**awt:**A felület megjelenítéséért nem a java platform,hanem a futtató os ablakozó rendszere a felelős.

Ennek hátránya,hogy ez a csomag nem tartalmaz annyi grafikus komponenst ,mint más grafikus csomagok.

**Swing:**Az awt kiküszöböléseire létrehozott grafikus könyvtár.Ezek javaban íródtak,önmaguk gondoskodnak a megjelenésükről,független az os ablakozó rendszerétől.

**AWT**

**Környezeti változó:** A **környezeti változók** (angolul *environment variables*) a [számítógép](https://hu.wikipedia.org/wiki/Sz%C3%A1m%C3%ADt%C3%B3g%C3%A9p) működésére nézve fontos információkat tartalmazó szöveges (sztring) szimbólumok, amelyeket az operációs rendszer tárol (A környezeti változók nem tévesztendőek össze a programok belső változóival). A környezeti változók mindig a programokon kívül léteznek, de a programok számára elérhetőek: a programok „közelében”, környezetében fordulnak elő. Környezeti [változó](https://hu.wikipedia.org/wiki/V%C3%A1ltoz%C3%B3) például a PATH, a PROMPT, a TEMP, stb. A környezeti változók értékeit a [programok](https://hu.wikipedia.org/wiki/Sz%C3%A1m%C3%ADt%C3%B3g%C3%A9pes_program) lekérdezhetik és felhasználhatják működésükhöz.

**Esemény objektum:**Az eseménytípusokat az esemény objektum getId() metódusával kérhetjük le, ez pedig arra jó,hogy megkülönböztethető egymástól minden eseménytípus(rákattitnott,felé vitte az egeret,beleírt .stb... ezek mind különböző típusok), illetve ezzel biztosítja a kompatibilitást a nem objektu m-orientált módon megvalósított ablakozó rendszerekkel.

A RESERVED\_ID\_MAX konstans érték mutatja azt a számot ,aminél csak nagyonn értéket adhatunk a saját eseményeinknek ID értékként.

-A MouseEvent getButton() metódusa visszaadja a lenyomott gombot reprezentáló konstans értékét(BUTTON1,BUTTON2,stb...).

A getClickCount visszaadja azt az értéket,hogy hányszor kattintottunk gyorsan egymás után a komponensre, ez az érték természetesen egérmozgatás esemény bekövetkeztekor természetesen nulla(Ha kattintunk és várunk 1 secet és újra kattintunk akkor ennek az értéke 1 lesz, viszont ha kurva sokat kattintunk egyszerre akkor kurva sok lesz az értéke).

Minden egéresemény bekövetkeztekor eltárolódik az egérkurzor aktuális pozíciója(getX,getY metódussal lekérdezhetők).

A megjegyzett koordinátákta a translatePoint metódussal meg is lehet változtatni.

MouseWheelEvent objektum egy egérgörgetést reprezentál.

Ez a MouseEvent leszármazottja , és a görgető esemény figyeléséhez implementálni kell a MouseWheelListener interfészt.

Ugyebár az egérgomb kattintás és egérvalamihez húzást a MouseListener interfésszel intézzük.

MouseMotionListener:Egérmozgatás eseményeknek megfelelő metódusokat definiálja(egérkurzor elmozdulása MouseMoved és lenyomott egérgombbal történő mozgatés MouseDragged).

Ezen egéreseményfigyelő interfészek a MouseEvent objektumot kapják paraméterül.

**public** **class** Own **implements** MouseListener,MouseWheelListener{

**public** Own(Container terulet){

terulet.setLayout( **new** java.awt.BorderLayout() );

JButton gomb = **new** JButton("Tes");

terulet.add(gomb);

gomb.addMouseListener(**this**);

gomb.addMouseWheelListener(**this**);

}

**public** **static** **void** build(Container terulet){

**new** Own(terulet);

}

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

Frame ablak = **new** Frame("Saját teszt...");

*build*(ablak);

ablak.pack(); //ablak méretezése

ablak.setVisible(**true**);

}

@Override

**public** **void** mouseClicked(MouseEvent arg0) {

}

@Override

**public** **void** mouseEntered(MouseEvent arg0) {

}

@Override

**public** **void** mouseExited(MouseEvent arg0) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** mousePressed(MouseEvent arg0) {

}

@Override

**public** **void** mouseReleased(MouseEvent arg0) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** mouseWheelMoved(MouseWheelEvent arg0) {

System.***out***.println(arg0.getWheelRotation());

}

}

-Billentyűzet események:Minden billentyűzettel kapcsolatos eseményt a KeyEvent objektum reprezentál.

