4. Gyakorlat

legendi@inf.elte.hu

2010. március 1.

Kiegészítés

String#split() Strint → String[], regexpek alapján (ld. FAQ).

```
for (String act : "a b c".split(" ")) {
    System.out.println(act);
}
```

Tömb értékű kifejezések Inicializálásnál elég a $\{ elem_1, elem_2, ... \}$ forma. DE! Mindenhol máshol, ahol tömb típusú kifejezést szeretnénk leírni (*new*), a fordítási hibák kiszűrése miatt a típust is meg kell jelölnünk. Pl.:

```
public static int sum(int[] arr) { ... }

public static void main(String[] args) {
    int result = sum( new int[] {1, 2, 3} );
}
```

Tömb segédosztály java.util.Arrays, hasznos pl. a toString(), binarySearch(), fill(), etc.
Részletesen: `http://java.sun.com/javase/6/docs/api/java/util/Arrays.html`

Konvenciók Igyekezzünk figyelni rá.

```
public class MyClass {
    public static final int MAX = 10;
    private boolean String someString;

    public void myFunction() {
        ...
    }
```

```
    public boolean isSomething() {
        ...
    }
}
```

Overriding, overloading Ha a szignatúra ugyanaz (függvény neve + paraméterek száma és típusa), akkor felüldefiniálás, egyébként túlterhelés. Más nem. Visszatérési értékre lehet specializálni, exception lista nem lényeges. Példák:

```
interface A {
    int f(int a, int b);
    int g() throws Exception;
    A h();
}

class B implements A {
    @Override
    int f(int a, int b) { return 0; } // Feluldefinialas

    void f(int a, int b) {}; // Hibas: visszateresi ertek nem kompatibilis
    int f(int a) { return f(a, 0); }    // Tulterheles (parameterek szama)
    int f(Integer a, Integer b) { ... } // Tulterheles (parameterek tipusa)

    @Override
    int g() { return 0; }     // OK: Exception lista nem szamit

    @Override
    B   h() { return null; }; // OK: Visszateresi ertekre specializal
}
```

Enumok

Motiváció: "enum pattern" kiváltása:

```
public static final int OS_WINDOWS = 0;
public static final int OS_LINUX   = 1;
public static final int OS_BSD     = 2;
public static final int OS_VMS     = 3;
```

Problémák:

Típusbiztonság hiánya Ez csak egy sima int, így mindenhol, ahol ilyen típusra számítunk, egy egyszerű intet is írhatunk véletlenül

Névtér hiánya Csak a prefix-szel kerülhetjük el a névütközéseket

Törékenység Új konstans beszúrása problémás, azonos értékek (copy-paste)

Érthetőség Kiírásnál csak az értékeket látjuk, nem túl informatívak (milyen típus, mit reprezentál)

Java 5.0 óta, egyszerű definíció:

```
// public, protected, private, static lehet csak
public static enum Os { Windows, Linux, BSD, VMS };

public static void main(final String[] args) {
    System.out.println("Elements:");
    for (final Os act : Os.values()) {
        System.out.println(act.ordinal() + " ->" + act.name());
    }

    // String -> Os
    final Os os = Os.valueOf(System.console().readLine());

    switch (os) {
    case Windows:
        System.out.println("Windows");
        break;

    case Linux:
    case BSD:
        System.out.println("Unix");
        break;

    default:
        System.out.println("Other");
        break;
    }
}
```

Java csel: viselkedés, plusz adattag adható az enumokhoz. Pl.:

```
public enum Guitar {
    Electronic (6),
    Acoustic (5),
    Bass (4); // elem definiciok vege

    // tulajdonsag leirasa (adattagok, fuggvenyek)
```

```
    private final int strings;

    private Guitar(final int strings) {
        this.strings = strings;
    }

    public int getNumberOfStrings() {
        return strings;
    }
}
```

További okosság: constant-specific függvények:

```
public enum Operation {
  PLUS   { double eval(double x, double y) { return x + y; } },
  MINUS  { double eval(double x, double y) { return x - y; } },
  TIMES  { double eval(double x, double y) { return x * y; } },
  DIVIDE { double eval(double x, double y) { return x / y; } };

  abstract double eval(double x, double y);
}
```

Igen ritkán van rá szükség, de akkor hasznos, hogy van. További okosság: java.util.EnumSet, java.util.EnumMap.

Részletesen: `http://java.sun.com/docs/books/tutorial/java/javaOO/enum.html`

Feladat

Készítsünk egy minimális BKV nyilvántartó programot! Az egyes járművekhez (Busz, HEV, Troli, Metro, Villamos) tároljuk el a különböző járatszámokat egy String tömbben, valamint egy logikai változóban tároljuk le, hogy az adott járművek Budapest közigazgatási határán kívülre is közlekednek-e (egyedül a HEV-ek ilyenek). Parancssori argumentumként kapunk egy járatszámot, majd írjuk ki, hogy a megadott jármű közlekedik-e Budapesten kívül!

