Szakdolgozat

Készítette:
Tuska Gábor

Debrecen
2007

Debreceni Egyetem,
Informatikai Kar

Grafikus Felhasználói Felület fejlesztése Java környezetben

Témavezető:
Simon Gyula

Készítette:
Tuska Gábor

Debrecen
2007

Grafikus Felhasználói Felület fejlesztése Java környezetben

Tartalomjegyzék

1. Bevezető

Napjainkban az informatika az élet minden terültén egyre nagyobb szerepet tölt be. A társadalmunk is már olyan szintű bonyolultságot ért el, hogy a hozzájuk kapcsolódó adatok tömegét szinte lehetetlen már papíron tárolni és kezelni, így ezeket kénytelen vagyunk, digitális módon tárolni, amely könnyebb elérhetőséget, áttekinthetőséget, összegezve könnyebb kezelhetőség, különböző technikákkal pedig nagyobb biztonságot jelent. Emellett megjelentek különböző nagy vállalatok melyeknek a hatékony működéséhez elengedhetetlen egy jól működő információs rendszer. Ezek következményeként jelenleg óriási méretű információs rendszerek léteznek, amelyeknek elvárása, hogy nagy mennyiségű ember férhessen hozzá. A nagyobb cégeknél óhatatlanul szinte minden alkalmazott kapcsolatba kerül egy ilyen rendszerrel, és használnia kell tudni a rendszer rávonatkozó részét, de mivel nem csak informatikusokból áll egy-egy ilyen cég biztosítani kell egy megfelelő interfészt az informatikához nem értő alkalmazottai számára is.

Egy informatikához nem értő ember, ha leül a számítógép elé és egy parancssoros interfésszel szembesül nyilvánvalóan képtelen lesz azt kezelni és beletanulnia sem könnyű, hiszen meg kell ismerni, mind a parancsokat, mind meg kell tanulnia, hogy hová tegye a rendszer különböző reakcióit. Ezentúl az alkalmazottak ilyen rendszerekre való betanítása is erőforrás igényes művelet. Ebből is érzékelhető, hogy a fejlesztők kénytelenek voltak, valami könnyen tanulható interfészt teremteni a felhasználó és a rendszer közötti interakcióra. Ennek a nyomásnak a hatására jöttek létre az úgynevezett grafikus felhasználói felületek a GUI-k. Ezek lényege, hogy az eddigi parancssoros, UI-val szemben egy grafikus felületet nyújtunk a felhasználónak, amelynél az eddigiekkel szemben nem szükséges különböző parancsok ismerete, hanem egy lehetőség szerint érthetően, valamilyen bevett referencia alapján felépített hierarchikus rendszerben navigálva kommunikálunk a felhasználóval. A hatékony navigálás érdekében pedig a használtabb funkciókhoz különböző gyors billentyűket rendelünk. Ezeknek a GUI-knak pedig a felhasználó barát adatbevitel érdekében egy grafikus segéd input eszközt is kitaláltak az egeret.

Mivel a GUI tényleg hatékony kommunikációt biztosított a hozzá nem értők számára is, ezzel nagymértékben vagy nullára csökkentette a betanítás költségeit, sőt lehetővé tette azt is, hogy a cégek ilyen felületekkel forduljanak az ügyfeleikhez is. Ennek egyértelmű

következménye az lett, hogy napjainkra a felhasználókkal való kommunikációt az információs rendszerek mind egy grafikus felületen keresztül oldják meg.

Ez az egyik oka a témám választásának. A másik, amit ebben a részben meg kéne indokolnom az, hogy miért Java?

Napjainkban információs rendszerek fejlesztése során nagyon felerősödött az objektum-orientált világ. Ennek az egyik fő oka az OO szemléletben történő fejlesztés a különböző információs rendszereknél jelentősen kényelmesebb és kézenfekvőbb, a különböző ilyen fejlesztések során fellépő problémákra. Ennek az objektum-orientált világnak az egyik legelterjedtebb képviselője a Java nyelvi környezet. A Java sikerességének fontos pontja az az óriási mennyiségű, előre megírt és letesztelt osztály, melyek a java nyelvre épülve a Java környezetet képezik. Ez a környezet többek között hatékony eszközöket biztosít GUI fejlesztéshez is.

