A Java nyelv

IV. rész – appletek, hálózati alkalmazások fejlesztése

A Java magasfokú objektumorientáltsága következtében egy Java program osztályok és objektumok összefüggő halmazát jelenti. A program futtatása nem más mint adott osztályok vagy objektumok metódusainak a meghívása.

Egy program megírása tulajdonképpen egy új osztály definiálását jelenti, a programvezérlés pedig nem más mint metódusok felüldefiniálása, megírása, illetve a megfelelő eseménykezelők meghatározása.

A Java nyelv fejlesztőkörnyezetei számos előredefiniált osztályt tartalmaznak, csomagokba szervezve, a programozás nagy részét ezeknek az osztályoknak a megfelelő használata teszi ki, lényeges tehát, hogy minél mélyebb belátást nyerjünk a Java csomagokba.

A Java programokat két nagy kategóriába sorolhatjuk:

Alkalmazások – önálló Java programok, fordításuk a *javac* fordítóval, végrehajtásuk a *java* szabványos értelmezővel történik.

Appletek – HTML oldalba beszerkeszthető Java programok. Végrehajtásukat a böngészők végzik vagy megtekinthetők az *appletviewer* segédprogrammal is.

A Java forrásállományainak neve mindig a *.java* kiterjesztésből és a fájlban szereplő egyetlen publikus osztály (***public*** módosítóval ellátott osztály) nevéből áll.

A forrásállományok megírásakor ajánlatos betartani a következő konvenciókat:

Az osztályok nevei mindig nagybetűvel kezdődnek (pl. *Thread*).

Az azonosítók nevei mindig kisbetűvel kezdődnek (pl. *first*).

Minden névben a szóösszetevők nagybetűvel kezdődnek (pl. *firstStep, ActionListener*).

A konstansok csupa nagybetűsek (pl. *PI*).

A { blokk-kezdő mindig a legutolsó utasítás után, a sor végén áll.

A } blokkzáró a sor elejeén a blokkot megelőző utasítással egy oszlopban áll.

```
public class Autó implements Serializable {
   public Autó(String r, String t) {
         rendszám = r;
         tulajdonos = t;
   }
}
```

A Java fordító a forrásállományt egy köztes byte-kódra fordítja. A forrásban definiált minden egyes osztály egy különálló *osztálynév.class* állományba kerül. A *java* értelmező ezt a byte-kódot hajtja végre. Futás közben egy adott osztály akkor töltődik be, amikor először hivatkozunk rá. A betöltést a *java.lang.ClassLoader* osztály felhasználásával lehet vezérelni. A Java értelmezője csak a *CLASSPATH* környezeti változóban szereplő könyvtárakban megtalálható osztályokat látja, vigyázzunk tehát, hogy ez a környezeti változó mindig legyen beállítva az *autoexec.bat*-ban.

Egy alkalmazás akkor fejeződik be, ha az értelmező a byte-kód végére ér, vagy ha a program a *System.exit* metódust hívja meg.

A Java alkalmazások rendelkezhetnek grafikus felülettel is, amely rendszerint egy *java.awt.Frame* ablakkeret objektumra épül, így érdemes már magát az alkalmazást ebből az osztályból származtatni. Ablakot megjelentetni a show() metódus segítségével lehet.

Appletek

Az Applet típusú programok beágyazhatók HTML oldalakba. Ez a beágyazás azonban nem forráskód szintjén történik (mint JavaScript esetén) hanem a *.class* bináris állományt tölti le és futtatja a navigátor. Minden applet a *java.applet.Applet* osztály leszármazottja kell, hogy legyen. A *java.applet* csomag tartalmaz minden appletspecifikus osztályt és interfészt.

Egy applet beágyazásakor a böngészőnek is fontos szerep jut. A böngésző létrehozza az applethez tartozó és az *AppletContext* interfészt implementáló objektumot, amely a tulajdonképpeni beágyazó objektum és a böngészőspecifikus szolgáltatásokat a böngészőhöz tartozó, az *AppletStub* interfészt implementáló objektumon keresztül veheti igénybe.

