

Java és Eclipse útmutató

Készítette: Simon Balázs, BME IIT, 2019.

Tartalomjegyzék

1	Bevezetés	2
2	JRE és JDK	2
3	Java és Javac	2
4	Környezeti változók	3
4.1	Környezeti változók beállítása Windows operációs rendszeren	3
4.2	Környezeti változók egyszeri beállítása parancssorban	5
5	Az Eclipse fejlesztőkörnyezet	6
5.1	Letöltés és telepítés	6
5.2	Workspace-ek használata	6
5.3	A fejlesztőkörnyezet	7
5.4	Java alkalmazás létrehozása	9
5.5	Egyszerű Hello World alkalmazás	11
5.6	Futtatás	12
5.7	Debuggolás	15
5.8	Jar fájlok használata	19
5.8.1	Jar fájlok készítése	19
5.8.2	Jar fájlok felhasználása	22
5.9	Beállítások	23
6	Parancssori fordítás és futtatás	25
6.1	Egyszerű Hello World	25
6.2	Jar fájlok használata	26

1 Bevezetés

A Java nyelv platformfüggetlen (operációs rendszertől független), és maga a lefordított kód is az. Ezen az elven alapul az egész Java fordító- és futtatókörnyezet. Ennek köszönhetően Java programok fordítása során nem keletkezik pl. exe fájl, a Java programokat más módon kell elindítani.

A labor célja a Java fordító- és futtatókörnyezetének megismerése. Először az Eclipse fejlesztőkörnyezetet, amely elrejtje a parancssori végrehajtás kényelmetlenségeit. Ezután pedig az alapvető parancssori működést fogjuk megvizsgálni, mivel fontos lehet a Java fejlesztés mögötti modell ismerete.

2 JRE és JDK

Amikor a Java-t telepíteni kívánjuk, két lehetőség közül választhatunk:

- **Java Runtime Environment (JRE):** a Java futtatókörnyezet. Ennek a segítségével futtathatunk lefordított Java programokat, azonban Java programok lefordítására nincs lehetőség a JRE segítségével.
- **Java Development Kit (JDK):** Java programok fejlesztéséhez szükséges környezet. Ennek a segítségével fordíthatunk és futtathatunk is Java programokat. A JDK telepítője tartalmazza a JRE telepítőjét is.

Fejlesztéshez tehát mindenképpen a JDK-t kell használni. Amennyiben elkészítettünk és lefordítottunk egy Java programot, akkor annak a futtatásához elegendő a JRE megléte, vagyis az egyszerű felhasználók számára csak a JRE szükséges.

Windows operációs rendszeren alapesetben a **c:\Program Files\Java\jreX.Y.Z_W** könyvtárba települ a JRE, és a **c:\Program Files\Java\jdkX.Y.Z_W** könyvtárba települ a JDK, ahol az **X.Y.Z_W** az telepített verziószám.

3 Java és Javac

Két fontos program van, amelyet egy Java alapú fejlesztés során használni fogunk:

- **java:** a Java futtató, ezzel lehet lefordított Java programokat futtatni
- **javac:** a Java fordító (Java Compiler), ezzel lehet Java programokat lefordítani

A **java.exe** program megtalálható a JRE és a JDK telepítési könyvtárán belül a **bin** almappában:

```
c:\Program Files\Java\jreX.Y.Z_W\bin\java.exe  
c:\Program Files\Java\jdkX.Y.Z_W\bin\java.exe
```

A **javac.exe** program a JDK telepítési könyvtárában a **bin** mappa alatt található meg:

```
c:\Program Files\Java\jdkX.Y.Z_W\bin\javac.exe
```

4 Környezeti változók

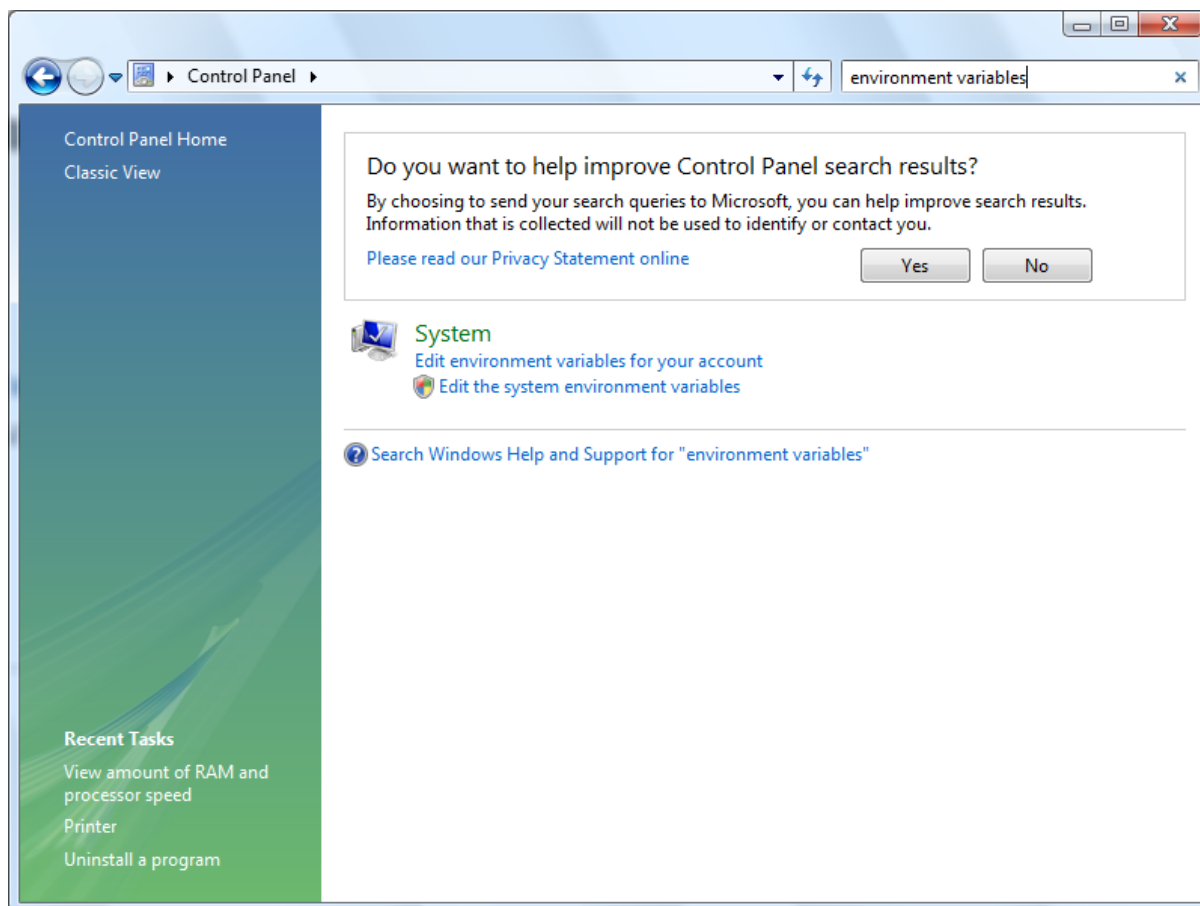
A Java fejlesztéshez célszerű néhány környezeti változót előre beállítani:

- **PATH:** itt keresi az operációs rendszer a parancssori programokat. Ide célszerű megadni a JDK telepítési könyvtára alatti **bin** könyvtárat, hogy a **java** és a **javac** is mindig elérhető legyen.
- **JAVA_HOME:** a JDK telepítési könyvtára. Nagyon sok Java program ebből a környezeti változóból indul ki, ezért fontos, hogy ezt beállítsuk.
- **CLASSPATH:** itt keresi a fordító és futtató környezet a lefordított fájlokat (osztályokat). Általában ezt a környezeti változót nem célszerű operációs rendszer szinten beállítani, mivel különböző Java programok különböző CLASSPATH-t használnak. Ezen kívül a CLASSPATH magának a **java** és **javac** parancsoknak is megadható külön parancssori argumentumként.

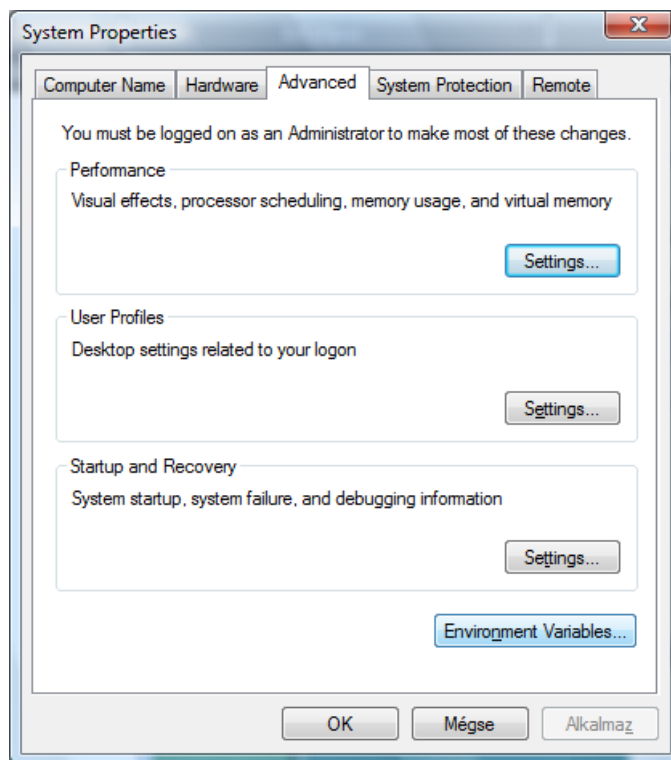
4.1 Környezeti változók beállítása Windows operációs rendszeren

A JDK telepítése után a következő lépéseket célszerű elvégezni, hogy a beállítások újraindítást követően is megmaradjanak.