Valósítsuk meg a feladatot enumokkal!

Névtelen osztályok

Osztályt a példányosító kifejezéshez függesztett osztálytörzzsel kiterjesztjük:

```
Type type = new Type() { /* +definitions */ }
```

Adattagokat, függvényeket vehetünk fel, implementálhatunk definiálhatunk felül. A kifejezés statikus típusa Type lesz, dinamikus típusa Type névtelen leszármazottja lesz. Konstruktort nem tartalmazhatnak (példányinicializátort azonban igen!). Pl.:

```
Alkalmazott alkalmazott = new Alkalmazott() {
    // Ha adattagot veszunk fel, az normal esetben csak itt lathato
    @Override
    public String toString() {
        return "(anon) " + super.toString();
    }
};

Vector vector = new Vector() {{ add(1); add(2); }};
```

GUI

Abstract Windowing Toolkit (nehézsúlyú) vagy Swing (pehelysúlyú komponensek). Előbbi oprendszer szintű, utóbbi Javaban íródott, független az oprendszertől. Egyéb ablakozó keretrendszer is használható (pl. SWT, Qt, etc.).

Használható komponensek (beviteli, vezérlő, megjelenítő komponensek): `http://java.sun.com/docs/books/tutorial/ui/features/compWin.html`

Ill. *c:/Program Files/Java/jdk1.6/demo/jfc* alatt SwingSet2 és SwingSet3 példaalkalmazások.

Példakód

```
import java.awt.*;

public class AWTTest {
    Frame frame = new Frame("AWTTest");
    Label label = new Label("Hello AWT!");

    public AWTTest() {
        frame.add(label);
        frame.setSize(200, 100);
        frame.setLocation(300, 300);
        frame.setVisible(true);
    }

    public static void main(String[] args) {
```

```
        AWTTest test = new AWTTest();
    }
}
```

Életciklus

1. Felület felépítése. Egyszerű példaalkalmazás

```
import java.awt.*;

public class AWTTest {
    Frame frame = new Frame("GUI Test");
    TextField textField = new TextField("10");
    Button button = new Button("Ok");
    Label label = new Label("=");

    public AWTTest() {
        frame.setLayout(new FlowLayout());

        frame.add(textField);
        frame.add(button);
        frame.add(label);

        frame.setVisible(true);
        frame.setSize(200, 100);
        frame.setLocation(300, 300);
    }

    public static void main(String[] args) {
        AWTTest test = new AWTTest();
    }
}
```

2. Használata (eseményfigyelők)

```
        button.addActionListener( new ActionListener() {
            @Override
            public void actionPerformed(final ActionEvent ae) {
                try {
                    final int N = Integer.parseInt(textField.getText());
                    int res = 1;
                    for (int i=2; i<=N; ++i) res *= i;
                    label.setText("= " + res);
```

```
                } catch (final NumberFormatException e) {
                    label.setText(e.getMessage());
                }

            }
        });
```

3. Bezárása

```
frame.addWindowListener( new WindowAdapter() {
    @Override
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
});
```

Adapterek, Listenerek

XXXListener interfész, ha nem kell az összes esemény, használható az XXXAdapter absztrakt osztály is (üres implementációkat tartalmaz). Pl. WindowListener, WindowAdapter. Hívási lánc alapján.

Fontosabb Listenerek:

- ActionListener: menü, gombnyomás, Enter egy TextFielden

```
public void actionPerformed(ActionEvent e) { ... }s
```

- KeyListener: Billentyű leütése

```
public void keyTyped(KeyEvent e)    { ... }
public void keyPressed(KeyEvent e)  { ... }
public void keyReleased(KeyEvent e) { ... }
```

- MouseListener: Egérlenyomás (van MouseMotionListener, MouseWheelListener is)

```
public void mousePressed(MouseEvent e)  { ... }
public void mouseReleased(MouseEvent e) { ... }
public void mouseEntered(MouseEvent e)  { ... }
public void mouseExited(MouseEvent e)   { ... }
public void mouseClicked(MouseEvent e)  { ... }
```

- WindowListener: ablak eseményeinek kezelése

```
public void windowActivated(WindowEvent e)   { ... }
public void windowClosed(WindowEvent e)      { ... }
public void windowClosing(WindowEvent e)     { ... }
public void windowDeactivated(WindowEvent e) { ... }
public void windowDeiconified(WindowEvent e) { ... }
public void windowIconified(WindowEvent e)   { ... }
public void windowOpened(WindowEvent e)      { ... }
```

- Stb., lásd referenciát.

Elrendezési stratégiák

Részletesen `http://java.sun.com/docs/books/tutorial/uiswing/layout/visual.html`

- BorderLayout
- BoxLayout
- CardLayout
- FlowLayout
- GridBagLayout
- GridLayout
- GroupLayout/SpringLayout

Komponensek

Konténer komponensek (Container), legfelső szinten top-level containerek (applet, frame vagy dialog). Minden grafikus osztály közös őse: (J)Component. A java.awt.* csomagban.

