Programozási Ismeretek

Rendezések

Dr. Gergely Tamás
Dr. Ferenc Rudolf, Dr. Jász Judit, Dr. Kiss Ákos

Szegedi Tudományegyetem
Informatikai Intézet
Szoftverfejlesztés Tanszék

2024

(v0214)

Tartalom

Tartalom

A Java I/O rendszer

- Adatfolyam alapú megközelítés
- Java 1.0: `InputStream`/`OutputStream` osztályok
  - bájt-orientált I/O
- Java 1.1: `Reader`/`Writer` osztályok
  - karakter-orientált I/O *unicode* támogatással
  - bővíti az `InputStream`/`OutputStream` osztályhiearchiát is

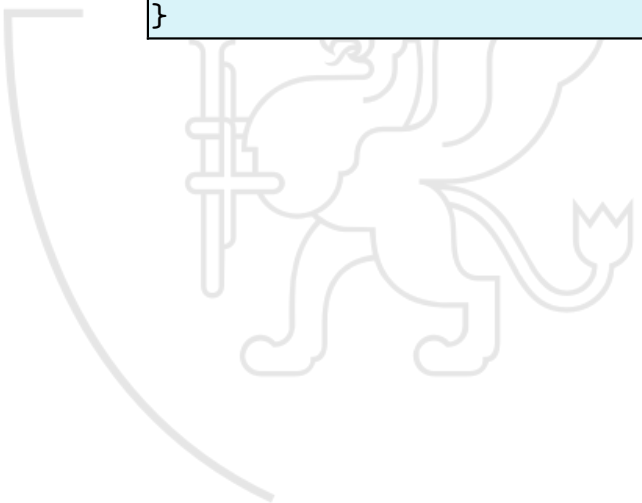
A `File` osztály

- Egy fájlt vagy könyvtárat reprezentál
- Az adott objektumon keresztül a kapcsolódó entitás megismerhető/módosítható

```java
import java.io.*;

public class DirList {
    public static void main(String[] args) {
        File path = new File(".");
        String[] list = path.list();
        for(int i = 0; i < list.length; i++)
            System.out.println(list[i]);
    }
}
```

```
MakeDirectories.class
DirList.class
DirList.java
MakeDirectories.java
```



A File osztály
(folyt.)

```java
import java.io.*;

public class MakeDirectories {
    public static void main(String[] args) {
        if (args.length < 1) {
            System.exit(1);
        }
        File f = new File(args[0]);
        System.out.println(
            "Abszolut utv.: "    + f.getAbsolutePath() +
            "\nNev:           " + f.getName() +
            "\nHossz:         " + f.length() +
            "\nModositva:     " + f.lastModified());
        if (f.isFile()) {
            System.out.println("fajl");
        } else if(f.isDirectory()) {
            System.out.println("konyvt.");
        }
        if (f.exists()) {
            System.out.println(f + " letezik, toroljuk");
            f.delete();
        } else {
            System.out.println("letrehozas... " + f);
            f.mkdirs();
        }
    }
}
```

