes bármilyen technológiához idomulni. Sok esetben nem csak a kódolást segíti, de például az *UML/UML2*

modellező nyelv használatával a programtervezés folyamatát is támogatja. De ne szaladjunk ennyire előre, előbb ismerkedjünk meg a „csupasz” *Eclipse* legfontosabb jellemzőivel: a telepítés folyamatával, a használati alapfilozófiával és a nagyszerű lehetőségekkel. Rendszerünk, ha nem is teljesen platformfüggetlen, mindenképp többplatformos. Letöltéskor kiválaszthatjuk, hogy milyen operációs rendszeren, milyen architektúra mellett szeretnénk használni. Létezik windowsos változat, linuxos állomány – ez utóbbin belül az *x86-os*, az *x86-64bit-es* és a *PowerPC* architektúra támogatott. Ezekon kívül elérhető még *Solaris 8-ra*, *AIX-re*, *Mac OS X-re*, és akkor még nem beszéltem az egyes rendszereken belül a különböző grafikus felületekről, amit a rendszer támogat. Alapjaiban mindezt úgy éri el az *Eclipse*, hogy a szoftver lelke java alapú. Nem nehéz kitalálni, hogy a futtatásához minimum *Java* futtatókörnyezetre (*J2RE*) van szükségünk, de az alapértelmezett használathoz inkább a *Java* fejlesztőkörnyezet (*J2SDK*) megléte javasolt. A „csupasz” fejlesztőkörnyezet ugyanis a *Java* nyelven történő programozást támogatja. Ha mi más nyelvet szeretnénk használni, ahhoz

bővítményre lesz szükség. A használat és a felépítés természetesen a bővítmények esetében is ugyanaz, ezért először az alapkörnyezetet fogjuk megismerni.

Telepítés

A telepítendő változat az *Eclipse 3.1.1-Linux-x86-GTK2* lesz. Ehhez szükségünk lesz *GTK2* környezetre, amely ma már az összes terjesztésnek alapértelmezetten része, mi több, általában telepítve is van, tehát ezzel a résszel nincs gondunk. Amire jobban oda kell figyelni, az a *Java* fejlesztőkörnyezet. Ez általában egyetlen *Linux* terjesztésnek sem része, ezt a *Sun* weboldaláról kell letölteni. Ha nekünk már van *Java* fejlesztőkörnyezetünk, a következő lépést kihagyhatjuk.

Java környezet

Az *Eclipse* a *Java2-t*, vagyis az *1.4-es* sorozatot javasolja, de tapasztalatom szerint a *Java5-tel* (*1.5-ös sorozat*) is kiválóan működik. Mindenesetre mi most maradjunk az első lehetőségnél. Látogassunk el a <http://java.sun.com/j2se/1.4.2/download.html> címre, és itt válasszuk a *J2SE SDK* letöltését, abból is a linuxos, önkicsomagoló változatot. Ha ez megvolna, indítsuk el az állományt. A licenfeltételek elfogadása

után a *Java* fejlesztőkörnyezet kicsomagolódik egy *jdk1.4.2_09* nevű könyvtárba. Ennek tartalmát helyezük át a */usr/lib/java* könyvtárba, majd a */usr/lib/java/bin/java* illetve */usr/lib/java/bin/javac* fájlokról készítünk szimbolikus linket a */usr/bin* könyvtárban. Ezzel a *Java* fordító illetve futtatókörnyezet telepítésével el is készültünk.

Eclipse

Az *Eclipse* telepítése ennél egyszerűbb. Mindenekelőtt töltsük le a <http://www.eclipse.org/downloads/index.php> címről az *Eclipse SDK 3.1.1*-es, linuxos, *GTK2*-t támogató változatát. (Az operációs rendszertől függően fel fogja ajánlani az oldal a nekünk megfelelőt.) Ha ezzel megvoldnánk, nincs is más dolgunk, mint kitömöríteni a *tar.gz* állományt egy általunk választott könyvtárba, és készen vagyunk. A fejlesztőkörnyezet a `<választott könyvtár>/eclipse/eclipse` paranccsal indul.