Az egyik ilyen eszközrendszer az Abstract Window Toolkit (AWT), amely grafikus felületek, felépítéséhez biztosít különféle elemeket (gombok, menük…). Az AWT által nyújtott osztály hierarchia lehetővé tette a grafikus felületek építését, de volt egy komoly hibája. Ez pedig az volt, hogy a grafikus felület elemeinek kirajzolásához az operációs rendszer eszközeit használta és ezzel a Java egyik alapelvét, a platform függetlenséget szegte meg, hiszen ettől a pillanattól ugyanaz a GUI nem ugyanúgy nézett ki két különböző platformon. Emellett még az is felróható volt az AWT-nek, hogy bizonyos igényelt bonyolultabb elemmel nem rendelkezett. Ennek az orvoslására jött létre az AWT-re épülő Swing. A Swing szemben az AWT-vel már saját maga rajzolta ki a komponenseket és habár ezzel némileg lassult a kirajzolás, de a számos előny mellett ez eltörpült. Az egyik ilyen előny a már korábban említett platform függetlenség megvalósulása és egy a saját kirajzolással megjelenő lehetőség a Pluggable Look and Feel, ami megjelenés testreszabását tette lehetővé.

Ezen szakdolgozaton belül a Swing nyújtotta lehetőségek egy részét fogom részletezni, néhány helyen pedig, ahol szükséges az AWT egyes eszközei is elő fognak kerülni, mivel a Swing AWT-re épülése miatt az AWT megkerülhetetlen.

2. A JFC GUI fejlesztésre nyújtott lehetőségei

A Java-ban történő GUI fejlesztésre a JFC (Java Foundation Classes) kínál eszközrendszert. A JFC fő részei az AWT, a Swing, az Accesibility, a Java 2D és a Drag and Drop. Ezek közül grafikus felület fejlesztésére közvetlenül az AWT (Abstract Window Toolkit) és a Swing alkalmas.

A Java-ban először, csak az AWT volt jelen, mint GUI fejlesztési eszköz, de egyrészről bizonyos a Java-val szemben támasztott követelményeknek nem felelt meg, másrészről pedig elég csekély eszközrendszerrel rendelkezett. A legfontosabb Java követelmény, amelynek nem tett eleget az AWT, az a platform függetlenség volt. Ennek fő oka abban volt keresendő, hogy az AWT komponenseinek kirajzolását az operációs rendszerrel végeztette el, így a megjelenés platformfüggővé vált, ami az előbb említett elvnek ellent mond. A másik fentebb említett probléma pedig a csekély eszközrendszer volt. Erről annyit írnék, hogy a grafikus felületek széleskörű elterjedésével párhuzamosan számos gyakran használt grafikus elemet definiálhattunk, melyek közül az AWT-nek csak a legalapvetőbbekre volt előre megírt komponense (pl.: button - nyomógomb, list - lista, scrollbar - görgetősáv…) és így számos gyakran szükséges elem hiányzott az eszközkészletéből (pl.: filechooser - tallozó, table - táblázat…).

Az AWT hibáinak és hiányosságainak pótlására jött létre a Swing A Swing első és fő feladata a platformfüggetlenség megvalósítása volt. Ennek érdekében egy új önálló és a Java nyújtotta lehetőségeket felhasználó megjelenítéssel rendelkező komponens hierarchiát készítettek a Swing-nek. Ezek az új komponensek a régi AWT-s komponensekre épültek az osztályhierarchia alapján öröklődés segítségével. Ennek következménye, hogy maga a Swing is az AWT-re építkezik. Az új eszközrendszer behozásával lehetőség nyílt az AWT eszközkészletéhez képest történő bővítésekre is, de ezt és a fent leírtak részletei lentebb következnek. A Swing-nek még fontos tulajdonsága, hogy a JavaBeans specifikációjának megfelelően fejlesztették, azaz a komponensei rendelkeznek, illetve rendelkezniük kell a JavaBean-ek öt tulajdonságával (introspection, properties, customization, communication, persistency).