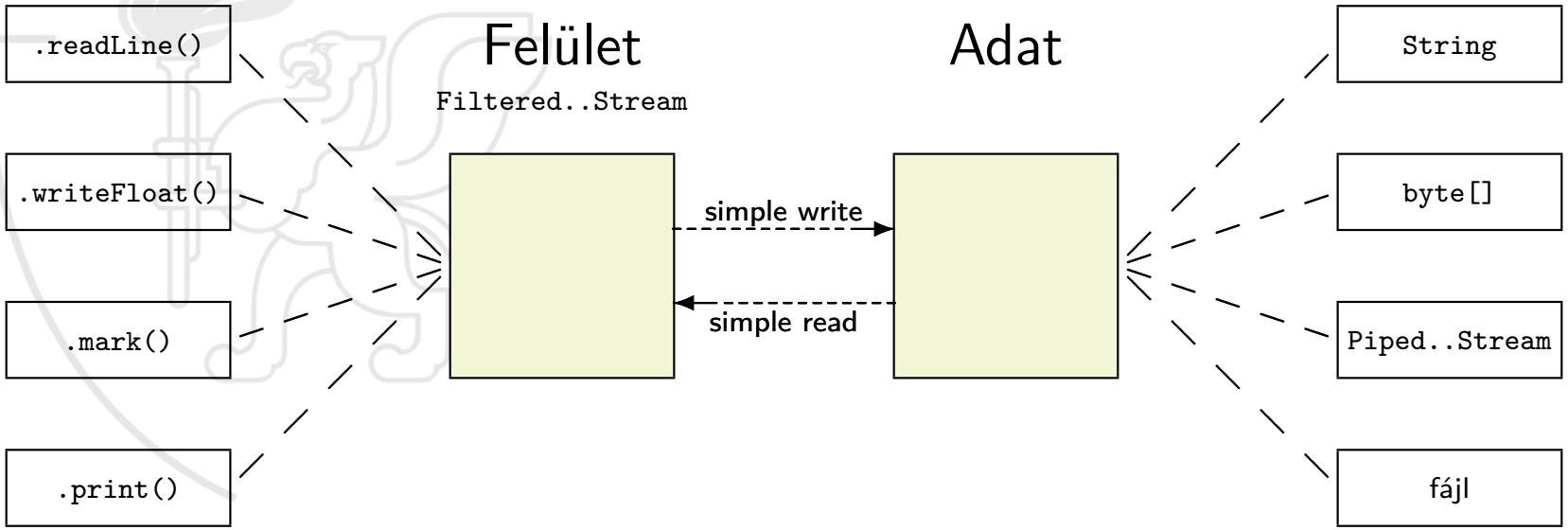
```
$ touch Hello
$ java MakeDirectories Hello
Abszolut utv.: /home/gertom/ProgIsm/io/Hello
Nev:           Hello
Hossz:         0
Modositva:     1548936042000
fajl
Hello letezik, toroljuk
```

```
$ java MakeDirectories Hello
Abszolut utv.: /home/gertom/ProgIsm/io/Hello
Nev:           Hello
Hossz:         0
Modositva:     0
letrehozas... Hello
```

```
$ java MakeDirectories Hello
Abszolut utv.: /home/gertom/ProgIsm/io/Hello
Nev:           Hello
Hossz:         4096
Modositva:     1548936052000
konyvt.
Hello letezik, toroljuk
```

InputStream és OutputStream

- Bájt alapú adatfolyam (*stream*)
  - Objektum, ami valamilyen adatforrást vagy adatnyelőt jelképez
- Két fő típusa van
  - `InputStream` – adatforrás folyam
  - `OutputStream` – adatnyelő folyam
- Feladat szerint
  - Felületet/funkciót ad – `Filtered...Stream`
  - Az adat helyét határozza meg



InputStream típusai

- `ByteArrayInputStream`
  - Az adatokat egy bájttömb tárolja, ebből olvasunk
- `StringBufferInputStream`
  - Az adatokat egy `String` objektum tárolja, ebből olvasunk
  - `@Deprecated`: nem javasolt a használata (`byte` – `char` konverzió)
- `FileInputStream`
  - Az adatokat egy fájlból olvassa be
- `PipedInputStream`
  - Egy `PipedOutputStream`-ből olvas, többszálú programok esetén a szálak közötti kommunikáció egyik eszköze
- `SequenceInputStream`
  - Más adatforrás folyamok sorozatát használja, ha az egyik „elfogy”, a következőből kezd olvasni
- `FilterInputStream`
  - Az adatfolyamot „dekoráló” osztályok őse

FilterInputStream típusai

- DataInputStream
  - Primitív adattípusok és String bináris olvasása
    - readByte()
    - readFloat()
    - readLine() (@Deprecated, byte – char konverzió)
    - ...
- BufferedInputStream
  - Pufferelt adatfolyam
    - mark()
    - reset()
- LineNumberInputStream (@Deprecated, byte – char konverzió)
  - Figyeli a sorszámot
    - getLineNumber()
- PushbackInputStream
  - A legutóbb olvasott karakterekt vissza lehet tenni az adatfolyamba
    - unread()