Az első lépések

A program az indulás után rögtön arra kérdez rá, hogy melyik könyvtár legyen a munkaterületünk. Alapértelmezett ezt a könyvtárat fogja használni mindenféle fájlművelet során. Nemcsak az indítás során tudjuk ezt megadni, de a későbbiekben is a *File* menü *Switch Workspace*



■ 1. ábra Az indítási folyamat során adhatjuk meg a használni kívánt munkakönyvtárat

menüpontjával is változtathatunk rajta. A munkaterületek használatával jobban elkülönülnek egymástól a különböző projektek.

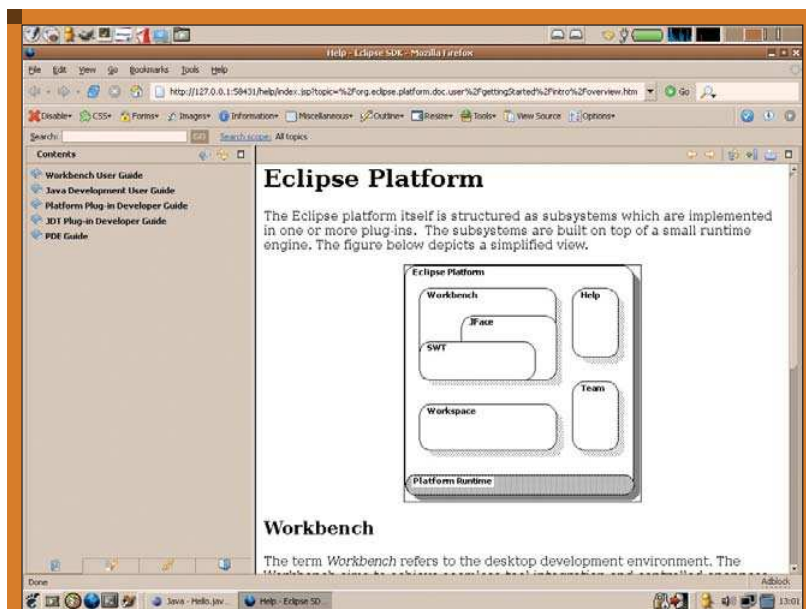
A program az első indítás során egy általános súgót jelenít meg. Innen érhetjük el többek között a teljes web alapú felhasználói dokumentációt, amely természetesen a helyi gépen található, tehát nincs szükség hozzá internetkapcsolatra. Ebben a bizonyos dokumentációban minden

szükséges információt megtalálunk: rendkívül részletes és egyértelmű. Javasolom, hogy mielőtt bármilyen komolyabb munkába kezdenénk, olvassuk el legalább az áttekintő részeket, hogy képet kapjunk a program egészéről.