3. Könnyű és nehézsúlyú komponensek

Az AWT a grafikus felület felépítéséhez úgynevezett komponenseket használ. Ezek a komponensek a Java osztályhierarchiájában a Component osztály leszármazottaiként jönnek létre (1.1-es ábra), amely az AWT része és az Object osztály közvetlen leszármazottja. Az AWT összes többi komponense ebből származik. A Swing, mint azt már korábban is említettem az AWT-re épül. Ennek az osztályhierarchia szintű megjelenése az, mely szerint a

Container (a Component leszármazottja) leszármazottjaként megjelenik a JComponent, mely néhány kivétellel a Swing komponenseinek szülőosztálya. Kivételt csak a Window és leszármazottainak megfelelői jelentenek, melyek közvetlen a megfelelőjükből származnak.

Ahogy az ábrán is rögtön látszik, nem egyszerűen csak saját megjelenítéssel rendelkező megfelelőket, hoztak létre, de rengeteg új, az alkalmazás fejlesztés során hasznos eszköz is megjelent a Swing-ben (JFileChooser, JProgressBar…). De ezen kívül megjelent még egy elég komoly eltérés is, amelyre mindenképp érdemes odafigyelni. Ez pedig a könnyű- (lightweight) és a nehézsúlyú (heavyweight) komponensek, amely a platform-független megjelenítéssel van közvetlen kapcsolatban.

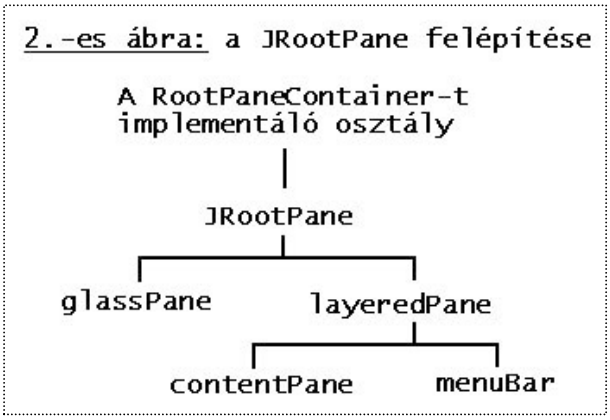
A Java hivatalosan azt mondja, hogy az AWT komponensek nehezek, míg a Swing egyik nagy újdonsága, hogy könnyű komponensek készletét kínálja, melyek a nehézsúlyú társaikkal szemben maximális valószínűséggel ugyanúgy néznek ki minden platformon. A tényleges különbség viszont abban mutatkozik, hogy a nehéz komponensekkel szemben a könnyűek képesek a saját z tengely szerinti helyzetüket is változtatni, míg erre a nehezek nem képesek. Emellett viszont megjelent egy mellékhatása is ennek, mégpedig, hogy a heavyweight komponensek elfedik az összes leightweight komponenst. Gyakorlatban erre az a szabály áll, hogy lehetőség szerint ne használjunk egyszerre nehéz és könnyű komponenst, illetve soha ne helyezzünk lightweight hordozóba heavywight komponenst. Ezek közül az első szabályra egy van egy nyilvánvaló kivétel, mégpedig a Swing által szolgáltatott négy nehéz komponens, melyek úgynevezett top-level hordozók és a grafikus felület alapját képező hordozók. A Swingben történő GUI fejlesztés során ezekbe már csak könnyű komponenseket helyezzünk a későbbiek során. Ez a négy nehéz Swing komponens, mint az az 1.1-es ábra (függelékek) alapján is sejthető, a JDialog, a JFrame, a JWindow és az ábrán nem szereplő JApplet. Így már az is érthető, hogy ezek miért nem a JComponent leszármazottai, hiszen a JComponent már biztosít bizonyos lightweight eljárásokat és ezek öröklődnek.