OutputStream típusai

- ByteArrayOutputStream
  - Az adatokat egy bájttömbbe írja
- FileOutputStream
  - Az adatokat egy fájlba írja
- PipedOutputStream
  - Egy PipedInputStream-be ír, többszálú programok esetén a szálak közötti kommunikáció egyik eszköze
- FilterOutputStream
  - Az adatfolyamot „dekoráló” osztályok őse
- ~~StringOutputStream~~
  - Nem lenne sok értelme, a String-et és StringBuffer-t egyszerű használni

FilterOutputStream típusai

- DataOutputStream
  - Primitív adattípusok bináris kiírása
    - writeByte()
    - writeFloat()
    - ...
- BufferedOutputStream
  - Pufferelt adatfolyam
    - flush()
- PrintStream
  - Formázott kimenet
    - format()
    - print()
    - println()

Reader és Writer

- Karakter alapú adatfolyamok
  - Objektum, ami valamilyen adatforrást vagy adatnyelőt jelképez
  - **unicode** támogatás
- Két fő típusa van
  - `Reader` – adatforrás folyam
  - `Writer` – adatnyelő folyam
- A legtöbb Java `InputStream`/`OutputStream` osztálynak (Java 1.0) megvan a megfelelő `Reader`/`Writer` párja
- Vannak konvertáló osztályok is
  - `InputStreamReader`
  - `OutputStreamWriter`

Megfeleltetés

Input

InputStream	–	Reader
FileInputStream	–	FileReader
StringBufferInputStream	–	StringReader
ByteArrayInputStream	–	CharArrayReader
PipedInputStream	–	PipedReader
FilterInputStream	–	FilterReader
BufferedInputStream	–	BufferedReader
DataInputStream	–	—
LineNumberInputStream	–	LineNumberReader
PushBackInputStream	–	PushBackReader

Megfeleltetés

Output

`OutputStream`	–	`Writer`
`FileOutputStream`	–	`FileWriter`
`ByteArrayOutputStream`	–	`CharArrayWriter`
`PipedInputStream`	–	`PipedReader`
`PipedOutputStream`	–	`PipedWriter`
`FilterOutputStream`	–	`FilterWriter`
`BufferedOutputStream`	–	`BufferedWriter`
`DataOutputStream`	–	—
`PrintStream`	–	`PrintWriter`

IO példa

Fájlból olvasás

```
import java.io.*;

public class IOStreamDemo {
    public static void main(String[] args)
            throws IOException {
        System.out.println("-- fajlbeolvasas soronkent --");
        BufferedReader brf =
            new BufferedReader(
                new FileReader("IOStreamDemo.java"));
        String s, f = new String();
        while((s = brf.readLine())!= null)
            f += s + "\n";
        brf.close();
```

```
-- fajlbeolvasas soronkent --
```

IO példa

Fájlból olvasás

```
        System.out.println("-- input memoriabol --");
        StringReader sr = new StringReader(f);
        int c;
        while ((c = sr.read()) != -1)
            System.out.print((char)c);
```

```
-- input memoriabol --
import java.io.*;

public class IOStreamDemo {
    public static void main(String[] args)
            throws IOException {
        System.out.println("-- fajlbeolvasas soronkent --");
        BufferedReader brf =
            new BufferedReader(
                new FileReader("IOStreamDemo.java"));
        String s, f = new String();
```

...

```
            new BufferedReader(new StringReader(f));
        PrintWriter pw =
            new PrintWriter(
                new BufferedWriter(
                    new FileWriter("IOStreamDemo.txt")));
        int lineCount = 1;
        while ((s = brs.readLine()) != null)
            pw.println(lineCount++ + ": " + s);
        pw.close();
    }
}
```