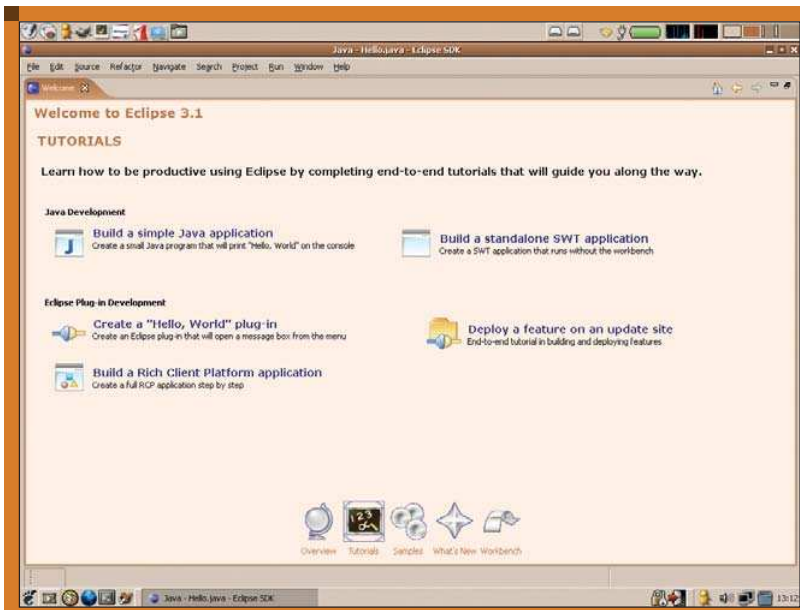
Az első indítás során az üdvözlőképernyőt láthatjuk. Itt rengeteg segítséget találunk az egyszerűbb folyamatok (például *Java* projekt létrehozása) bemutatásától kezdve egészen a már említett komplett dokumentációig. Érdeemes barangolni itt is egyet. (Ez a felület egyébként a *Help* menü *Welcome* menüpontjának segítségével bármikor előhozható.)

Az üdvözlőképernyő bezárása után kapjuk meg a fejlesztőkörnyezet igazi felületét. A felület több alrekeszre tagolódik, amelyek elrendezése természetesen változtatható – ez a kép-lékenység egyébként az egész felületre is igaz.

A különböző fejlesztői tevékenységek ellátásához természetesen különböző felületi elemekre van szükségünk. Például néhol szükségünk van a szövegszerkesztőre, néhol nincs, néhol pedig egyenesen két darab kell. Ennek a helyzetnek a megoldására a rendszer különböző perspektívákat tartalmaz. Minden perspektíván más-más felületi elemek kaptak helyet, így ha például nyomkövetést szeretnénk végezni, elég csak „néze-



■ 2. ábra A web alapú dokumentációból mindent megtudhatunk



■ 3. ábra Az első indítás során felkarol bennünket a segítőkész üdvözlőképernyő

tet váltani”, és máris minden szükséges dolog a helyére kerül. Ha végeztünk, csak vissza kell mennünk az előző nézetre.

Az alapértelmezett fejlesztői nézet minden programnyelv esetében nagyjából ugyanolyan: baloldalunk található a projekt böngésző, itt található a projekt minden eleme (lásd később) faszervezetben mutatva. Ez talán a környezet leggyakrabban használt eleme, egyfajta navigátor. Mellette található maga a szövegszerkesztő, amiről szintén később szöveg. A szövegszerkesztő mellett jobb oldalon van az *áttekintés (Outline)* ablak, amely az éppen megnyitott fájlban, mint programkódban található osztályokat, tagfüggvényeket, változókat tartalmazza. A szerkesztő és az áttekintő ablak alatt található egy sok füles ablak, ahol megtalálhatjuk a programkódban lévő hibák listáját, változómeghatározásokat, valamint a program kimenetét (standard output).

Mindenek lelke: a projektek

A fejlesztőkörnyezet projekt alapú. Ez azt jelenti, hogy bármi, amit csinálunk, fejlesztünk, egy projekt része, eleme kell legyen, s ezt nem lehet megkerülni. Nem tudunk például olyat csinálni, hogy van 2-3 *Java* vagy *PHP* fájlunk, és azokat szerkesztgetjük... muszáj létrehozni

egy projektet (egy afféle tartályt), amihez a fenti fájlokat hozzárendeljük. Ez természetesen néha hátrány: nem mindig van szükségünk ugyanis ilyen tartalmazó objektumra, amit a *projekt* testesít meg az *Eclipse*-ben.

