Programozás III

SAJÁT ESEMÉNY +
OSZTÁLYTÍPUSOK

„SAJÁT” ESEMÉNY DEFINIÁLÁSA

Előfordulhat, hogy olyan eseményt kell kezelnünk, amelyre nincs kész metódus.

Pl.: Amíg el nem készül egy rajz, addig ne lehessen megnyomni egy gombot.

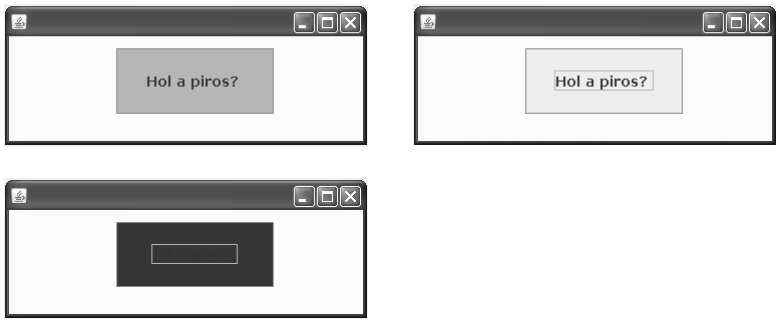
Az eseményfigyelők őse az EventListener interface.

Erre alapozva saját interface-t is írhatunk:

SajatEsemeny extends EventListener{…},

de most egy egyszerűbb megoldást nézünk.

„SAJÁT” ESEMÉNY DEFINIÁLÁSA



A gombnyomás hatására véletlenszerűen változik a gomb színe, és ha piros, akkor megváltozik a felirata is.

„SAJÁT” ESEMÉNY DEFINIÁLÁSA

A legegyszerűbb, ha magába a gombnyomás esemény kezelésébe építjük be a feliratváltást is, de most (kissé erőltetetten ugyan) egy másik eseménnyel oldjuk meg. (Ez vezeti elő a saját eseményt.)

```
szinGomb.addChangeListener(new javax.swing.event.ChangeListener() {
    public void stateChanged(javax.swing.event.ChangeEvent evt) {
        szinGombStateChanged(evt);
    }
});
```

„SAJÁT” ESEMÉNY DEFINIÁLÁSA

```
private void szinGombStateChanged(javax.swing.event.ChangeEvent e
    if (szinGomb.getBackground().equals(szinek.get(0))) {
        szinGomb.setText("Itt a piros!");
    } else {
        szinGomb.setText("Hol a piros? ");
    }
}
```

(A szinek lista elemei: Color.red, Color.blue, stb.)

Ez az esemény még nem „saját”, generálható, de olyan értelemben általános, hogy bármilyen változást képes figyelni.

„SAJÁT” ESEMÉNY DEFINIÁLÁSA

Alternatív megoldás: saját gomb definiálása.

```
public class SajatGomb extends JButton implements
                                        ActionListener,
                                        ChangeListener {

  private List<Color> szinek = new ArrayList<>();

    public SajatGomb() {
        szinek.add(Color.red);
        szinek.add(Color.blue);
          ...
        this.setText("Hol a piros?");
    }
```

„SAJÁT” ESEMÉNY DEFINIÁLÁSA

```
@Override
public void actionPerformed(ActionEvent e) {
    Color szin = szinek.get((int)(Math.random()*szinek.size()));
    this.setBackground(szin);

}

@Override
public void stateChanged(ChangeEvent e) {
      if(this.getBackground().equals(szinek.get(0))) {
          this.setText("Itt a piros!");
      }
      else this.setText("Hol a piros? ");
}
```

„SAJÁT” ESEMÉNY DEFINIÁLÁSA

```
public class PeldaPanel extends javax.swing.JPanel {

    public PeldaPanel() {
        initComponents();
        sajatGomb.addActionListener(sajatGomb);
        sajatGomb.addChangeListener((ChangeListener) sajatGomb);
    }
```

„SAJÁT” ESEMÉNY DEFINIÁLÁSA

Másik példa: Szöveggenerálás

Egy külön szál betűket generál, majd jelzi a panelnek, ha történt benne változás.

„SAJÁT” ESEMÉNY DEFINIÁLÁSA

A feladat megoldása:

A szálakat felkészítjük rá, hogy valaki figyelhesse, hogy történt-e bennük változás, azaz írunk bennük egy-egy

addChangeListener()

metódust – ennek paraméterében megadható, hogy ki figyelheti őket.

```
private ChangeListener chListener;

public void addChangeListener(ChangeListener chListener){
    this.chListener = chListener;
}
```

„SAJÁT” ESEMÉNY DEFINIÁLÁSA

A run() metódusban meghívjuk az előbb definiált ChangeListener változó stateChange() metódusát. Ennek paramétere egy ChangeEvent típusú változó.

```
private ChangeEvent chEvent = new ChangeEvent(this);

        chListener.stateChanged(chEvent);
```

„SAJÁT” ESEMÉNY DEFINIÁLÁSA

Már csak az van hátra, hogy a panelt felkészítsük az események figyelésére.

Ehhez a panelnek implementálnia kell a ChangeListener interfészt, meg kell hívni a szálak add…Listener() metódusát – evvel beállítjuk, hogy a panel figyelni fogja a szálak változását.

Majd meg kell írnunk azt az (öröklött) metódust, amelyet a változáskor kell végrehajtania (stateChanged()).

E szerint, ha az esemény forrása a betűszál, akkor elkérjük tőle a generált betűt, és kirakjuk a szövegmezőre. Ha a szóközszál a forrás, akkor szóközt írunk.

„SAJÁT” ESEMÉNY DEFINIÁLÁSA

A kódrészletek:

```
public class SzovegPanel extends javax.swing.JPanel implements ChangeListener{

                betuSzal.addChangeListener(this);

               szokozSzal.addChangeListener(this);

   @Override
   public void stateChanged(ChangeEvent ce) {
       if(ce.getSource().equals(betuSzal)){
           szovegMezo.append(String.valueOf(betuSzal.getBetu()));
       }
       if(ce.getSource().equals(szokozSzal)){
           szovegMezo.append(String.valueOf(szokozSzal.getKoz()));
       }
   }
```

„SAJÁT” ESEMÉNY DEFINIÁLÁSA

Általánosabb megoldási lehetőség: a saját eseményt felfűzzük egy eseményfigyelő láncra. Evvel lehetővé tesszük, hogy többen is figyelhessék az osztályváltozás eseményét.

```
private EventListenerList changeListeners;

public CsigaPanel() {
    this.changeListeners = new EventListenerList();
    initComponents();
}
```

„SAJÁT” ESEMÉNY DEFINIÁLÁSA

```
    public void addChangeListener(ChangeListener l) {
        this.changeListeners.add(ChangeListener.class, l);
    }

    public void removeCahngeListener(ChangeListener l) {
        this.changeListeners.remove(ChangeListener.class, l);
    }

private void notifyChanges() {
    ChangeEvent evt = new ChangeEvent(this);
    for (ChangeListener l:this.changeListeners.getListeners(ChangeListener.class))
        l.stateChanged(evt);
}
```

A végén: `this.notifyChanges();`

„SAJÁT” ESEMÉNY DEFINIÁLÁSA

Továbbiak:

ld. HELP,

vagy pl.:

http://docs.oracle.com/javase/tutorial/uiswing/events/

http://docs.oracle.com/javase/tutorial/uiswing/events/changelistener.html