IO példa

Fájlból olvasás

```
        System.out.println("-- formatalt input memoriabol --");
        try {
            DataInputStream dis =
                new DataInputStream(
                    new ByteArrayInputStream(f.getBytes()));
            while (true)
                System.out.print((char)dis.readByte());
        } catch (EOFException e) {
            System.err.println("Adatfolyam vege");
        }
```

```
-- formatalt input memoriabol --
import java.io.*;

public class IOStreamDemo {
    public static void main(String[] args)
            throws IOException {
        System.out.println("-- fajlbeolvasas soronkent --");
        BufferedReader brf =
```

...

```
                new BufferedWriter(
                    new FileWriter("IOStreamDemo.txt")));
        int lineCount = 1;
        while ((s = brs.readLine()) != null)
            pw.println(lineCount++ + ": " + s);
        pw.close();
    }
}
```

```
Adatfolyam vege
```

IO példa

Fájlból olvasás

```
        System.out.println("-- fajl output --");
        BufferedReader brs =
            new BufferedReader(new StringReader(f));
        PrintWriter pw =
            new PrintWriter(
                new BufferedWriter(
                    new FileWriter("IOStreamDemo.txt")));
        int lineCount = 1;
        while ((s = brs.readLine()) != null)
            pw.println(lineCount++ + ": " + s);
        pw.close();
    }
}
```

```
-- fajl output --
```

```
1: import java.io.*;
2:
3: public class IOStreamDemo {
4:     public static void main(String[] args)
5:             throws IOException {
6:         System.out.println("-- fajlbeolvasas soronkent --");
7:         BufferedReader brf =
8:             new BufferedReader(
9:                 new FileReader("IOStreamDemo.java"));
10:         String s, f = new String();
                            ...
40:         while ((s = brs.readLine()) != null)
41:             pw.println(lineCount++ + ": " + s);
42:         pw.close();
43:     }
44: }
```

Standard I/O

- `System.in` (*standard input*): a program bemenete
  - `InputStream` típusú, így „dekorálni” kell
- `System.out` (*standard output*): a program normál kimenete
  - `PrintStream` típusú, így közvetlenül használható
- `System.err` (*standard error*): a program „hiba” kimenete
  - `PrintStream` típusú, így közvetlenül használható
- Mindhárom stream a program indításának pillanatától nyitott, és a lezárásukról is a futtató keretrendszer gondoskodik

```
import java.io.*;

public class Echo {
    public static void main(String[] args) throws IOException {
        BufferedReader in =
            new BufferedReader(
                new InputStreamReader(System.in));
        String s;
        // ures sorra kilep
        while ((s = in.readLine()).length() != 0)
            System.out.println(s);
    }
}
```

```
haliho
haliho
hehe
hehe
papagaj
papagaj
```

Tartalom

Kivételkezelés

- Java alapfilozófia: *Rosszul formált kód nem fog jól futni*
- Ideális lenne, ha minden hibát fordítási időben ki lehetne deríteni, de a valóságban ez nem lehetséges
- Korábbi nyelvekben (pl. C-ben) a hibakezelés inkább csak konvenció volt
  - tipikusan, a függvény visszaad egy hibakódot, vagy beállít egy jelzőt, amit a hívó fél kell(ene), hogy ellenőrizzen
  - ez általában elmarad: pl. ki ellenőrzi le, hogy a `scanf()` mit ad vissza?
- A megoldás: Nyelvi szinten beépíteni és megkövetelni a hibakezelést
- Nem a Java találmánya, már az 1960-as években ismerték operációs rendszer szinten
- A Java kivételkezelése a C++-on, az pedig az Ada-n alapul
- Még egy jelentős előny: „Megtisztul” a kód, mert különválik az „igazi” kód a hibakezeléstől

Kivételek

- *Kivételes feltétel*: egy probléma, ami meggátolja a program futást egy adott metódusban (blokkban)
- Egyszerű példa a `null` referencia: érdemes leellenőrizni és kivételt „dobni” (`throw`) ahelyett, hogy folytatnánk a programot

```
if (obj == null)
    throw new NullPointerException();
```