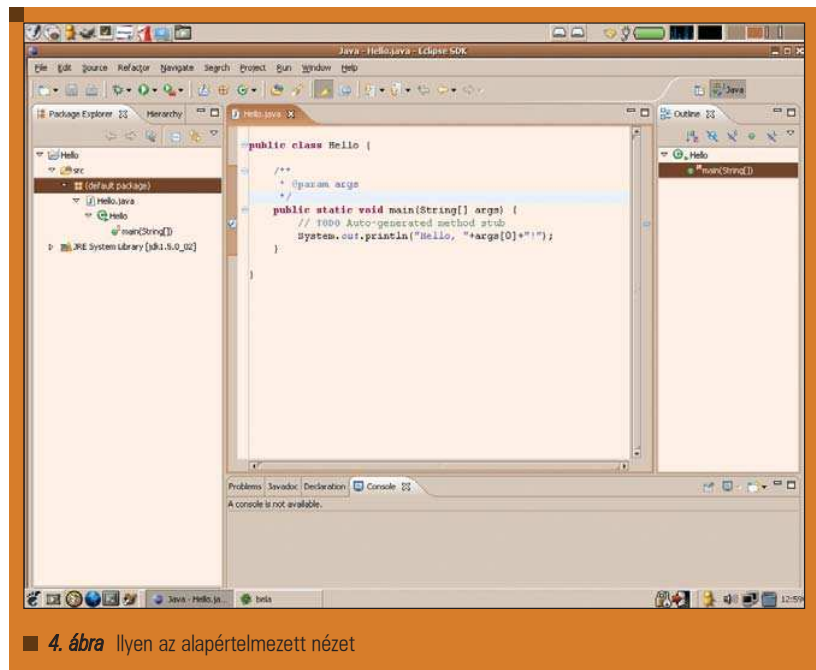
Habár az is igaz, hogy ilyen esetekben nincs szükségünk magára a fejlesztőkörnyezetre sem.

Az *Eclipse* célja, hogy segítségével össze tudjuk fogni a fejlesztést,

keretet adjon a munkának, letisztult képet mutasson a bonyolult fejlesztésekről, egy szóval az összetett, nagy projektet támogatása, ami viszont elképzelhetetlen lenne a fenti megszorítások nélkül. Ennek megfelelően persze a fejlesztőkörnyezet lesz kissé bonyolult, amint az a beállítási lehetőségek mennyiségéből is látszik.

Ha más rendszerekről térünk át, netán egy fejlesztőrendszert teljesen nélkülöző (csak forrásfájlokat tartalmazó) kódot szeretnénk *Eclipse* segítségével továbbvinni, ahhoz elég egy üres projekt, ahová később néhány kattintással importálhatjuk a „külső” kódot. Így biztosítja a rendszerünk, hogy a más felépítésű fejlesztőkörnyezetekről is zökkenőmentesen térhessünk át *Eclipse*-re. Később látni fogjuk, hogy nemcsak helyi fájlok, de távoli CVS-ben tárolt projektek is játszva fejleszthetők tovább a segítségével. Látszik, hogy az *Eclipse*-t fejlesztő közösség nagyon jól kitalálta a rendszernek ezt a részét: bármilyen távoli projekthez igen egyszerűen hozzáférhetünk.

Egy-egy ilyen projekt további sajátossága, hogy egyéni beállításokat adhatunk meg mindegyik számára. Ezek alapértelmezetten a globális beállításokból öröklődnek, de lehetőségünk van a különböző elvárásoknak megfelelően, úgy mond