Háromféle eseményt különböztet meg:

¤ KEY\_PRESSED Billentyű lenyom

¤ KEY\_RELEASED Billentyű felenged

¤ KEY\_TYPED Az előző kettő kombinációja,tehát az előző kettő az alacsonyabb szintű esemény.

Billentyű lenyomás és felengedés esetén van a KeyEvent objektumnak egy getKeyCode metódusa, ami egy egész szám, és ez os által generált kód a billentyűhöz.

Nagyon sok konstansa van:

KeyEvent.VK\_S

KeyEvent.VK\_A

KeyEvent.VK\_D

KeyEvent.VK\_0 ezek minden billentyűre vannak.

**public** **void** keyPressed(KeyEvent e) {

System.***out***.println(e);

**if** (e.getKeyCode()==KeyEvent.***VK\_W***) {

camera.move(0.05f,2);

queenCoord[2] += 0.05f;

} **else** **if** (e.getKeyCode()==KeyEvent.***VK\_S***) {

camera.move(-0.05f,2);

queenCoord[2] -= 0.05f;

**Események feldolgozása:**Minden eseményt egy objektum reprezentál.

Minden eseményvezérelt program rendelkezik egy eseménysorral,ami a keletkezett eseményobjektumokat tárolja ,lehetővé téve azok sorban való végrehajtását.

Ezt az eseménysort az EventQueue osztály valósítja meg , melyből minden program egyetlen példánnyal rendelkezik csak.A Toolkit osztály getSystemEventQueue statikus metódusával nyerhetjük ki ezt a referenciát.

Az eseménysort legalább két programszál is használhatja egyidejűleg:

Az egyik az ablakozó rendszer által generált szál, és innen kerülnek be az eseménysorba az események a postEvent metódus segíttségével.

A másik szál az eseményfeldolgozó programszál,ami a kettő közül magasabb prioritású.Ez a getNextEvent metódus segítségével sorra veszi az eseményeket.Ha nincs feldolgozásra váró esemény ,akkor a getNextEvent addig blokkolja a szálat,amíg nem jön esemény objektum.

Ezek a programszálak nem démon jellegűek, ezért lehet az,hogy végigmegy a progi a main-en és akkor még nem feltétlenül áll le a progi.

**Események figyelése:**Egy eseményt egy olyan osztály példánya kezelhet mely implementálja az adott …Listener interfészt(KeyEvent kezelő osztálynak implementálnia kell a KeyListener interfészt).

Az interface annyi metódust tartalmaz, ahány esemény következtében keletkezik az adott esemény objektum PL. a KeyListener interfész 3 metódust definiál(keyTyped,keyPressed,keyRelease),mivel ezen három esemény hatására keletkezhet KeyEvent.

Feiratkozás az eseményre összefoglalva pl. menüelemek, nyomógombok esetén előfordulhat ActionEvent ha rá kattintanak. Tehát létre kell hozni egy ActionListener interface-t implementáló osztályt, majd ennek egy példányát be kell jegyezni eseménykezelőnek a gomb vagy menüelem addActionListener metódusának meghívásával paraméterként megadva az eseménykezelő objektumot. Az ActionListener egy void actionPerformed(ActionEvent) metódust deklarál, az objektumnak ez metódusa fog meghívódni az esemény előfordulásakor.

Add…Listener

Remove…Listener

**Eseménykezelés:**

Az eseménykezelő osztályt tervezhetem úgy, hogy - egy komponens eseményére reagáló: Sok eseménykezelő osztályunk lesz, de egynek a kódja rövidebb, gyorsabb - több komponens eseményére reagáló: Kevesebb eseménykezelő osztályunk lesz, de egynek a kódja hosszabb, hiszen ilyenkor az eseménykezelő metóduson belül el kell ágazni az esemény forrása szerint. Az esemény paraméter tartalmaz információkat az esemény forrásáról. Kétféle információ alapján ágazhatunk el:

A) az esemény forrás referenciája alapján

B) az esemény forrás akció parancsa alapján

Egy eseménykezelő, több forrás

A) Lekérdezhető az esemény forrásának referenciája a getSource() metódussal (Ne felejtsük, hogy ilyenkor az eseménykezelő osztályban elérhetőnek kell lennie a komponens referenciájának, hogy legyen mivel összehasonlítani! (btnIr2, btnIr3))

public void actionPerformed(ActionEvent arg0) {

if (arg0.getSource() == btnIr2)

System.out.println("Gomb kettő");

else if (arg0.getSource() == btnIr3)

System.out.println("Gomb három");