  - kivétel objektum jön létre a *heap*-en
  - az aktuális programvégrehajtás megáll és a kivétel objektum referenciája „kidobódik”
  - a *kivételkezelő mechanizmus* veszi át az irányítást és keres egy megfelelő *kivételkezelőt*, ami lekezeli a hibát és folytatja a program végrehajtását

Kivételek argumentumai

- Minden beépített kivételnek két konstruktora van: a default és ami egy `String`-et vár (ezt később fel lehet használni)

```
if (obj == null)
    throw new NullPointerException(
            "obj␣nincs␣inicializalva!");
```

- A kivételkezelés valójában egy alternatív „return” mechanizmus
  - a kivétel objektum van „visszaadva” (a metódus visszatérési típusa más!)
  - nem biztos, hogy csak egy szinttel tér vissza! (addig, amíg valaki nem kezeli le a kivételt)

Kivétel elkapása

- Ha valahol el lett dobva egy kivétel, akkor az feltételezi, hogy azt valahol máshol „elkapják”
- *Védett régió*: a kód olyan része, ami kivételeket hozhat létre és amit hibakezelő kód követ

```
try {
    // ‘‘normál kód’’, amiben kivétel keletkezhet
    // (védett régió)
} catch(ExceptionType1 e1) {
    // hibakezelő kód az e1 kivételre
} catch(ExceptionType2 e2) {
    // hibakezelő kód az e2 kivételre
    throw e2; // tovább is lehet dobni
} catch(Exception e) {
    // hibakezelő kód az összes (megmaradt) kivételre
}
```

Saját kivételek

- Saját kivételeket is lehet írni
- Származtatni kell valamelyik létező kivétel osztályból
  - pl. Exception (ős)osztályból

```
class SajatException extends Exception {
    public SajatException(String s) {
        super(s);
    }
}

public class KivetelPelda {
    public void f() throws SajatException {
        System.out.println("Dobunk egy SajatException-t f()-ből");
        throw new SajatException("f()-ből dobtak");
    }
    public static void main(String[] args) {
        KivetelPelda kp = new KivetelPelda();
        try {
            kp.f();
            System.out.println("Ezt a kivétellel átugorjuk");
        } catch(SajatException e) {
            System.err.println("Elkaptuk!");
            System.err.println(e);
        }
    }
}
```

```
Dobunk egy SajatException-t f()-ből
Elkaptuk!
SajatException: f()-ből dobtak
```

Kivétel specifikáció

- Java-ban a `throws` kulcsszóval meg kell adni, hogy egy metódus milyen kivételeket dobhat

```
void f() throws SimpleException, NullPointerException {
    // ...
}
```

  - ez része a metódus-deklarációnak
  - fordítási hibát jelent, ha nem tesszük meg

Java standard kivételek

- A `Throwable` ősosztálynak két leszármazottja van:
  - `Error`: általában nem kell vele foglalkozni (rendszerhibákat képvisel, akkor meg már régen rossz)
  - `Exception`: a Java programozó számára ez az „ősosztály”
- A kivételek leírása megtalálható a Java honlapján:
  - `https://docs.oracle.com/javase/6/docs/api/java/lang/Exception.html`
  - `https://docs.oracle.com/javase/9/docs/api/java/lang/Exception.html`

RuntimeException

- Java standard kivétel
  - a metódus specifikációban nem kell külön megadni (ahogy az `Object`-ből való leszármazást sem kell külön megadni az osztályoknál)
- Azon kivételek őse, amiket a virtuális gép normális működés közben is bárhol dobhat, pl.:
  - `NullPointerException`
  - `ClassCastException`
  - `IndexOutOfBoundsException`
  - stb.