- Panel (konténer)
- Label
- Button
- RadioButton
- TextArea, TextField
- etc.

Swing

Az osztályok nevének közös prefixe egy J betű. AWT-s eseménykezelés itt is megvan. A javax.swing.* csomagban (Java extended). További előnyök:

- Független az oprendszertől
- Szabad forma
- Átlátszó felületek
- Egyszerűen módosítható megjelenés

Menük

frame.setMenuBar(), MenuBar -> Menu, Menu -> Menu, MenuItem. Van szeparátor is.

```
JMenuBar menuBar = new JMenuBar();

JMenu fileMenu = new JMenu("File");
menuBar.add(fileMenu);

JMenuItem exitMenu = new JMenuItem("Exit");
fileMenu.add(exitMenu);
exitMenu.addActionListener( ... )

frame.setJMenuBar(menuBar);
```

Üzenetablakok

JOptionPane.showXXXDialog(), pl.:

```
JOptionPane.showMessageDialog(null, // ki az ose, ha van, akkor modalis
    "alert",                        // uzenet
    "alert",                        // title
    JOptionPane.ERROR_MESSAGE       // tipus
);
```

Részletesen: `http://java.sun.com/javase/6/docs/api/javax/swing/JOptionPane.html`

Scrollozható komponensek

```
frame.add( new JScrollPane(textArea) );
```

Linkek

1. `http://community.java.net/javadesktop`
2. `http://java.sun.com/javase/technologies/desktop/articles.jsp`
3. `http://java.sun.com/docs/books/tutorial/ui/index.html`
4. `http://java.sun.com/docs/books/tutorial/uiswing/index.html`

+/- Feladatok

A feladatokat a `legendi@inf.elte.hu` címre küldjétek, **szombat éjfélig**! A subject a következőképp nézzen ki:

```
csop<csoportszám>_gyak<gyakorlat száma>_<EHA-kód>_<a megoldott feladatok száma>
```

Például:

```
csop1_gyak4_LERIAAT_1  (csak az első feladattal)
csop1_gyak4_LERIAAT_12 (mindkét feladattal)
```

Mellékelni csak a Java forrásfájlokat mellékeljétek (semmiképp ne teljes Eclipse/NetBeans projecteket), esetleg rarolva, zipelve, és egy levelet küldjetek (több levél esetén az utolsó mellékleteit értékelem)! Ajánlott az első feladatot NetBeans, a másikat Eclipse alatt készíteni. Szokni kell a környezetet, debuggolást. További fontos kritériumok:

- Ékezetes karaktereket **ne** használjatok! Főleg azonosítók esetében ne! A Java ugyan ezt megengedi, ugyanakkor a különböző környezetekbe való konvertáláskor ($latin2 \leftrightarrow UTF-8 \leftrightarrow Cp1250$) összetörnek a karakterek! Az ilyen forrásokat fordítani, következésképp értékelni sem tudom.

1. Feladat

Készítsünk egy egyszerű szövegszerkesztő alkalmazást! A programból a menüből lehessen szöveges állományt megnyitni, elmenteni, valamint kilépni.

Ezen kívül az alábbi funkciók közül legalább hármat valósíts meg!

1. Használj tooltipeket, amikben HTML formázott szöveg van (`setToolTipText("<html> ...")`)!
2. Állíts be az alkalmazásnak egy tetszőleges ikont (`setIconImage(...)`)!

3. Készíts egy *Help* menüt, amibe teszel egy egyszerű `About...` menüelemet. Ha erre rákattint a felhasználó, írja ki a program verziószámát, valamint a készítő nevét (JOptionPane.)!

4. Állíts be gyorsgombokat és mnemonicokat az egyes menüelemekhez (setAccelerator(...), setMnemonic(...))!

5. Az alkalmazás egy `JFileChooser` segítségével válassza ki, hogy milyen fájlt szeretnél megnyitni!

6. Használj egy JTabbedPane panelt, amivel egyszerre több állomány is megnyitható!

2. Feladat

Készítsünk egy egyszerű, 3x3-as kirakós játékot! A fő panel tartalmazzon 9 nyomógombot, amelyeken 1..8-ig számok szerepelnek, a kilencediken pedig egy "*" karakter. A számokat kezdetben véletlenszerűen helyezzük el (Math.random(), de az is elég, ha van egy beégetett összekevert kombináció). Ha egy számot tartalmazó nyomógombra kattintunk, akkor ha az szomszédos a * karakterrel, cseréljük fel a gombok szövegét! A cél, hogy a számokat sorfolytonosan kirakjuk, és a * karakter a jobb asó sarokban legyen. Ha ezt sikerült elérnie a felhasználónak, adjunk egy gratuláló üzenetet!