Egy appletet a következő HTML kulcsszóval lehet beágyazni:

```
<APPLET     [CODEBASE=url] [ARCHIVE=archívum,...]
 CODE=fájlnév vagy OBJECT=objektumnév
 [ALT=szöveg]
 [NAME=azonosító]
 WIDTH=szám HEIGHT=szám
 [ALIGN=érték]
 [VSPACE=szám] [HSPACE=szám]
 [MAYSCRIPT]
>
[PARAM NAME=azonosító VALUE=érték]
...
</APPLET>
```

A *CODEBASE* az applet kódját tartalmazó könyvtár címe. Ha nincs megadva, akkor a HTML címén keresi az appletet. Az *ARCHIVE* az applet kódját és erőforrásait tartalmazó archívumok. A *CODE* az applet kódját tartalmazó állomány neve, az *OBJECT* az appletet tartalmazó fájl neve. Az *ALT* segítségével egy szöveget adhatunk meg, amelyik akkor jelenik meg, ha a böngésző nem képes grafikusan megjelentetni az appletet. A *NAME* segítségével is hivatkozni lehet az appletre. A *WIDTH* és *HEIGHT* a grafikus terület szélességét és magasságát adja meg, az *ALIGN*-nal pedig igazítani lehet az applet ábrázolását. A *VSPACE* és a *HSPACE* az applet felett és mellett üresen hagyandó képpontok száma. Az applet csak akkor kommunikálhat JavaScript-tel, ha a *MAYSCRIPT* is szerepel a paraméterek között. Ha az appletnek paramétereket is akarunk átadni, akkor ezt a *PARAM* kulcsszóval tehetjük meg. A paramétereket az *Applet* osztály *String getParameter(String)* metódusával lehet lekérdezni, ahol az argumentum a lekérdezendő paraméter neve, a visszaszolgáltatott sztring pedig a paraméter értéke.

Ha futtatni akarunk egy appletet, ez be kell legyen ágyazva egy HTML oldalba, majd ezt az oldalt kell megjelentetni valamilyen böngészővel vagy az *appletviewer* segédprogrammal. Ha az applet egy eddig be nem töltött osztályra hivatkozik, akkor a böngésző azt először a helyi géppen keresi, majd ha nem találja meg, megpróbálja letölteni onnan, ahonna az applet érkezett.

A következő applet egy böngészőben jeleníti meg a „Helló Világ!" szöveget:

```
import java.awt.*;
import java.applet.Applet;

public class Hello extends Applet {
public void paint(Graphics g) {
g.drawRect(25, 2, 90, 25);
g.drawString("Helló Világ!", 50, 20);
}
}
```

Az appletet egy HTML oldalba kell beágyazni:

```
<HEAD>
<TITLE>Helló applet</TITLE>
</HEAD>
<BODY>
<APPLET CODE="Hello.class" WIDTH=200 HEIGHT=50>
Az APPLET nem működik.
```

```
</APPLET>
</BODY>
</HTML>
```

majd egy böngésző segítségével meg lehet tekinteni a HTML oldalt.

Hálózati alkalmazások fejlesztése

A Java hálózati programozási nyelv. Segítségével nagyon könnyen fejleszthetünk olyan alkalmazásokat, amelyeknek futási felülete több gépre, számítógépes hálózatokra is kiterjedhet. A Java ezeket az osztályokat és objektumokat a *java.net* csomagban tárolja. Hogy megértsük a csomag által szolgáltatott osztályokat, metódusokat, foglaljuk össze röviden mindazt, amit a hálózatokról tudnunk kell.