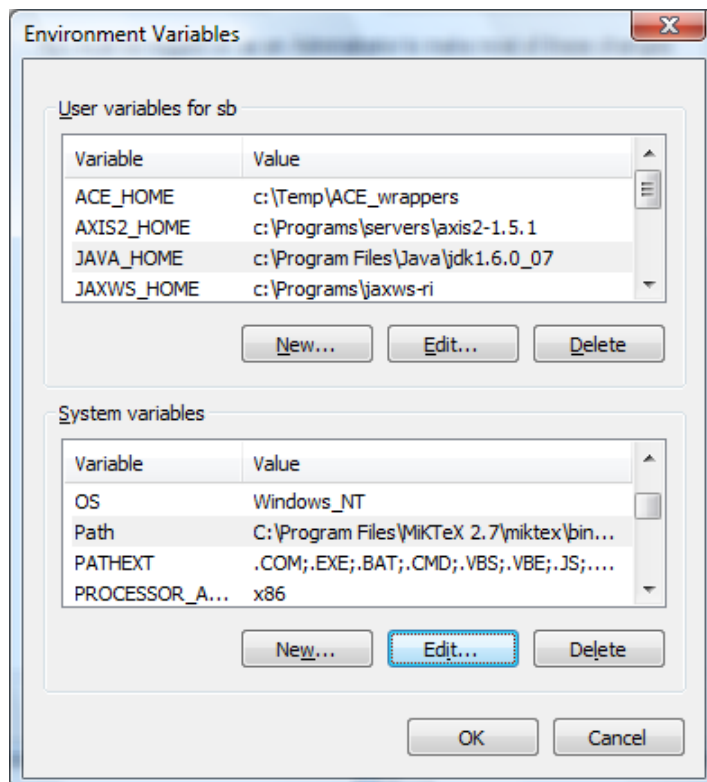
Indítsuk el a **Control Panel**-t (vezérlőpult), és írjuk be a jobb felső sarokban a keresőbe az „**environment variables**” („környezeti változók”) szöveget:



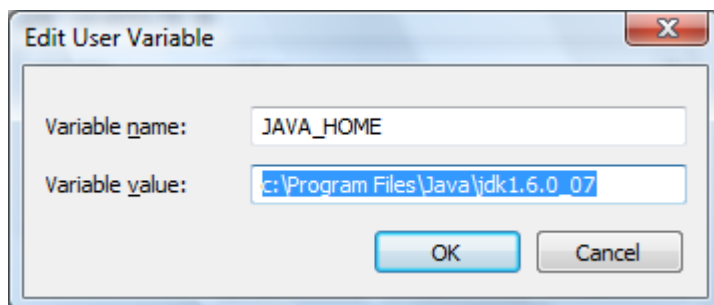
A System alatt megjelenik két találat is. Válasszuk ki valamelyiket, de célszerű a rendszer szintű beállításokat alkalmazni (**Edit the system environment variables**):



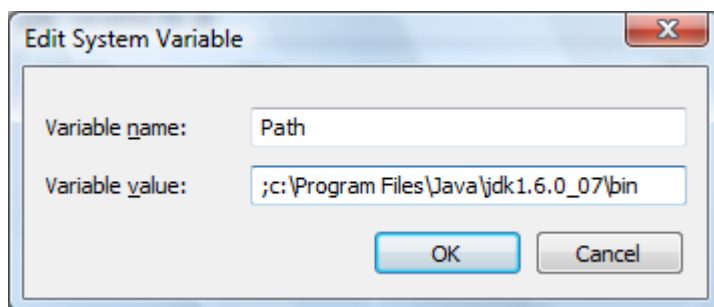
Itt válasszuk az **Environment Variables...** gombot:



A **User variables for ...** alatt nézzük meg a listát! Ha még nem létezik benne a **JAVA_HOME**, akkor a **New...** gombbal adjuk hozzá a listához! Ha már létezik, akkor az **Edit...** gombbal javítsuk ki az értéket, ha szükséges! Természetesen az **X.Y.Z_W** helyére a megfelelő verziószámot írjuk be, pl.:



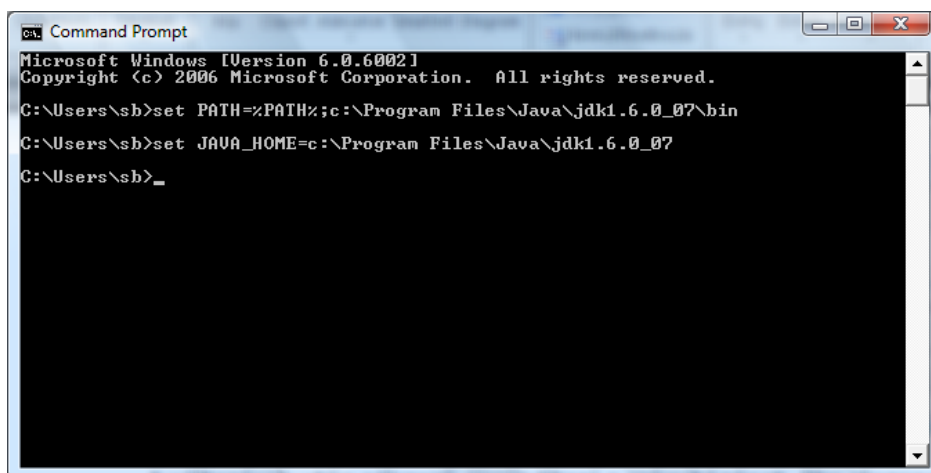
A **System variables** listában keressük meg a **PATH** környezeti változót, és a végére pontosvesszővel elválasztva adjuk meg a JDK alatti **bin** könyvtárat, pl.:



A változások után célszerű újraindítani a számítógépet. (De elegendő, ha csak bezárunk minden programot és újra elindítjuk őket.)

4.2 Környezeti változók egyszeri beállítása parancssorban

A környezeti változókat beállíthatjuk úgy is, hogy nyitunk egy parancssort, majd kiadjuk a következő parancsokat (természetesen a megfelelő verziószámokkal):



Ekkor a környezeti változók beállításai csak erre a konzol ablakra vonatkoznak, és csak addig élnek, amíg be nem zárjuk ezt az ablakot.

5 Az Eclipse fejlesztőkörnyezet

A parancssori fejlesztés kényelmetlenségeit kiküszöbölhetjük egy jó fejlesztőkörnyezettel. A jelenleg elérhető fontosabb nyílt forráskódú Java fejlesztőeszközök a következők:

- Eclipse: <http://www.eclipse.org/>
- Netbeans: <http://netbeans.org/>

A laboron az Eclipse-et fogjuk használni.

5.1 Letöltés és telepítés

Az Eclipse fejlesztőkörnyezetet a következő weboldalról tölthetjük le:

<http://www.eclipse.org/downloads/>

Itt több változatot is találunk. Ennek oka az, hogy az Eclipse egy eléggé moduláris eszköz, így különböző feladatokra különböző változatokat adtak ki.

Arra mindig ügyeljünk, hogy a JDK architektúrájának megfelelő Eclipse-et töltsük le! Tehát 32 bites JDK-hoz 32 bites Eclipse, 64 bites JDK-hoz 64 bites Eclipse használandó! Ellenkező esetben az Eclipse nem fog elindulni!

Számunkra az **Eclipse IDE for Java Developers** változat fontos. Ezzel egyszerű, önállóan is futó Java alkalmazásokat lehet készíteni.

Bonyolultabb, szerverre is telepíthető Java alkalmazások készítéséhez az **Eclipse IDE for Java EE Developers** változatot célszerű használni.

Ezek mellett még számos más változatot is megtalálhatunk a letöltési listában.

Fontos megjegyezni, hogy az Eclipse a háttérben ugyanazokat a parancssori programokat használja, mint amiket az előző fejezetben megismertünk, csupán egy grafikus felülettel könnyíti meg azok működtetését. Ezért az Eclipse használatához fel kell telepíteni a JDK-t és megfelelően beállítani a **JAVA_HOME** környezeti változót (ld. 4.1 szakasz).

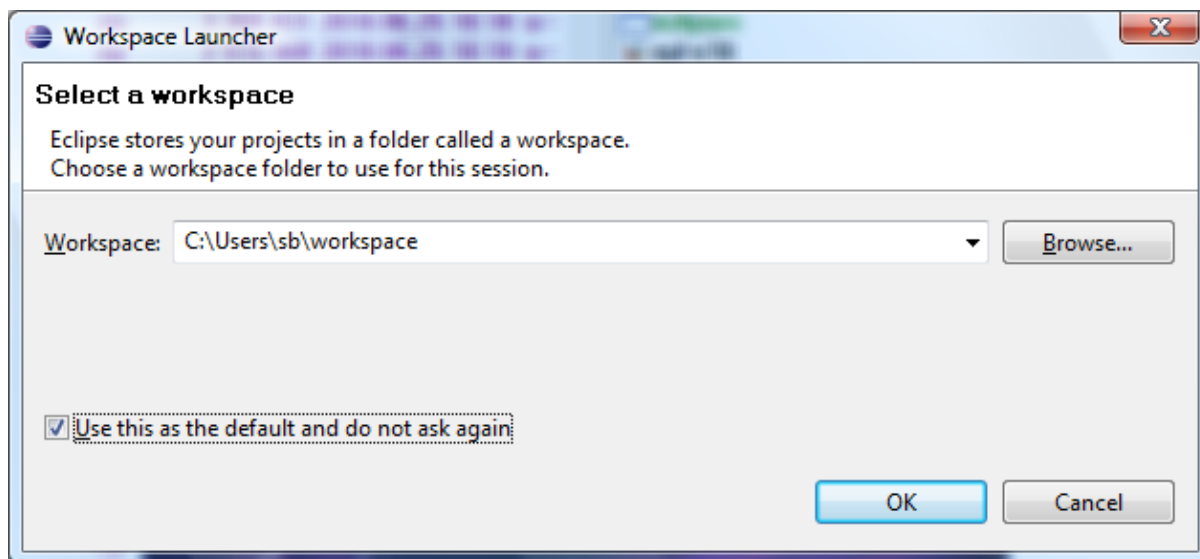
Az Eclipse telepítése nagyon egyszerű: csak ki kell csomagolni a letöltött zip fájlt, és az **eclipse.exe** programmal indítható az alkalmazás.

5.2 Workspace-ek használata

Az Eclipse-ben az alkalmazásokat projektekbe szervezhetjük. Amennyiben szükséges, a projektek között definiálhatunk függőségeket is, vagyis egy projektből használhatunk más projektekben lévő erőforrásokat. Az aktuálisan fejlesztett projekteket egy workspace fogja össze. Saját fejlesztéseknél általában elegendő egyetlen workspace-t létrehozni. Azonban céges-nagyvállalati környezetben célszerű lehet a projekteket valamilyen módon csoportosítani, és itt már szükség lehet több workspace használatára.

Számunkra elegendő egyetlen workspace is.