…

B) Lekérdezhető a forrás komponens akció parancsa. Néhány komponensfajtának lehet megadni egy akció parancsot (ez egy egyszerű szöveg)

public void actionPerformed(ActionEvent arg0) {

if (arg0.getActionCommand().equals("Iro2"))

System.out.println("Gomb kettő");

else if (arg0.getActionCommand().equals("Iro3"))

System.out.println("Gomb három");

…

gomb = **new** JButton("Tes");

masikgomb = **new** JButton("Katt");

terulet.add(masikgomb);

terulet.add(gomb);

gomb.addMouseListener(**this**);

masikgomb.addMouseListener(**this**);

@Override

**public** **void** mouseClicked(MouseEvent arg0) {

**if**(arg0.getSource() == **this**.gomb){

System.***out***.println("Elso");

}**else** **if**(arg0.getSource() == **this**.masikgomb){

System.***out***.println("Masik");

}

}

A definiáló osztályban kezelem az eseményeket,és a két gombhoz nem írtam külön osztályt,hanem egyetlen kezelő osztályban elágaztattam.

Az eseménykezelő osztályt tervezhetem úgy, hogy

A) A csúcs konténert definiáló osztály legyen,

B) külső, önálló osztály legyen,

C) beágyazott tagosztály legyen, nincs gond a private adatok elérésével

D) névtelen beágyazott osztály, csak ha egy komponens eseményeire reagál

E) névtelen beágyazott osztály, amely meghívja a beágyazó egy metódusát

A GUI szerkesztők általában a D) vagy E) séma szerint generálják le a kódot, de átállítható.

Az eseménykezelő osztály fajtái

1. A csúcs konténert definiáló osztály egyben az eseménykezelő osztály is. Amellett, hogy a csúcskonténert definiáló osztályban pakolom össze a komponens fát, még meg kell valósítsa a megfelelő eseménykezelő interface-t(-eket) is. - Kevesebb osztály, de az osztályom kódja hosszabb, nehezebben áttekinthető.

public class EsemenyPelda extends JFrame implements ActionListener {

… @Override public void actionPerformed(ActionEvent arg0) { … }

Ebben az esetben amikor egy objektumhoz figyelőt rakunk ,akkor a this-t kell neki átadni értékül,hisz maga ez az osztály fogja kezelni az eseményt.

Gomb.addActionListener(this);

1. Egy külső, önálló osztályt készítünk az esemény kezeléshez. Az osztály meg kell valósítsa a megfelelő eseménykezelő interface-t is. - Sok osztály (sok file), de az egyes eseménykezelő osztályok kódja kizárólag az eseménykezelő metódust tartalmazza. - Nehézkes ebből az osztályból elérni a grafikus felületünket tartalmazó osztály változóit, pedig gyakran van rá szükség (pl. a grafikus felület módosítása miatt). - Ritkán használt megoldás, hiszen felesleges publikus, más program által is használható osztályokat készíteni. Helyette inkább beágyazott osztályokat célszerű használni.
2. Egy beágyazott tagosztály az eseménykezelő. - Kevesebb forrás file mintha önálló osztályokat használnánk és lehet private elérésű is. A beágyazó osztály forrása viszont hosszabb. - Statikus beágyazott tagosztály esetén (úgy mint külső osztálynál) gondot jelenthet a módosítani kívánt adatok elérése, hiszen ezek példányszintűek általában. Megoldás lehet átadni a referenciáját. - Példányszintű beágyazott tagosztálynál a beágyazó osztály adatai könnyen elérhetők

Példányszintű beágyazott tagosztály példa részlet.

public class EsemenyPelda {

public EsemenyPelda() {

… this.btnSzinez3.addActionListener(new Szinez3Kez());

… }

…

private class Szinez3Kez implements ActionListener{

public void actionPerformed(ActionEvent e) {

EsemenyPelda.this.btnIr2.setBackground( Color.YELLOW);

}

}

}

1. Névtelen beágyazott osztály az eseménykezelő. Mivel minden egyes esemény lekezelés egyedi így felesleges elnevezett osztályt létrehozni. - Hasonlít a Példányszintű beágyazott tagosztályokhoz, de a kód közelebb van a felhasználáshoz.

gomb.addMouseListener(**new** MouseListener(){

@Override

**public** **void** mouseClicked(MouseEvent e) {

System.***out***.println("Névtelen");

}

@Override

**public** **void** mouseEntered(MouseEvent e) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** mouseExited(MouseEvent e) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** mousePressed(MouseEvent e) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** mouseReleased(MouseEvent e) {

// **TODO** Auto-generated method stub

}

});

1. Névtelen beágyazott osztályból meghívva a beágyazó egy metódusát. Ez egyesíti az A) és D) megoldás előnyeit.

**Egérkurzor:**Ugyebár az egérkurzor lehet homokóra,kéz stb, és ezt valahogy állítgatnunk is kell.

Az egérkurzor alakját a Cursor osztály képviseli.Ennek rengeteg konstansa lehet melyek,különböző kurzor kinézeteket reprezentálnak.

getName() metódussal kinyerhetjük a jelenlegi kurzor típusát.