Számítógépes hálózaton olyan számítógépek összességét értjük, melyek valamilyen úton-módon képesek egymással kommunikálni, adatokat és információkat szolgáltatni egymásnak. A kommunikáció előfeltétele, hogy a számítógépek megértsék egymást, egy „közös nyelven beszéljenek". Ezért számos olyan szabályhalmaz – protokoll – alakult ki, amelyek képesek arra, hogy az adatokat, információkat minden számítógép számára érthetővé tegyék. Talán a legelterjedtebb ilyen protokoll a TCP/IP (Transmission Control Protocol / Internet Protocol) család, amely két ismert transzport-protokollt tartalmaz: a TCP-t és az UDP-t (User Datagram Protocol). A két protokoll közötti különbségeket talán legszemléletesebben a telefon és a postai levelezés összehasonlításával tudjuk érzékeltetni. A TCP a telefonvonalnak felelne meg és tulajdonképpen egy telefonbeszélgetést modellez: A hívó fél felemeli a kagylót, tárcsáz, ha a hivott fél is felemeli a kagylót, akkor létrejön egy állandó kapcsolat, minek folyamán adatokat és információkat lehet cserélni mindaddig, míg valamelyik a két fél közül meg nem szakítja a beszélgetést. Az UDP a postai levelezésnek felelne meg. Megírjuk a levelet, borítékba helyezzük, rájírjuk a címet majd bedobjuk a postaládába. A levél továbítása a postára van bizva. A címzett megkapja a levelet, majd szintén egy levéllel válaszol rá. Tehát a TCP állandó összeköttetést biztosít két kommunikációs végpont között, az UDP pedig egy összekötés-mentes protokoll.

Egy másik igen fontos fogalom a *port* fogalma. A port egy egész szám, amely egyértelmüen azonosít egy kommunikációs csatornát. Minden program legalább egy porton keresztül kommunikálhat egy másik programmal vagy önmagával. Nemcsak a portok, hanem a gépek is azonosítva vannak a hálózatban. Minden számítógép egy egyedi azonosítószámot, úgynevezett IP címet visel. Ez négy darab, egy-egy ponttal elválasztott 0 és 255 közötti tizes számrendszerben felírt szám (pl. 192.168.80.62). A port és az IP cím egyértelmüen meghatározza azt, hogy melyik gépen melyik programmal (milyen komminikációs csatornán keresztül) kommunikálok egy adott pillanatban.

Talán a legelterjedtebb alkalmazásszervezési modell a *kliens-szerver architektúra.* Az architektúra alapja az, hogy bizonyos programok (szerverek) futnak és kéréseket várnak, amelyeket kiszolgálnak. A kliensek rájelentkeznek a szerverre és közlik vele azokat a kéréseket, kérdéseket, amelyekre választ várnak. A szerverek értelmezik, feldolgozzák a kérést, megkeresik a választ és ezt visszaszolgáltatják a kliensnek. A kliensek a szerverrel kommunikációs végpontok és nyitott csatornák segítségével (*socket*) kommunikálnak.

A TCP alapú szerver feladatait ellátó kommunikációs végpontokat Javaban a *ServerSocket* osztály implementálja. A szerverrel kapcsolatot felépíteni szándékozó kliensek egy várakozási sorba kerülnek és a szerver mindig a legelső kliensel építi fel a kapcsolatot az *accept* metódus meghívásával, ami egy *Socket* osztályhoz tartozó objektumot ad vissza Ez azonosítja a kialakított kliens-szerver kommunikációs csatornát.

Az UDP alapú kapcsolatok kiépítésére a Java a *DatagramSocket* valamint a *DatagramPacket* osztályokat használja.