Első indításkor az Eclipse megkérdezi, hogy melyik könyvtár legyen a workspace könyvtára. Itt adjunk meg egy számunkra megfelelő könyvárat, és pipáljuk ki a „Use this as the default and do not ask again” opciót, hogy legközelebb már ne kelljen ezzel a kérdéssel foglalkozni:



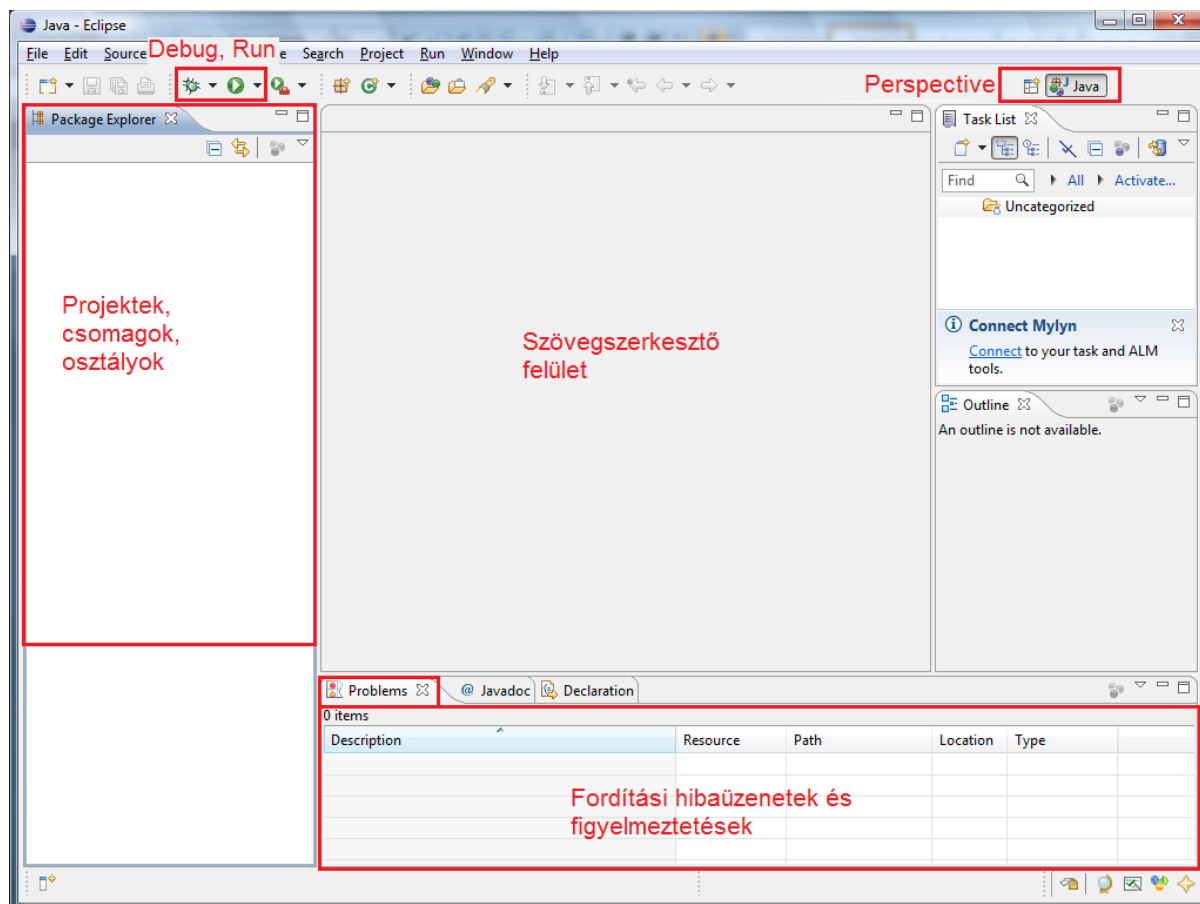
5.3 A fejlesztőkörnyezet

Amikor az Eclipse először elindul, a következő képernyő fogad minket:



Kattintsunk a jobb oldalon található nyílra!

Ezután és minden további indítás után már a következő képernyővel fogunk találkozni:



Bal oldalon található a workspace által összefogott projektek, csomagok és osztályok hierarchiája.

Lent a **Problems** fül alatt jelennek meg a fordítás során keletkező hibaüzenetek és figyelmeztetések.

Középen található a szövegszerkesztő felület.

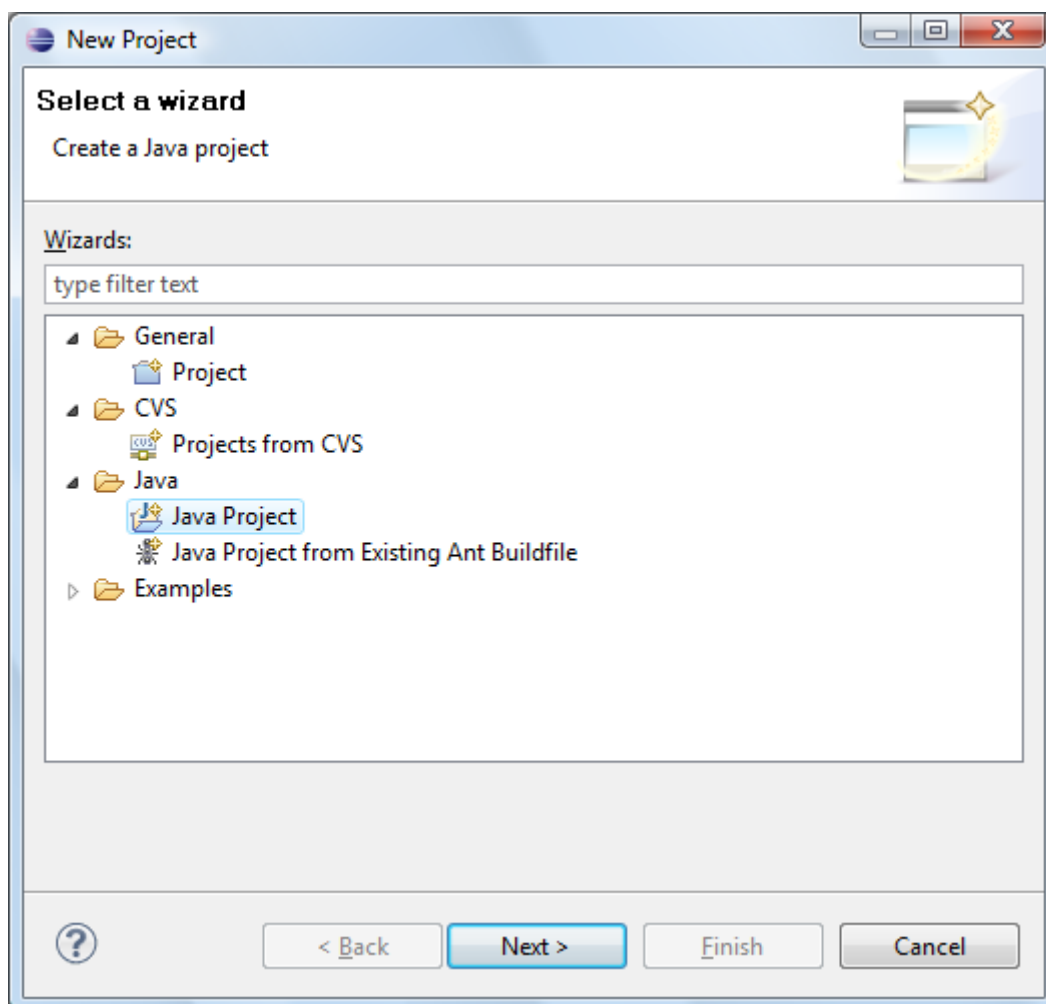
Fent az ikonok között foglal helyet a két legfontosabb parancs ikonja: **debug** és **run**.

A jobb felső sarokban választhatunk perspektívát. Egy perspektíva azt mondja meg, hogy az Eclipse ablakán melyik fül milyen elrendezésben található. A Java perspektíva a fenti képen is látható elrendezést tartalmazza. Debuggoláskor egy másfajta perspektíva fog megjelenni, de még számos egyéb elrendezés is létezik.

Ha valamelyik ablak nem látható, akkor az a **Window > Show View > ...** menüponttal előhozható. Ha itt sem találjuk az ablakot, vizsgáljuk meg, hogy a megfelelő perspektíva (nekünk a Java perspektíva fontos) van-e kiválasztva! Perspektívát a **Window > Open Perspective > ...** menüponttal válthatunk.

5.4 Java alkalmazás létrehozása

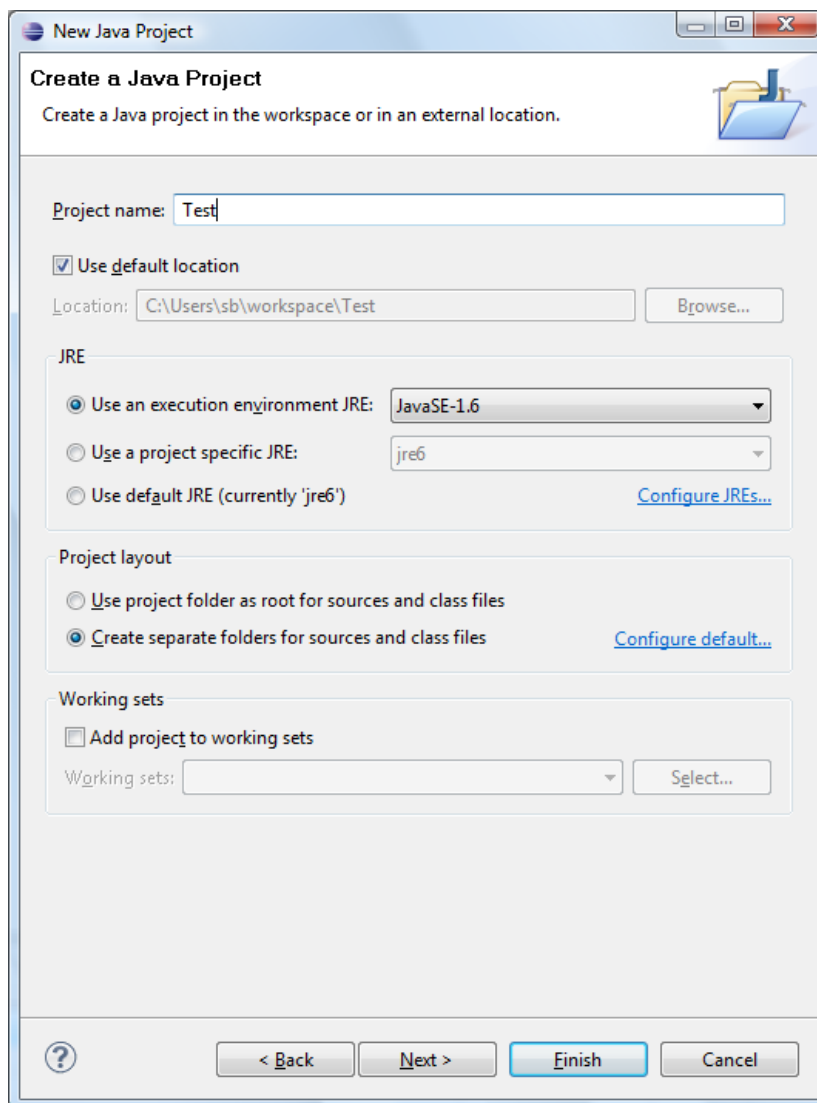
Egy új alkalmazás létrehozásához válasszuk a **File > New > Project...** menüpontot. Ekkor a következő ablak fogad minket:



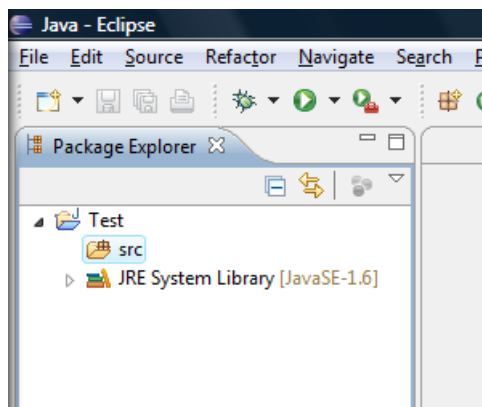
Itt találhatóak az adott Eclipse változat által támogatott projekt típusok. A lista hossza az adott Eclipse változattól és az aktuálisan kiválasztott perspektívától függően jóval hosszabb is lehet.