■ 4. ábra Ilyen az alapértelmezett nézet

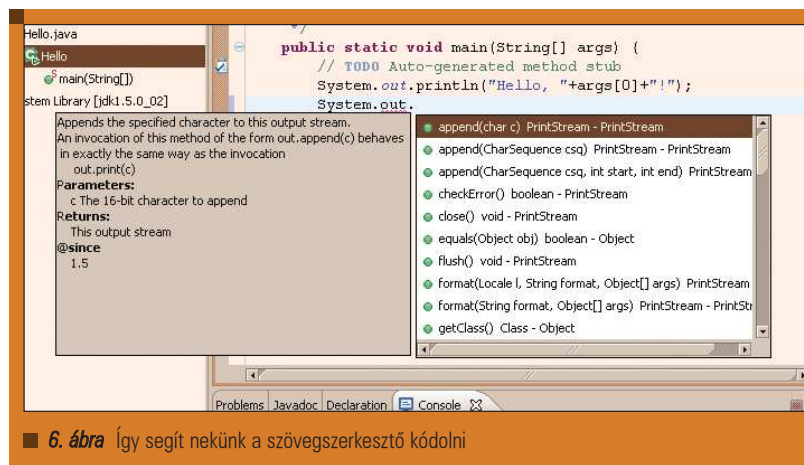
testreszabni egy projekt esetére a fejlesztőkörnyezet beállításait, amely változtatások azonban nem terjednek ki a többire.

A kódolás alapeleme: a szövegszerkesztő

Ha programfejlesztés, akkor kódolás, ha kódolás, akkor valamilyen szövegszerkesztő. Az *Eclipse*-ben található szövegszerkesztő méltó feladatára: elegendően sokat tud, hogy valóban alapeleme legyen a fejlesztőkörnyezetnek, de semmivel sem bonyolultabb annál, mint ami feltétlenül szükséges. Ez utóbbi akkor nagy előny, ha valaki találkozott már olyan szerkesztőkkel, mint például a *JEdit*, amely vitathatatlanul nagyszerű, de egyrészt bonyolult és nehezen átlátható, másrészt az kódoló a számtalan nyújtott szolgáltatásnak mindössze töredékét használja, tehát bátran mondhatjuk, hogy felesleges.

A szerkesztő legfontosabb ismérvei:

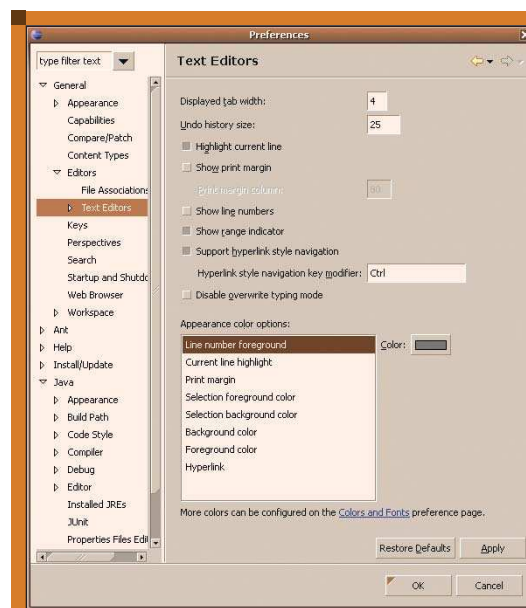
- Támogatja a programkódon belüli blokkok elrejtését. Ilyen blokk elrejtés lehet egy már megírt, jól működő tagfüggvény „bezárása” oly módon, hogy a szerkesztőben csak a függvénydefiníció látszik, a függvény tartalma helyén három pontocskára jelenik meg egy négyzetben, amire kattintva



■ 6. ábra Így segít nekünk a szövegszerkesztő kódolni

újra „lenyílik” az adott függvény kódja. Hasonló a helyzet a hozzáfűzött komment alapú dokumentációval, amelyet szintén egy sorra csukhatunk össze, amennyiben szükséges.