Nézzünk meg a következőkben egy TCP típusú kapcsolatot kialakító egyszerű Java kliens és szerver alaklamazást. A következő szerver egy, a paraméter által

megadott TCP porton várakozik, a rákapcsolódott klienstől egy pozitív számot vár, majd ennek a négyzetét szolgáltatja vissza. Ha a kapott szám –1, akkor kilép az alkalmazásból.

```
import java.io.*;
import java.net.*;

public class MyServer {

 public static void main(String[] args) {
  int port = 0;
  ServerSocket ss = null;
  Socket s = null;
  String kérés = null, válasz = null;
  int érték = 0;
  boolean kilép = false;

  try { port = Integer.parseInt(args[0]); }
   catch (NumberFormatException e) {
    System.out.println("Hibás argumentum!");
 }

  try {
   ss = new ServerSocket(port);
   while (!kilép) {
    s = ss.accept();
    BufferedReader input = new BufferedReader(
     new InputStreamReader(s.getInputStream()));
    PrintWriter output =
     new PrintWriter(s.getOutputStream());
    kérés = input.readLine();
    try {
     érték = Integer.parseInt(kérés);
     válasz = Integer.toString(érték*érték);
    } catch (NumberFormatException e) {
    válasz = "Hibás adat!"; }
    output.println(válasz); output.flush
    if (érték == -1) kilép = true;
    s.close();
   }

  } catch (IOException e) {
   System.err.println(e);
  }
  finally {
   try { ss.close(); } catch (IOException ex) {
    System.err.println("Nem lehet lezárni!");
    }
  }
 }
}
```

A *getInputStream()* illetve a *getOutputStream()* metódusok segítségével tudunk adatokat olvasni, illetve adatokat írni a kommunikációs végpontokra.

A következő kliens felveszi az argumentumban megadott IP és portú szerverrel a kapcsolatot, majd elküld neki egy, szintén argumentumként kapott egész számot, amit a szerver négyzetre emel, ha a szám –1, akkor ez a szerver lezárását vonja maga után.

```
import java.io.*;
import java.net.*;

public class MyClient {
 public static int ÉRTÉK;

 public static void main(String[] args) {
  int port = 0;
  Socket s = null;
  String válasz = null;

  try { port = Integer.parseInt(args[1]); }
   catch (NumberFormatException e) {
    System.out.println("Hibás argumentum!"); }
  try { ÉRTÉK = Integer.parseInt(args[2]); }
   catch (NumberFormatException e) {
    System.out.println("Hibás argumentum!"); }
  try {
   s = new Socket(args[0], port);
   BufferedReader input = new BufferedReader(
     new InputStreamReader(s.getInputStream()));
   PrintWriter output = new
     PrintWriter(s.getOutputStream());
   output.println(Integer.toString(ÉRTÉK));
   output.flush();
   válasz = input.readLine();
   System.out.println("A négyzet: " + valasz); }
    catch (IOException e) {
     System.err.println(e); }
 }
  finally
   try { s.close(); } catch (Exception ex) {
    System.err.println("Nem lehet lezárni!");
    }
  }
 }
}
```

Főbb Java csomagok

– *java.lang:* A programok futtatásához szükséges alapvető osztályokat definiálja. Ilyen osztályok az *Object*, a típusosztályok (*String, Integer, Boolean* stb.), *StringBuffer, Math, Exception* stb.

– *java.util:* Olyan osztályokat és interfészeket tartalmaz, amelyek segédeszközként hasznosak lehetnek alkalmazásaink fejlesztéséhez. Ilyen osztályok az *Enumeration, Observer, BitSet, Date, HashTable, Dictionary* stb.

– *java.io:* A bemenet és kimenet (input/output) kezelését támogató osztályokat tartalmazza: *OutputStream, Writer, Reader, InputStream, FileWriter, FileReader, CharArrayWriter* stb.

– *java.net:* A Java hálózat-elérését megvalósító osztályokat tartalmazza.

– *java.awt:* A Java programok garafikus felületét (awt – Abstract Window Toolkit) definiálja. Tulajdonképpen nem más mint előredefiniált grafikus objektumokat megvalósító osztályok gyűjteménye. A nevében szereplő abstract jelző arra utal, hogy a grafikus felület független marad a különböző operációs rendszerek által implementált grafikus rendszerektől.

– *java.applet:* Az appleteket megvalósító osztályokat tartalmazza.

Kovács Lehel