Tartalom

A `Class` objektumok

- Speciális objektumtípus, amely tartalmazza a „szokásos” objektumok létrehozásához szükséges információt, vagyis a „szokásos” objektum osztályának „leírását”
- A programban minden osztályhoz létezik egy `Class` típusú objektum
  - valójában ez az objektum van eltárolva a `.class` fájlban
  - „Java-ban minden objektum”
- Futásidőben, *amikor szükség van* egy adott típusú objektumra, akkor a JVM betölti a megfelelő `.class` fájlból a típust leíró `Class` objektumot (ha még nincs betöltve)
  - egy Java program tehát általában nem egyszerre töltődik be a memóriába
  - a szerkesztési (*linking*) feladatokat a JVM futás közben végzi el

A `Class` objektum elérése

- Dinamikusan, osztálynév segítségével
  - `Class.forName("OsztalyNev")`
- A „`.class` syntax” segítségével
  - `OsztalyNev.class`
- Az `Object` ősosztály `getClass()` metódusa segítségével
  - `x.getClass()`, ahol `x` egy objektum

Példa reflection-re

`Class.forName()`

```
class Alakzat {
    static {
        System.out.println("Alakzat␣osztály␣betöltése");
    }
    public void rajzolj() {
        System.out.println(this + ".rajzolj()");
    }
}
```

```
class Haromszog extends Alakzat {
    static {
        System.out.println("Haromszog␣osztály␣betöltése");
    }
    public String toString() {
        return "Haromszog";
    }
}
```

```
public class AlakzatReflectionPelda1 {
    public static void main(String[] args) {
        new Alakzat();
        try {
            Class c = Class.forName("Haromszog");
        } catch (ClassNotFoundException e) {
            e.printStackTrace(System.err);
        }
    }
}
```

```
Alakzat osztály betöltése
Haromszog osztály betöltése
```

Példa reflection-re

Statikus instanceof

```
class Alakzat {
    static {
        System.out.println("Alakzat osztály betöltése");
    }
    public void rajzolj() {
        System.out.println(this + ".rajzolj()");
    }
}
```

```
class Haromszog extends Alakzat {
    static {
        System.out.println("Haromszog osztály betöltése");
    }
    public String toString() {
        return "Haromszog";
    }
}
```

```
import java.util.*;

public class AlakzatReflectionPelda2 {
    public static void main(String[] args) {
        List s = new ArrayList();
        s.add(new Kor());
        s.add(new Haromszog());
        s.add(new Negyzet());
        Iterator i = s.iterator();
        while (i.hasNext()) {
            Object o = i.next();
            if (o instanceof Haromszog)
                ((Haromszog)o).rajzolj();
        }
    }
}
```

```
Alakzat osztály betöltése
Kor osztály betöltése
Haromszog osztály betöltése
Negyzet osztály betöltése
Haromszog.rajzolj()
```

Példa reflection-re

„.class syntax” és dinamikus .isInstance()

```
class Alakzat {
    static {
        System.out.println("Alakzat␣osztály␣betöltése");
    }
    public void rajzolj() {
        System.out.println(this + ".rajzolj()");
    }
}
```

```
class Haromszog extends Alakzat {
    static {
        System.out.println("Haromszog␣osztály␣betöltése");
    }
    public String toString() {
        return "Haromszog";
    }
}
```

```
import java.util.*;

public class AlakzatReflectionPelda3 {
    public static void main(String[] args) {
        List s = new ArrayList();
        s.add(new Kor());
        s.add(new Haromszog());
        s.add(new Negyzet());
        Iterator i = s.iterator();
        while (i.hasNext()) {
            Object o = i.next();
            if (Haromszog.class.isInstance(o))
                ((Haromszog)o).rajzolj();
        }
    }
}
```

```
Alakzat osztály betöltése
Kor osztály betöltése
Haromszog osztály betöltése
Negyzet osztály betöltése
Haromszog.rajzolj()
```