- Ismeri az adott programnyelvre jellemző szintaktikát, a szintaktikai kiemelés végez, amelyet szabadon testreszabhatunk, és nem utolsó szempont, hogy ha hibázunk, figyelmeztet. Nem csak a gépelési hibát, pontosvessző hiányát, vagy zárójel hibákat vesz észre, de például Java nyelv esetén a nyelvi sajátosságoknak megfelelő problémákat is megjeleníti. Ilyen például, ha egy statikus változó dinamikus környezetben szeretnénk használni, vagy ha nem létező, nem megfelelő típusú változót, objektumot akarunk használni, ha mellőzzük a szükséges *try-catch* blokkokat, elavult (*deprecated*) függvényeket próbálunk használni stb.
- Ismeri a programban használt gyári és saját fejlesztésű osztályokat, így a gépelés közben a gépelt szöveg alatt lenyíló menüben felajánlja a választható tagfüggvényeket,



■ 5. ábra Az Eclipse környezet átfogó beállításait egy helyről módosíthatjuk

ven áll rendelkezésre, de a megfelelő bővítmények megléte esetén *C/C++* vagy akár *PHP* nyelven is.

- Tartalmaz kódolás segítő (*code-assist*). Ez leggyakrabban abban nyilvánul meg, hogy bezárja helyettünk a zárójelet, idézőjelet, sortörés esetén az épp szerkesztett karaktersorozat (*string*) megfelelő szintaktika szerint törli meg. Ide tartozik még a fejlesztő által megadott sablonok kezelése. Ezek többnyire *XML/HTML* címkék lehetnek, amelyek így elég elkezdni beírni a nevét, a szerkesztő felajánlja a teljes kódrészletet, és segít az egyes jellemzők (*attributes*) kitöltésében.

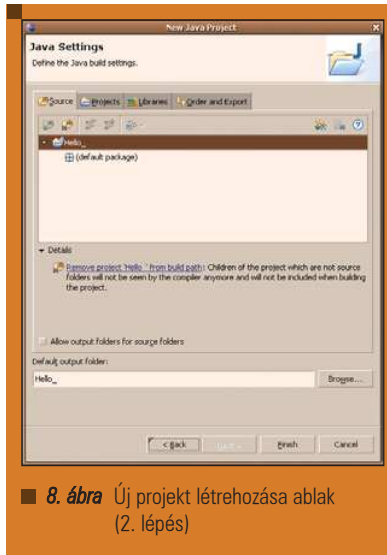
Csapatmunka

Ma már a legtöbb fejlesztés elképzelhetetlen csapatmunka és változatkezelés nélkül. Ennek fényében az *Eclipse*-nek szerves része a *CVS (Concurrent Versioning System)* változatkezelő. Nem csak azt teszi lehetővé, hogy kezeljük a változatokat, változtatásokat, de segítségével egyszerre több ember is dolgozhat ugyanazon a projekten, ugyanazonokon a fájlokon. Nagy előnye a rendszernek, hogy nincs szükség külön szoftver telepítésére, a szolgáltatás az *Eclipse*-be van építve. Ha létrehozunk egy projektet, aztán

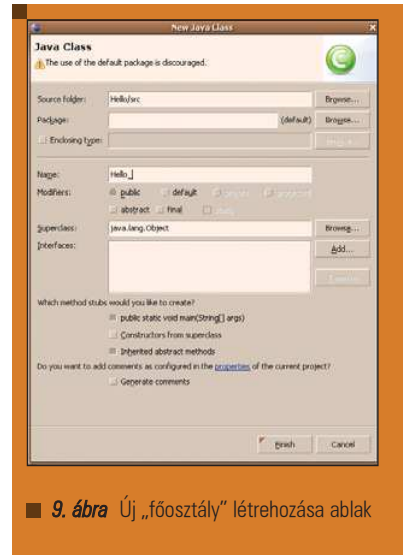
© Kiskapu Kft. Minden jog fenntartva



■ 7. ábra Új projekt létrehozása ablak (1. lépés)



■ 8. ábra Új projekt létrehozása ablak (2. lépés)