Az alábbi megoldással a jelenlegi keretünkhöz a várakozó kurzort használjuk.Azért a this.setCursor()-t használjuk,mivel a jelenlegi osztály a JFrame ablak leszármazottja ,így nála kell bejegeyezni a kurzort.

Cursor cur = **new** Cursor(Cursor.***WAIT\_CURSOR***);

**this**.setCursor(cur);

**Betűtípus:**A betűtípusokat a Font osztály képviseli.

A konstruktor első paramétere egy string ami jelképezi, a második a Font osztály konstansai,melyek betűtípusokat reprezentálnak,ha több sílust szabnánk rá akkor bitenkénti vagy művelettel megoldható,a harmadik paraméter pedig a betűméret.

Látható,hogy itt a jtextfieldhez is hozzáadtuk külön a stílust,tehát a szövegdobozba írt szövegek félkövér dőltek lesznek.

Font style = **new** Font("alma", Font.***ITALIC*** | Font.***BOLD***, 25);

**this**.setFont(style);

JTextField js = **new** JTextField(15);

js.setFont(style);

**Színkezelés:**Színek reprezentálására a Color osztályt hozták létre.

Color szin = new Color(1,1,0.5f,1);

A konstruktorban megadhatjuk rgba komponenseivel a színt.

Másik inicializálási lehetőség:

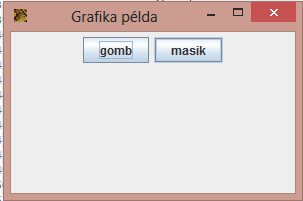
Color szin = Color.BLUE; A Color osztály konstansai.

**Képkezelés:**

Képet általában fileból vagy hálózatról szedhetünk elő.A Toolkit osztály createImage() metódusával létrehozhatunk Image objektumot.Létezik getImage metódusa is,de azt nem érdemes használni,mert a már betöltött képeket tárolja a memórában és ez plusz hely.A createImage mindig új objektumot hoz létre.

Image kep = Toolkit.*getDefaultToolkit*().createImage("./bg1.png");

**this**.setIconImage(kep);



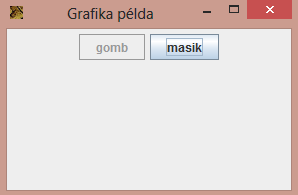
Látható,hogy a fejlécben nem a java jel van,hanem az általam beállított kép.

**Grafikus komponensek:**Minden komponenst egy objektum reprezentál.

A komponens vizuális tulajdonságának változásakor ComponentEvent esemény generálódik,ezt a ComponentListener figyelők kezelik.

Az egérkurzor alakját a komponensen belül a get/setCursor metódussal lehet állítani vagy lekérdezni.

gomb.setEnabled(**false**);



A gombra az enablet falsera állítjuk.

Az alábbi példában egy gombra állítunk kattintás kezelőt.Az esemény objektumtól elkérjük a gomb referenciáját,majd arra meghívjuk a háttérszinének beállítását, és a betűszínének beállítását.Tehát a komponenseknek ezen metódusaival lehet színeket állítani.

@Override

**public** **void** mouseClicked(MouseEvent arg0) {

arg0.getComponent().setBackground(Color.***GREEN***);

arg0.getComponent().setForeground(Color.***CYAN***);

}

System.out.println(arg0.getComponent().getLocationOnScreen());

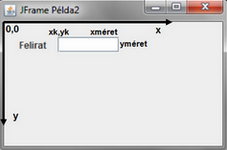
A fenti metódus egy Point objektumot ad vissza ami tartalmaz egy x és egy y koordinátát, ami a kopmonens bal felső sarkának képernyő koordinátái.

Set/getSize metódusokkal a komponens szélessége és magassága állítható és kérdezhető le.

Locale objektum is adható a komponenshez a nyelvfüggetlen program miatt.set/getLocale

A komponens láthatóságát a set/isVisible metódusok segíttségével szabályozhatjuk.A komponensek létrehozásukkor autómatikusan látszódva inicializálódnak.

Az y koordináta lefelé menve növekszik:



**Konténerek:**