Ha ebben az ablakban nem találjuk a megfelelő projekt típust, akkor megintcsak vizsgáljuk meg, hogy a megfelelő perspektíva van-e nyitva (nekünk a Java perspektívára van szükségünk)!

Egy egyszerű Java alkalmazás készítéséhez válasszuk a **Java** mappa alatt a **Java Project** elemet, majd kattintsunk a **Next** gombra:

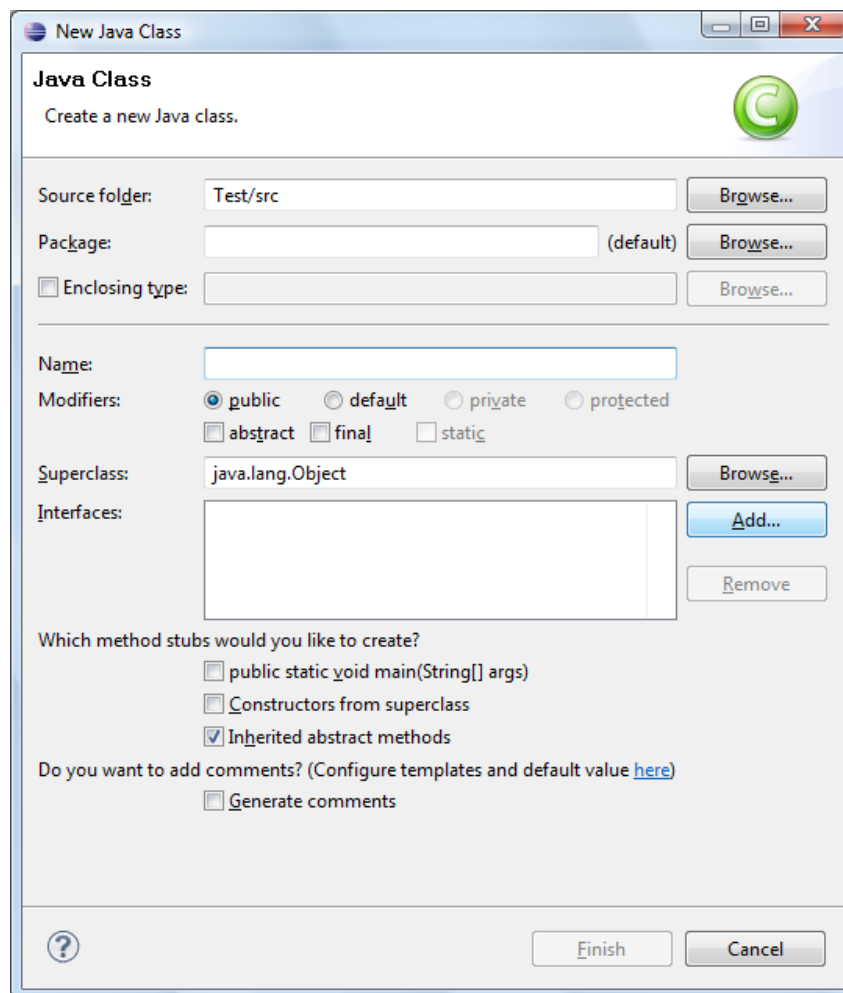


Itt megadhatjuk a projekt nevét, végül pedig kattintsunk a **Finish** gombra! Ezután létrejön a Java projekt, amibe elkezdhetjük felvenni a Java forrásfájlokat:



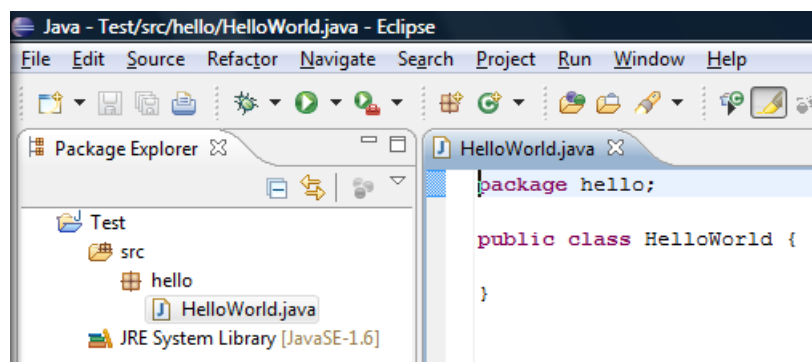
5.5 Egyszerű Hello World alkalmazás

Hozzunk létre egy Java projektet! Ezután válasszuk a **File > New > Class** menüpontot, aminek hatására a következő ablak jelenik meg:



Itt megadhatunk egy csomagnevet (package), az osztály nevét (name), és egyéb opciókat is beállíthatunk, pl. ősoosztályok, ősinterfészek, legyen-e automatikusan generált main függvény, stb.

A csomagnév legyen "hello", az osztály neve pedig legyen "HelloWorld"! Ezután kattintsunk a **Finish** gombra. Az eredmény a következő:



Egészítsük ki a jobb oldali forráskódot a következőképpen:

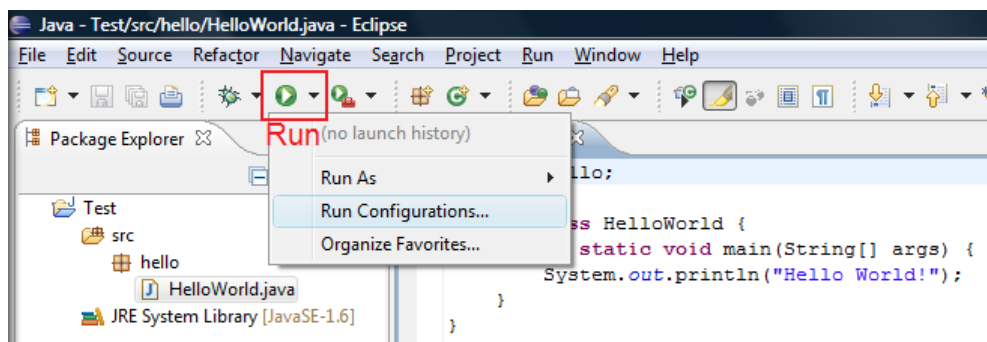
```
package hello;

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

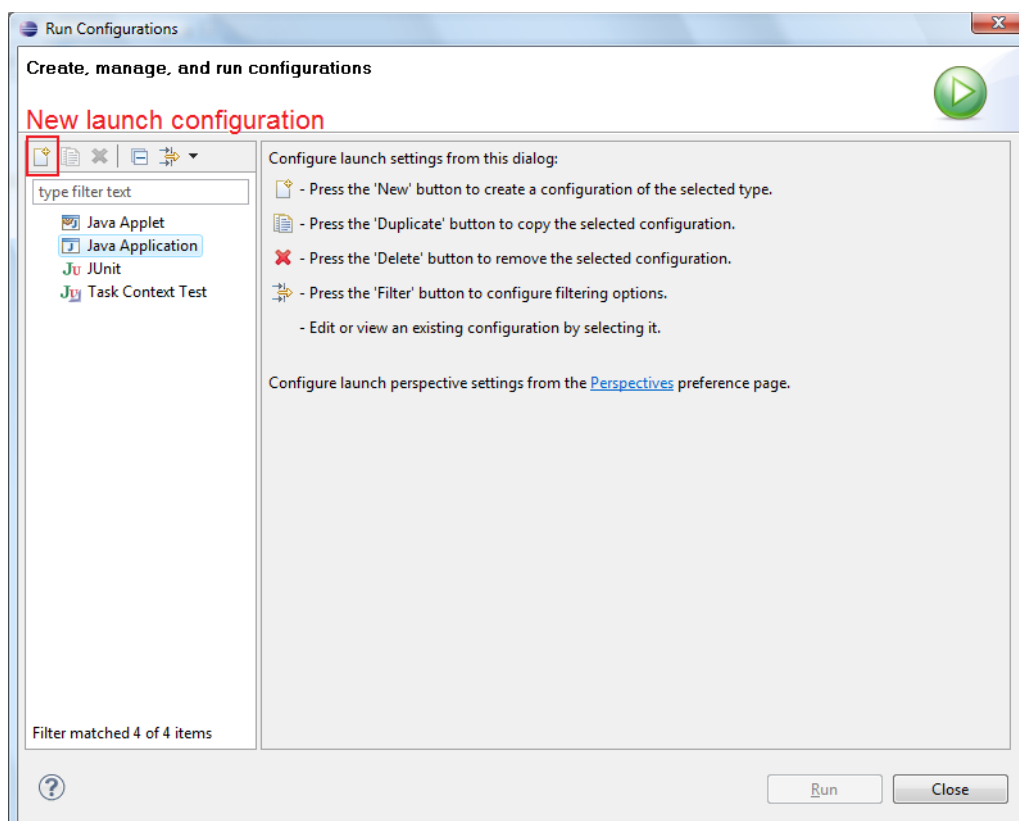
Ezzel elkészült az első Java alkalmazásunk Eclipse alatt.

5.6 Futtatás

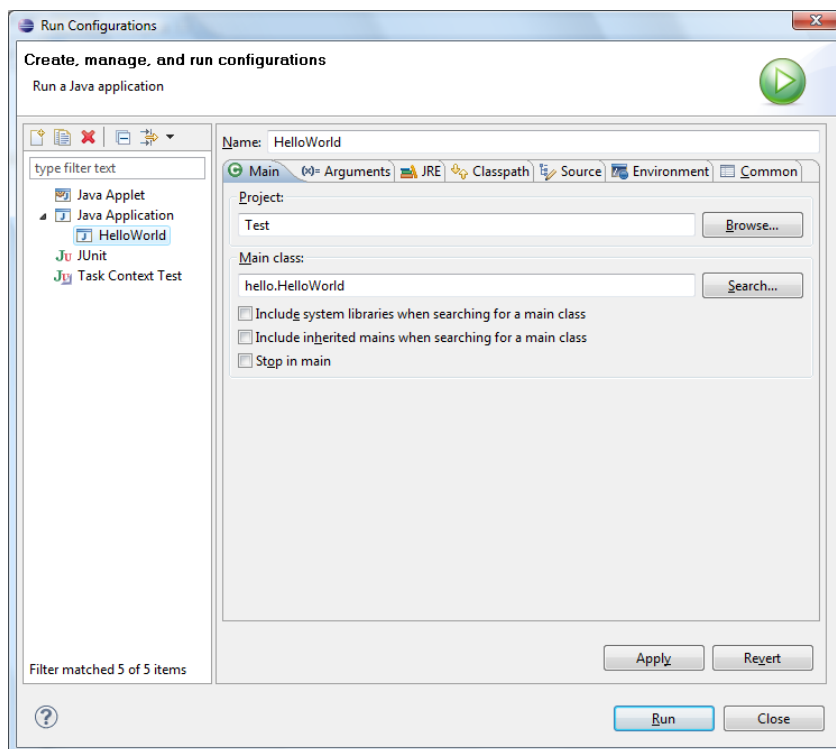
Az alkalmazás futtatásához az Eclipse ablakának felső sorában lévő eszköztáron a **Run** ikon melletti nyilat nyissuk le, és válasszuk a **Run Configurations...** menüpontot:



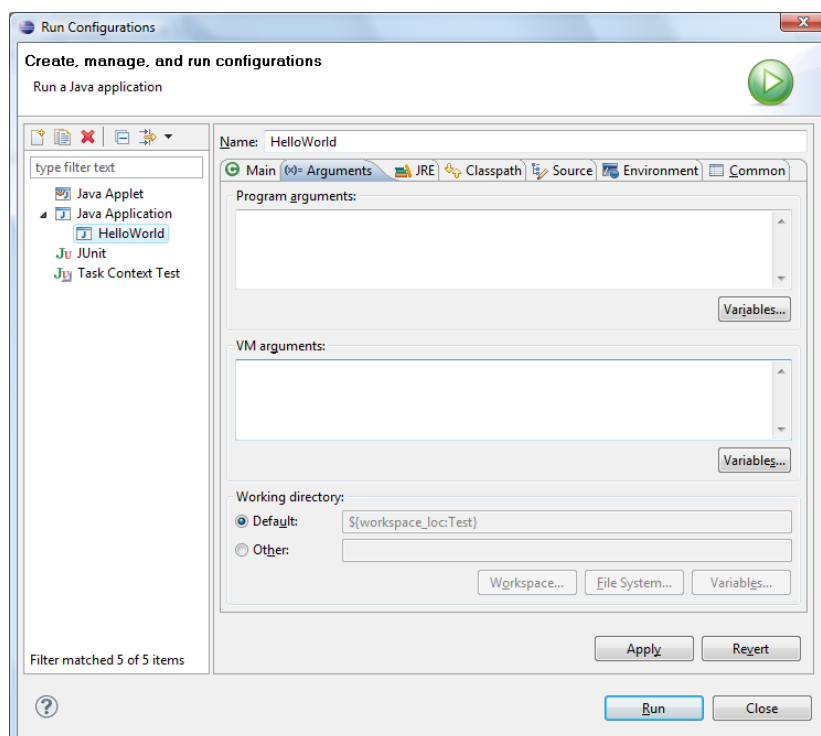
A következő ablak fog megjelenni:



Itt bal oldalon válasszuk a **Java Application** elemet, majd kattintsunk a **New launch configuration** ikonra. Ennek hatására az Eclipse az aktuális projektből felismeri, hogy a HelloWorld osztályban van main függvény, és a következő futtatási konfigurációt hozza létre:

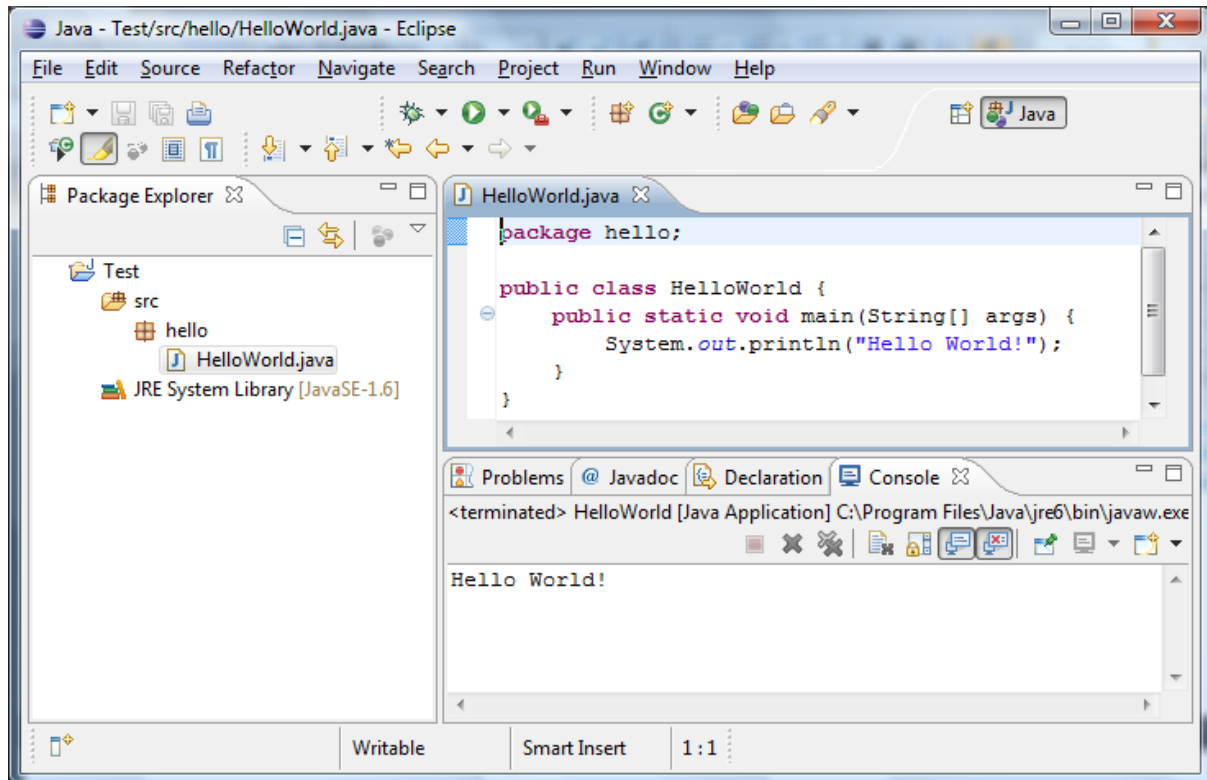


Az **Arguments** fülön egyéb opciókat is megadhatunk, pl. a program parancssori argumentumait illetve a Java virtuális gép (VM) speciális paramétereit, de ezeket most hagyjuk üresen:

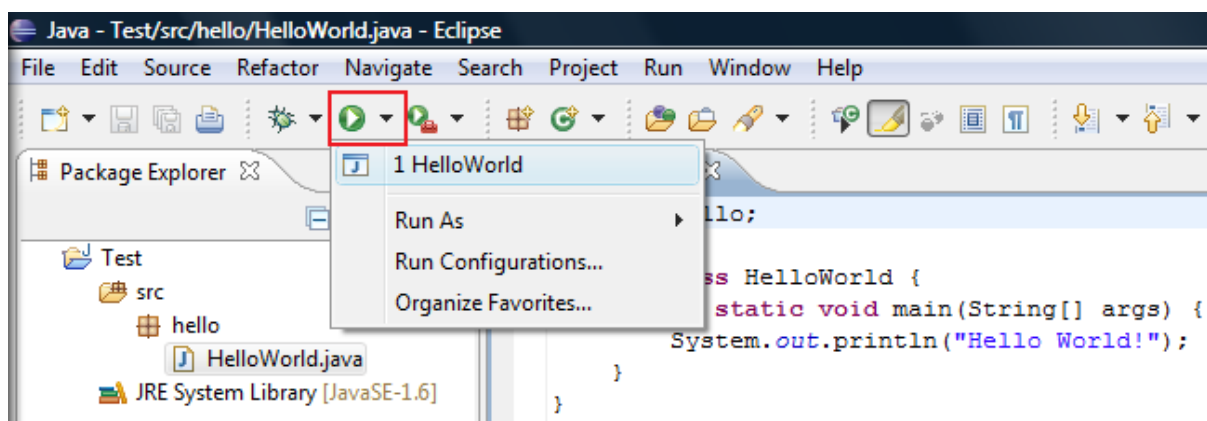


A többi fülön még egyéb opciókat is tehetünk, ha szükséges, de most ezek nem fontosak.

A **Run** gombra kattintva elindul az alkalmazás, és lent egy **Console** nevű fülön láthatjuk az eredményt:



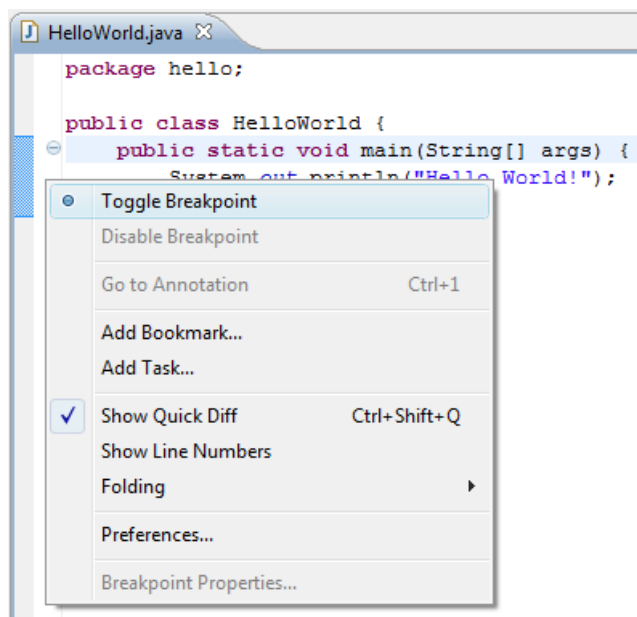
Ha változtatunk a programon és még egyszer futtatni kívánjuk, akkor elegendő a **Run** ikon mellett nyíllra kattintva kiválasztani a megfelelő konfigurációt:



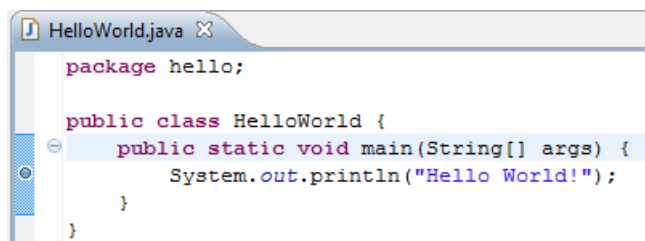
5.7 Debuggolás

Az Eclipse nagy előnye az egyszerű parancssori fejlesztéshez képest, hogy támogatja a debuggolást is.

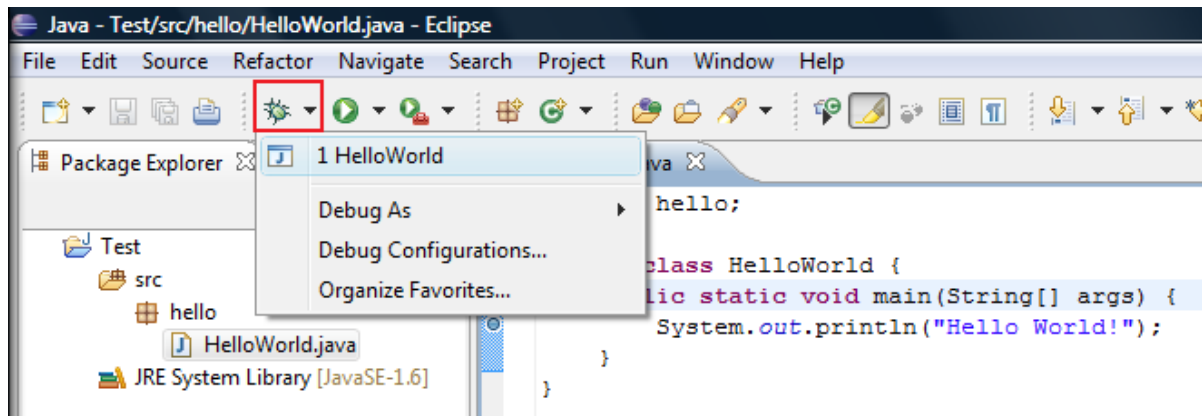
Töréspont elhelyezéséhez kattintsunk jobb gombbal a sor elején, majd válasszuk a **Toggle Breakpoint** menüpontot:



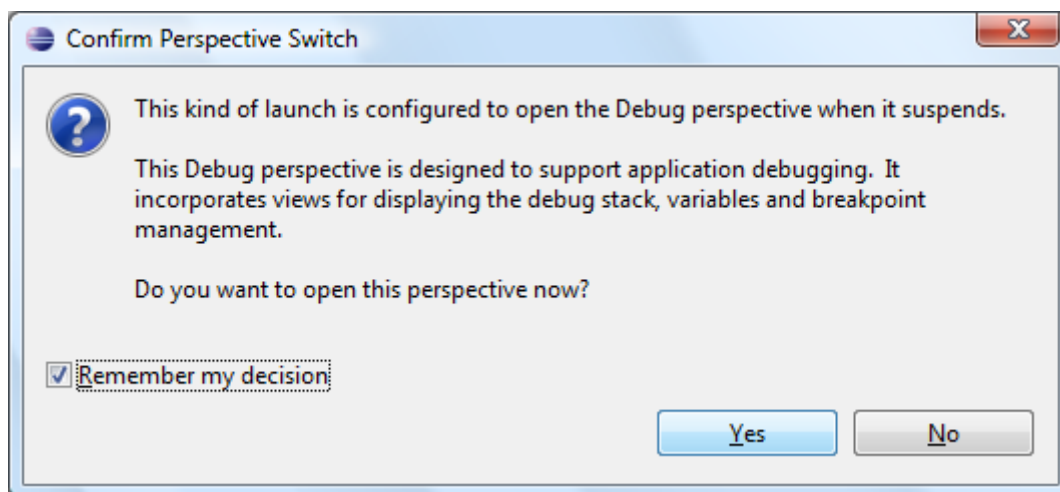
Ennek hatására egy pötty jelenik meg a sor elején, ahol a program futása debuggolás során meg fog állni:



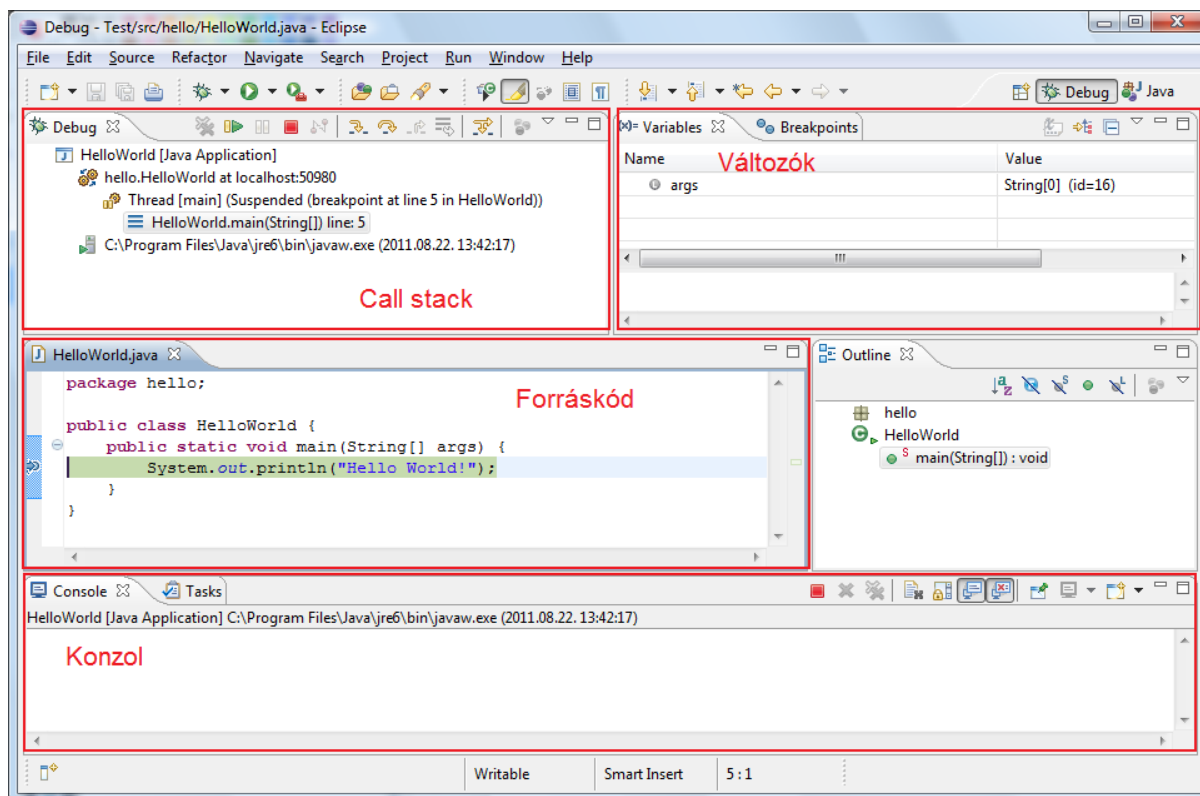
A debuggoláshoz a felső ikonsorban kattintsunk a kis bogár melletti nyíllra és válasszuk az előző szakaszban megadott **HelloWorld** konfigurációt:



Ennek hatására felugrik a következő ablak, amely arra kérdez rá, hogy szeretnénk-e átváltani **Debug** perspektívába:

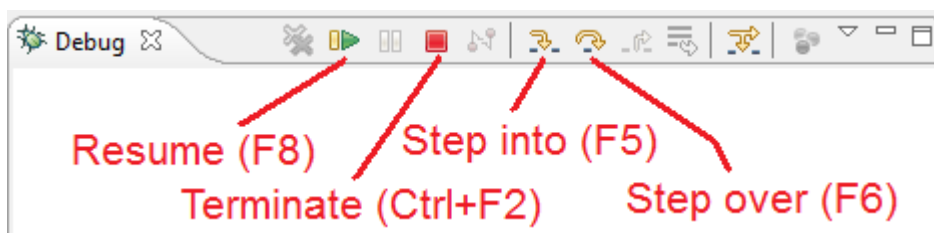