■ 9. ábra Új „főosztály” létrehozása ablak

azt „megosztjuk” úgy, hogy egy CVS táriba helyezzük, akkor minden egyes projekt-elem (fájl, erőforrás, stb) mellett megjelenik a fájl állapota, változatszám, stb. A fájl helyi menüjében egyszerűen kattintásokkal lehet összehangolni a helyi változatot a CVS tárral mindkét irányban – mindenféle nehézség nélkül. Annak érdekében, hogy ez a művelet sok fájl, sok változtatás esetén kényel-

mes legyen, rendelkezésünkre áll egy kifejezetten erre a célra készült nézet (CVS Repository Explorer). Itt láthatjuk a fájlokat, összehasonlíthatjuk a központi tárral, megnézhetjük, hogy a fájl mely sorai és hogyan változtak – mindezt természetesen grafikus formában, két egymás mellé helyezett, szinkronban mozgó szövegszerkesztővel, ahol színek, nyilak mutatják, hogy mi miről mire változott, akár vissza-

menőleg az összes „történeti” változtatásra. Ugyanitt lehetőség van a fizikai szinkronizációra is: a változtatások központi táriban történő rögzítésére. Véleményem szerint talán a rendszer egyik legnagyobb előnye, hogy ilyen jó CVS-kezelőt tartalmaz (külön szoftver esetében sem láttam még ilyen fejlett változatot).

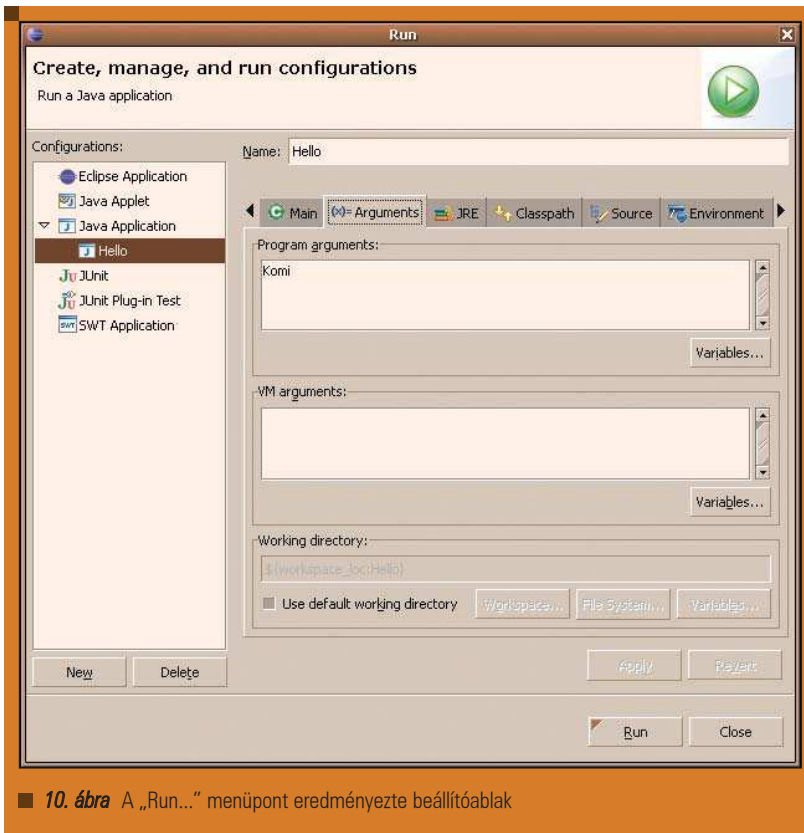
Egy példa

Egy alapvetően grafikus felületet természetesen igen nehéz írásban bemutatni. Ezt a nehézséget enyhítendő, megpróbálom egy egyszerű Java alapú projekt létrehozásával bemutatni a fent leírtakat: a rendszer alapvető felépítését.

A példa egy „Hello Világ!” program készítése, amit aztán később paraméterfüggő köszöntéssé alakítunk át. Indítsuk el az Eclipse-t.

A File menü New->Project menüpontjával kezdjük el létrehozni a projektünket. A megjelenő ablakban válasszuk a Java Project lehetőséget, majd a megjelenő ablakban adjuk a projektnek a „Hello” nevet. A többi beállítást hagyjuk változatlanul. Ismét egy ablak jelenik meg, amit szintén változatlanul hagyjunk jóvá.