Példa reflection-re

Dinamikus `Class.forName()` és `.isInstance()`

```java
import java.util.*;

public class AlakzatReflectionPelda4 {
    public static void main(String[] args) {
        List<Alakzat> s = new ArrayList<Alakzat>();
        s.add(new Kor());
        s.add(new Haromszog());
        s.add(new Negyzet());
        for (Alakzat a : s) {
            try {
                if (Class.forName(args[0]).isInstance(a))
                    a.rajzolj();
            } catch (ClassNotFoundException e) {
                e.printStackTrace(System.err);
            }
        }
    }
}
```

```
$ java AlakzatReflectionPelda4
Alakzat osztály betöltése
Kor osztály betöltése
Haromszog osztály betöltése
Negyzet osztály betöltése
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
        at AlakzatReflectionPelda4.main(AlakzatReflectionPelda4.java:12)
```

Példa reflection-re

Dinamikus `Class.forName()` és `.isInstance()`

```java
import java.util.*;

public class AlakzatReflectionPelda4 {
    public static void main(String[] args) {
        List<Alakzat> s = new ArrayList<Alakzat>();
        s.add(new Kor());
        s.add(new Haromszog());
        s.add(new Negyzet());
        for (Alakzat a : s) {
            try {
                if (Class.forName(args[0]).isInstance(a))
                    a.rajzolj();
            } catch (ClassNotFoundException e) {
                e.printStackTrace(System.err);
            }
        }
    }
}
```

```
$ java AlakzatReflectionPelda4 Paralelepipedon
Alakzat osztály betöltése
Kor osztály betöltése
Haromszog osztály betöltése
Negyzet osztály betöltése
```

```
java.lang.ClassNotFoundException: Paralelepipedon
        at java.net.URLClassLoader.findClass(URLClassLoader.java:382)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:424)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:349)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:357)
        at java.lang.Class.forName0(Native Method)
        at java.lang.Class.forName(Class.java:264)
        at AlakzatReflectionPelda4.main(AlakzatReflectionPelda4.java:12)
...
```

Példa reflection-re

Dinamikus `Class.forName()` és `.isInstance()`

```
import java.util.*;

public class AlakzatReflectionPelda4 {
    public static void main(String[] args) {
        List<Alakzat> s = new ArrayList<Alakzat>();
        s.add(new Kor());
        s.add(new Haromszog());
        s.add(new Negyzet());
        for (Alakzat a : s) {
            try {
                if (Class.forName(args[0]).isInstance(a))
                    a.rajzolj();
            } catch (ClassNotFoundException e) {
                e.printStackTrace(System.err);
            }
        }
    }
}
```

```
$ java AlakzatReflectionPelda4 Negyzet
Alakzat osztály betöltése
Kor osztály betöltése
Haromszog osztály betöltése
Negyzet osztály betöltése
Negyzet.rajzolj()
```

Egyéb `Class` metódusok

- `getName`
- `isInterface`
- `getMethods`
- `getConstructors`
- `getSuperclass`
- `getPackage`
- stb.

Példa reflection-re

Osztály felderítés

```
class Alakzat {
    static {
        System.out.println("Alakza
    }
    public void rajzolj() {
        System.out.println(this +
    }
}
```

```
class Kor extends Alakzat {
    static {
        System.out.println("
    }
    public String toString()
        return "Kor";
    }
}
```

```
import java.lang.reflect.*;

public class ReflectionPelda {
    public static void main(String[] args) {
        if (args.length != 1)
            System.exit(0);
        try {
            Class clazz = Class.forName(args[0]);
            System.out.println(clazz.getName());
            System.out.println(clazz.isInterface());
            Constructor[] c = clazz.getConstructors();
            Method[]      m = clazz.getMethods();
            for (int i = 0; i < m.length; ++i)
                System.out.println(m[i]);
            for (int i = 0; i < c.length; ++i)
                System.out.println(c[i]);
        } catch(ClassNotFoundException e) {
            System.err.println(e);
        }
    }
}
```

```
$ java ReflectionPelda Kor
Alakzat osztály betöltése
Kor osztály betöltése
Kor
false
public java.lang.String Kor.toString()
public void Alakzat.rajzolj()
...
public boolean java.lang.Object.equals(java.lang.Object)
public native int java.lang.Object.hashCode()
public final native java.lang.Class java.lang.Object.getClass()
...
```