Itt pipáljuk ki a **Remember my decision** opciót, és kattintsunk a **Yes** gombra. Ennek hatására a következő módon alakul át az Eclipse ablaka:



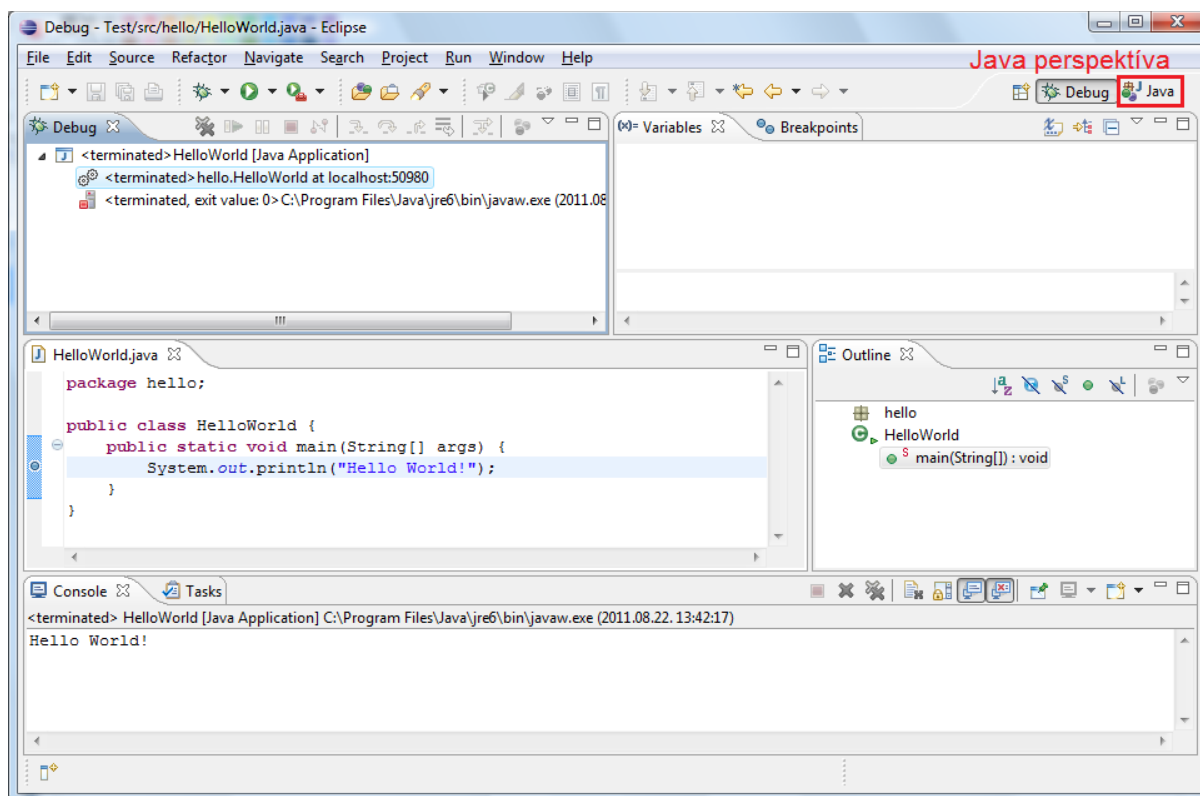
A Debug ablakban láthatjuk, hogy éppen hol tartunk a hívási hierarchiában (call stack). A változók ablakban vizsgálhatjuk meg, hogy az egyes változók éppen milyen értéket vesznek fel. A forráskód ablakban követhetjük nyomon, hogy a program melyik sorában tartunk éppen. A konzol ablak pedig a konzolos kimenetet ábrázolja.

A **Debug** fül mellett található néhány hasznos ikon:



A **Resume** gombbal futtathatjuk tovább a programot a következő breakpoint-ig. A **Terminate** gombbal megszakíthatjuk a futást. A **Step into** gomb belelép a soron következő függvényhívásba. A **Step over** lefuttatja a következő függvényhívást anélkül, hogy belelépne, és folytatja a futást a következő sornál.

A **Resume** gombbal engedjük tovább a program futását! Ekkor a konzol ablakon megjelenik a várt szöveg:

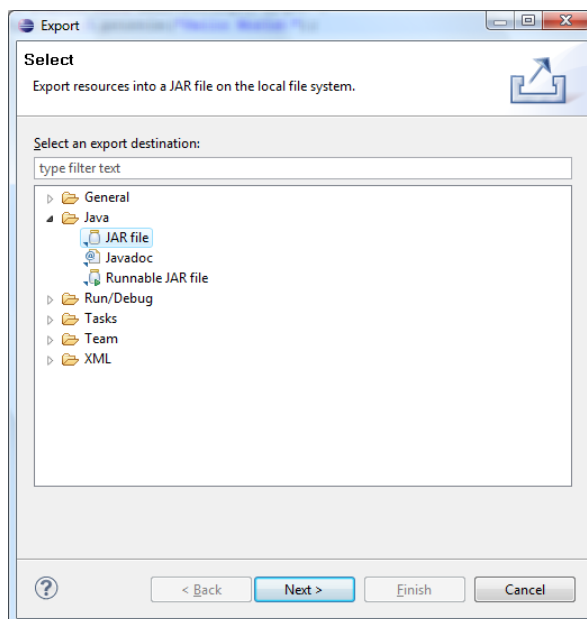


A jobb felső sarokban a **Java perspektíva** választásával térhetünk vissza a megszokott Eclipse elrendezésbe.

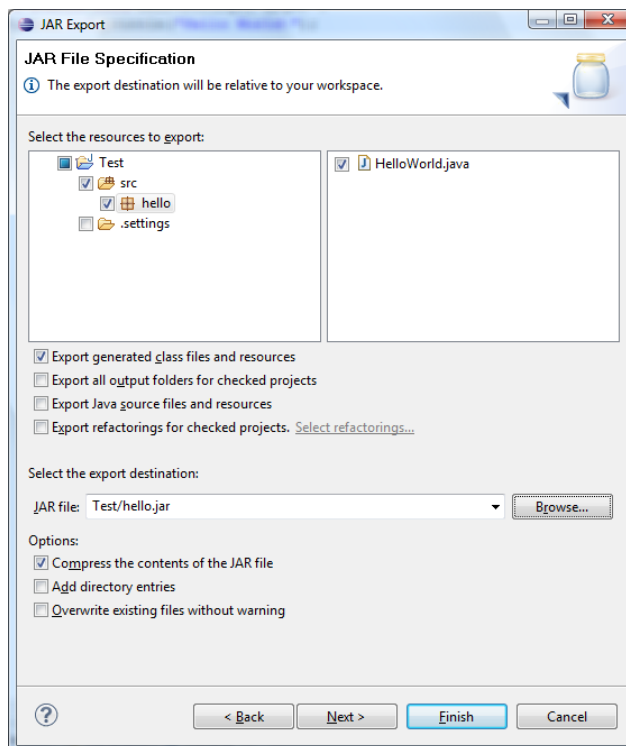
5.8 Jar fájlok használata

5.8.1 Jar fájlok készítése

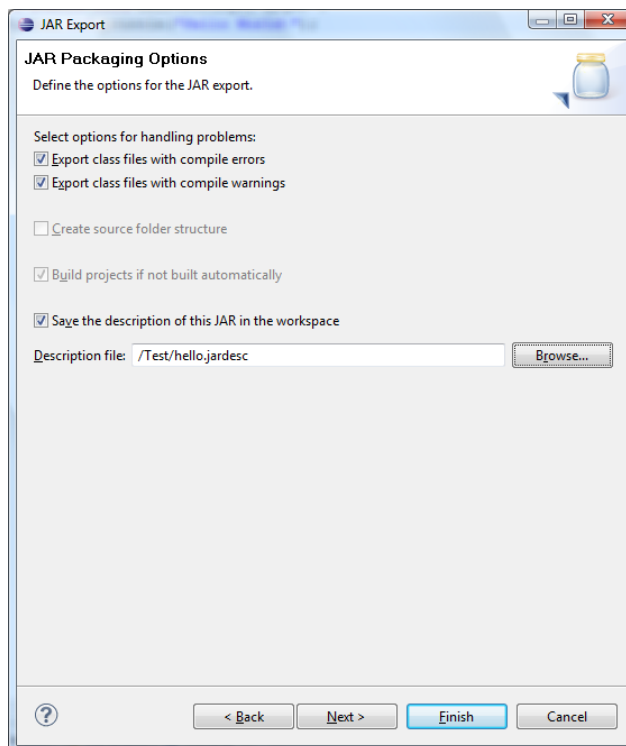
Mivel a sok **.class** fájl önmagában történő hordozása kényelmetlen, lehetőség van ezeket egy fájlba összefogni. Egy ilyen összefogó fájl valójában egy normál **.zip** fájl **.jar** kiterjesztéssel. Eclipse alól is van lehetőség **.jar** fájlok készítésére. Ehhez kattintsunk a **File > Export...** menüpontra, majd a megjelenő ablakban a Java alatt válasszuk a JAR file opciót:



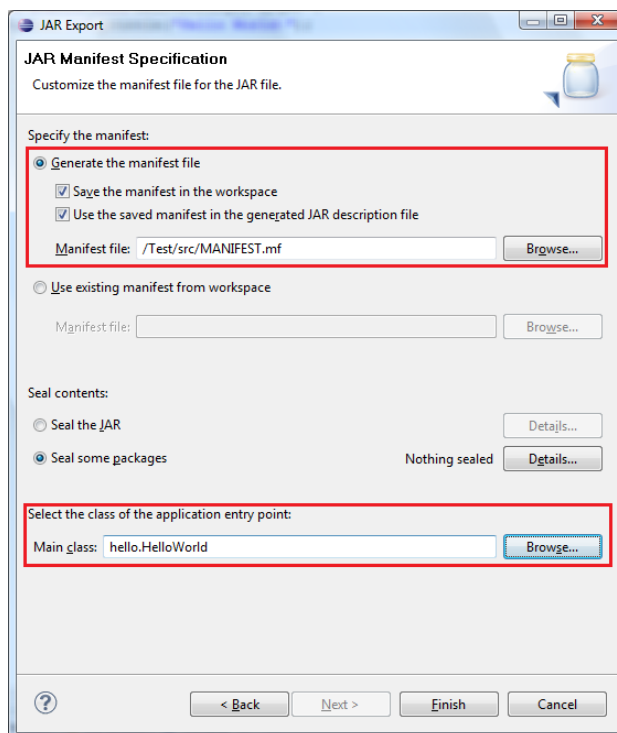
Ezután kattintsunk a **Next** gombra! A következő oldalon kiálthatjuk, hogy a projektből mely fájlokat kívánunk beletenni a **.jar** fájlba, illetve, hogy hova kerüljön az elkészült **.jar** fájl, és annak mi legyen a neve. Például:



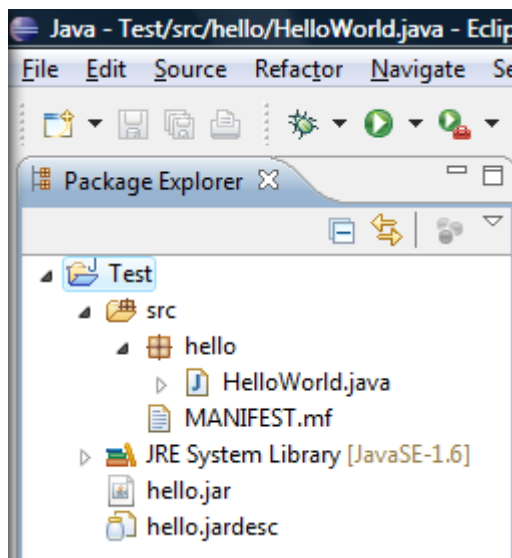
Kattintsunk ismét a **Next** gombra! A következő oldalon pipáljuk ki a „**Save the description of this JAR in workspace**” opciót és adjunk meg egy **Description file** nevet! Ez azért fontos, mert így majd nem kell mindig újra lefuttatni ezt az export varázslót. Az adatok tehát a következőképpen néznek ki:



Kattintsunk ismét a **Next** gombra! A megjelenő ablakban a **Generate the manifest file** opciót válasszuk, majd alatta töltsük ki a következő piros téglalapokkal jelölt beállításokat:



Végül kattintsunk a **Finish** gombra! Ekkor elkészül a manifest fájl, a jardec fájl és a jar fájl is:



Még egyszer összefoglalva:

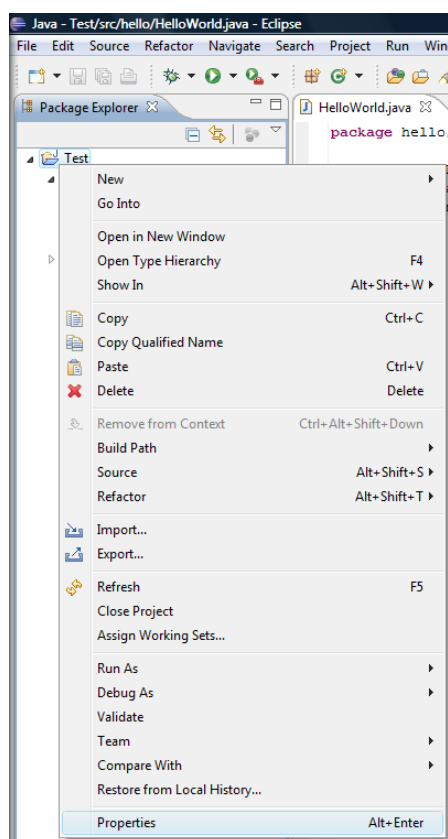
- A **MANIFEST.mf** tartalmazza a főprogramot, vagyis azt az osztálynevet, amelyiknek van **main** függvénye.

- A **hello.jar**desc egy olyan leírás, amely segítségével a **.jar** fájlt mindig újra létrehozhatjuk anélkül, hogy a teljes export varázslót újra le kéne futtatni. Ehhez kattintsunk jobb gombbal a **hello.jar**desc fájlra, és válasszuk a **Create JAR** menüpontot!
- A **hello.jar** tartalmazza az exportált **.class** fájlokat.

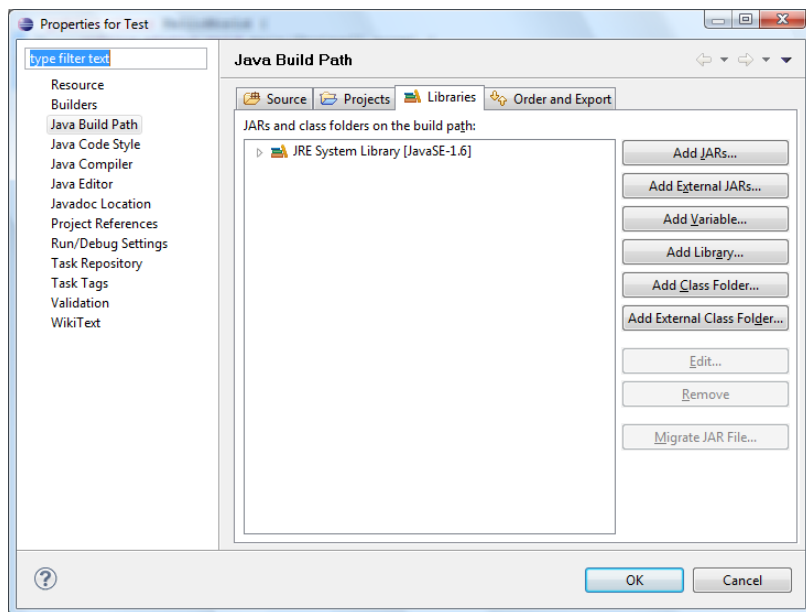
Amennyiben később új forrásfájlokat adunk a projekthez, akkor majd azokat is bele kell tenni a **.jar**desc leírásba. Ehhez kattintsunk duplán a **hello.jar**desc fájlra, és módosítsuk a kívánt beállításokat.

5.8.2 Jar fájlok felhasználása

Sokszor lehet szükség arra, hogy másoktól kapott **.jar** fájlokat használjunk fel a projektünkben. Ehhez kattintsunk jobb gombbal a **Package Explorer**-ben a projekten, és válasszuk a **Properties** menüpontot:

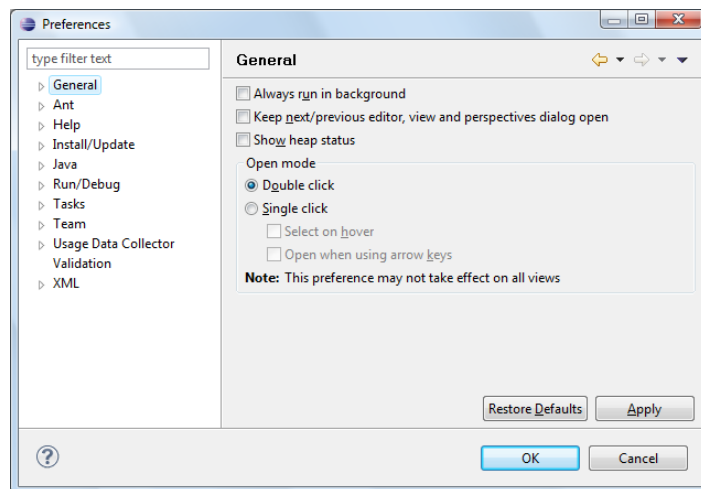


A megjelenő ablakban a **Java Build Path** alatt a **Libraries** fülön adhatunk külső **.jar** fájlokat a projekthez:



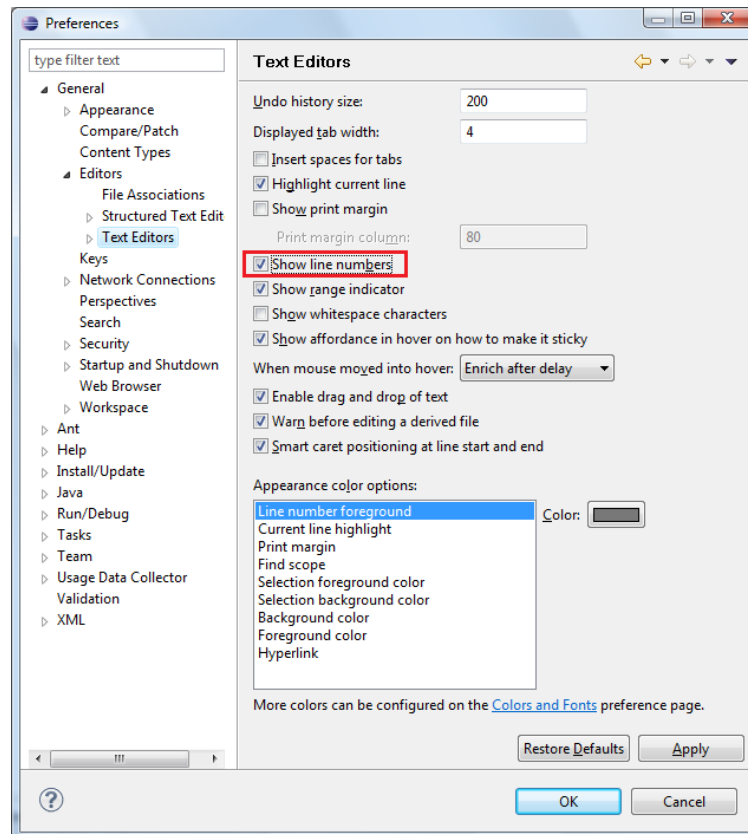
5.9 Beállítások

Az Eclipse fejlesztőkörnyezet nagyon jól testre szabható. A beállításokat a **Window > Preferences** menüpont alatt változtathatjuk meg:



Itt számos opció található, többek között például a forráskód színezésének és formázásának lehetőségei, a gyorsbillentyűk, stb.

Egy fontos opció alap esetben ki van kapcsolva, ezt célszerű engedélyezni. Ez pedig a sorok számozása, ami hasznos lehet a hibaüzenetek esetén a hiba helyének megkeresésekor. A sorok számozását a **General > Editors > Text Editors** lapon kapcsolhatjuk be:



6 Parancssori fordítás és futtatás

6.1 Egyszerű Hello World

Figyelni kell arra, hogy Java-ban az osztály és a fájl nevének meg kell egyeznie!

Írjuk meg notepad-ben, vagy egy tetszőleges szövegszerkesztőben a következő Java programot **Hello.java** néven:

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

Ezután parancssorból ugyanebből a könyvtárból a fordításhoz adjuk ki a következő parancsot:

```
javac Hello.java
```

Ekkor létrejön egy **Hello.class** fájl, ami a Java bájtkódot tartalmazza. Ez önmagában nem futtatható, mint egy exe fájl, azonban ezt a Java virtuális gép (JVM) végre tudja hajtani. Ehhez adjuk ki a következő parancsot:

```
java Hello
```

Figyeljünk arra, hogy csak az osztály nevét kell megadni, a **.class** kiterjesztést nem szabad kiírni! A futás eredménye:

```
Hello World!
```

Előfordulhat, hogy a program nem fut le, hanem a következő hibaüzenetet kapjuk:

```
Exception in thread "main" java.lang.NoClassDefFoundError: Hello
Caused by: java.lang.ClassNotFoundException: Hello
    at java.net.URLClassLoader$1.run(Unknown Source)
    at java.security.AccessController.doPrivileged(Native Method)
    at java.net.URLClassLoader.findClass(Unknown Source)
    at java.lang.ClassLoader.loadClass(Unknown Source)
    at sun.misc.Launcher$AppClassLoader.loadClass(Unknown Source)
    at java.lang.ClassLoader.loadClass(Unknown Source)
    at java.lang.ClassLoader.loadClassInternal(Unknown Source)
Could not find the main class: Hello. Program will exit.
```

Vagyis a **Hello** nevű osztály nem található. Ennek két oka lehet:

1. Rossz könyvtárból adtuk ki a **java Hello** parancsot, vagyis a **Hello.class** fájl nem létezik az aktuális könyvtárban. A megoldáshoz fordítsuk le a **Hello.java** programot, és a keletkező **Hello.class** könyvtárából adjuk ki a **java Hello** parancsot.
2. A **CLASSPATH** környezeti változó rosszul van beállítva, és az aktuális könyvtár hiányzik belőle. A megoldáshoz adjuk hozzá az aktuális könyvtárat (ezt ponttal jelöljük) a **CLASSPATH** értékéhez a következő parancssal:

```
set CLASSPATH=%CLASSPATH%;.
```

6.2 Jar fájlok használata

Mivel a sok **.class** fájl önmagában történő hordozása kényelmetlen, lehetőség van ezeket egy fájlba összefogni. Egy ilyen összefogó fájl valójában egy normál **.zip** fájl **.jar** kiterjesztéssel.

Ilyen **.jar** fájlokat a **jar.exe** paranccsal hozhatunk létre, amely ugyancsak a JDK bin könyvtárában található. Az előző szakaszban bemutatott példánál maradván a **>**-rel jelölt könyvtárból adjuk ki a parancsot:

```
jar cf hello.jar sample/calc/*.class sample/echo/*.class sample/prog/*.class
```

A **cf** opció azt jelenti, hogy létre akarunk hozni egy új **.jar** fájlt. A **hello.jar** a készíthető **.jar** fájl neve. A többi paraméter azokat a fájlokat tartalmazza, amelyeket bele kívánunk tenni a **.jar** fájlba.

Ezután már csak ezt a **.jar** fájlt kell hordozni. Ennek a segítségével a következő módon indíthatjuk el a programot:

```
java -cp hello.jar sample.prog.Program
```

A java parancsnak a **-cp** paraméterrel adhatjuk meg a CLASSPATH-t, vagyis azt, hogy hol keresse a **.class** fájlokat. Jelen esetben ez azt jelenti, hogy a **.jar** fájlból keresse ki a **.class** fájlokat.

Ez a módszer azért hasznos, mert így teljes API könyvtárakat lehet **.jar** fájlként hordozni. Például az alap Java API is ilyen módon települ fel a gépünkre. Ha belenézünk a JRE illetve a JDK könyvtárainak mélyére, akkor ott megtaláljuk a Java API alap **.jar** könyvtárait. A legfontosabb ezek közül az **rt.jar**, ez tartalmazza a Java API legnagyobb részét.

Lehetőség van arra is, hogy a **main** függvényt tartalmazó osztály nevét ne kelljen megadni explicit. Ehhez egy ún. manifest leíró fájlt kell létrehozni. A **>**-rel jelölt könyvtárban készítsünk egy **MANIFEST.mf** nevű szövegfájlt a következő tartalommal:

```
Main-Class: sample.prog.Program
```

Fontos: ügyeljünk arra, hogy legyen egy üres sor a fájl végén, tehát mindenképpen üssünk enter-t a sor végén!

Ezután adjuk ki a következő parancsot:

```
jar cfm hello.jar MANIFEST.mf sample/calc/*.class sample/echo/*.class  
sample/prog/*.class
```

Itt az **m** opció azt jelenti, hogy manifest fájlt is adunk meg. Ezt a fájlt a **.jar** fájl neve után közvetlenül kell megadni.

Ezt követően a program a következő paranccsal is indítható:

```
java -jar hello.jar
```

Sőt, ha az operációs rendszerből duplán kattintunk a **.jar** fájlra, akkor úgy indul a program, mintha ez egy **.exe** fájl lenne.