Baloldalon, a böngészőben megjelenik a Hello projekt. Jobb gombbal kattintva a New->Class menüvel adjuk hozzá az egyetlen, fő osztályunkat. Adjuk neki a „Hello” nevet. Az alatta lévő csomagnévhez is nyugodtan írjuk ugyanezt. Ha nem tesszük, egy default nevű csomagba rakja az



■ 10. ábra A „Run...” menüpont eredményezte beállítóablak

osztályunkat. (A csomagok használatát a projektekéhez hasonlóan nem lehet megkerülni.) Ha ezzel megvoldánk, alul jelöljük be, hogy szeretnénk a `public static void main(String[] args)` nevű tagfüggvénycsontk létrehozását, majd egy bátor **Finish** gombnyomással nyugtázzuk az elvégzettekét. lépés: miután a szövegszerkesztő megnyitotta számunkra a létrehozott osztályt, elkészítette a belépési pontot, a `main` függvény vázát, csak annyi a dolgunk, hogy beírjuk az alábbi sort a függvény törzsébe:

```
System.out.println("Hello
    ✎ világ!");
```

Futtassuk le az elkészült programot. A projekt böngészőben válasszuk ki a `Hello.java` fájlt, majd jobb gombbal elérhető helyi menüben válasszuk a `Run as -> Java Application` menüpontot. Ennek hatására a program lefordul, majd lefut, és alul, a `Console` feliratú fül fogja tartalmazni a program kimenetét:

```
Hello világ!;
```

Érdekes dolog a futtatás panelje a jelenlegi `Hello` alkalmazásunknak. A feladat most az, hogy a köszöntést tegyük személyhez szólóvá: a programnak átadott paraméterben szereplő néven köszöntsön.

A szövegszerkesztőben a `Hello.java` `main` függvényének törzsében írjuk át az egyetlen sort az alábbira:

```
System.out.println("Hello,
    ✎ "+args[0]+"!");
```

A `Run` menü `Run...` menüpontját választva láthatjuk a környezethez tartozó futtatási beállításokat, szabályokat. A `Java Application` csoport alatt található a mi `Hello` nevű futtatási szabályunk, amely az előző feladat utolsó lépéseként jött létre. Kattintsunk rá, és a jobb oldalon megjelenő panel `Arguments` címkejű fülén a `Program arguments` mezőbe írjuk be a saját nevünket. Ez az `Apply`, majd a `Run` gombok megnyomása után láthatjuk az eredményt. A továbbiakban a következő módosításig ez a futtatási szabály lesz érvényben, ez az alapértelmezett, de természetesen tetszőleges számú futtatási szabályt alkalmazhatunk egy

adott projektre, ekkor a futtatáskor ki kell választani, hogy melyik szerint szeretnénk a programunkat lefuttatni.

Összefoglalás

A fent leírtak tényleg csak ízelítő jellegűek. Egy ilyen fejlesztőkörnyezet bemutatása egy teljes tankönyvnyi helyet emésztené fel – ez inkább csak kedvcsináló némi kezdő tanáccsal, hogy az esetleges kudarcok ne vegyék el kedvünket a továbbiaktól. Számátalan dolog kimaradt a cikkből, például a Java fejlesztés részletesebb ismertetése konkrétumokkal, a `Javadoc`, mint eszköz alkalmazása, a különböző bővítmények használata, és még sorolhatnám. Ezen hiányok pótlása cikkünk egy következő részében esedékes, és ahogy a showman mondaná: *Kérjük maradjanak továbbra is velünk!*



Komáromi Zoltán

(komi@kiskapu.hu)

25 éves, mérnök-informatikus. Rendszermérőként dolgozik a Kiskapu Kft.-nél, szabadidejében szívesen fotózik.