4. A top-level Swing komponensek és működésük

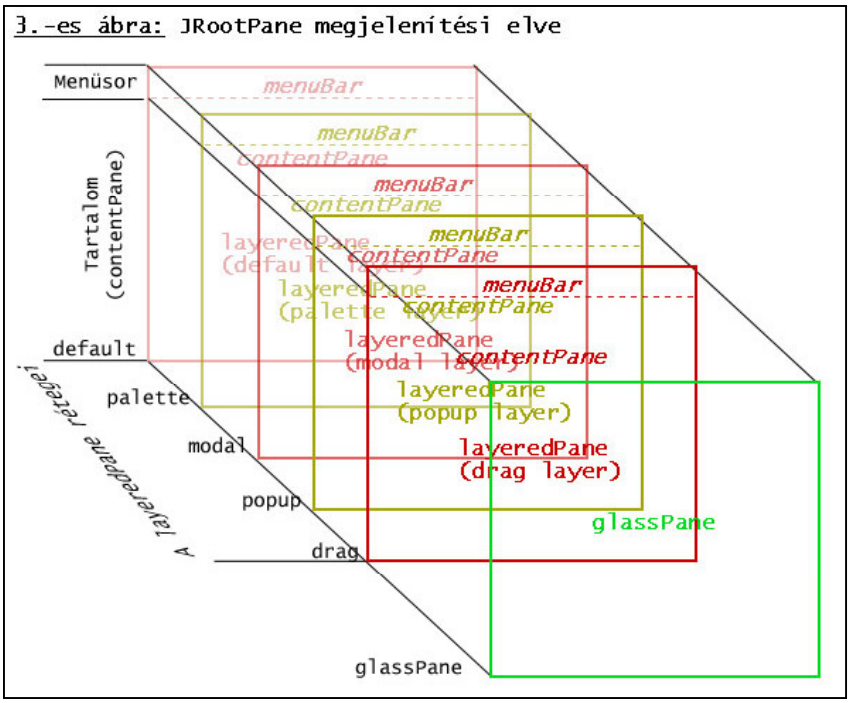
Négy úgynevezett nehézsúlyú Swing komponens van, melyből itt csak hármat tárgyalok, kihagyva a JApplet-et. Ezek a komponensek az úgynevezett top-level komponensek. Mint azt a nevük is jelzi feladatuk abban áll, hogy erre építkezzenek a felhasználók, ezek képezzenek egy „alapot" a fejlesztéshez. Ezeknek a komponenseknek kivétel nélkül container, azaz hordozó szerepük van, a fejlesztés során ezekbe helyezzük a különböző könnyűsúlyú komponenst, illetve megjegyzendő, hogy nem szabad, ezeket a top-level komponenseket egymásba helyezni, erre vannak könnyűsúlyú hordozó eszközök is a Swing-ben, illetve a későbbiek során az okát is ismertettem. Az előbbiek úgy is összefoglalhatóak, hogy a GUI építés során egy hierarchiát építünk különböző Swing komponensekből és ennek a hierarchiának a gyökere egy top-level hordozó.

A top-level komponensek megértéséhez ebben a fejezetben először is két Swing-beli interfészt ismertettek. Ezt egyrészről azért teszem, mert ezek az interfészek szinte teljesen meghatározzák ezeket a komponenseket, másrészről pedig szinte csak a top-level komponensek tartalmazzák őket.

4.1 A RootPaneContainer interfész



A top-level komponensek egyik fő tulajdonsága, hogy kivétel nélkül implementálják a RootPaneContainer interfészt. Ez az interfész a javax.swing része és az öt implementáló hordozóknak egyetlen közvetlen gyermekük (hordozott kompens), mégpedig egy JRootPane osztályú objektum. A JRootPane egy könnyűsúlyú komponens, melynek a feladata a RootPaneContainer-t implementáló osztály objektumának általunk megadott gyermekeit tartalmazása. Ez az oka, annak, hogy top-level hordozók nem építhetőek egymásba, hiszen annak ellenére, hogy látszólag nehézsúlyú komponenseket építünk egymásba, a valóságban könnyűsúlyú komponensek is beépülnek az egymásba ágyazódási láncba. Visszatérve a



JRootPane-re, mint az a 2-es ábrán is jól látszik, két fő része van: a glassPane és a layeredPane. A glassPane alapértelmezve egy láthatatlan JPanel, mely az egész JRootPane tetején található és a feladata az egérmozgás kezelése. A glassPane a JRootPane teljes területét kitölti. A glassPane bármely komponenssel helyetesíthető. Ezentúl még a glassPane alkalmas a RootPaneContainer tartalma előtt más tartalmat megjelenítenni (3.-as ábra). A layeredPane hasonlóan a glassPane-hez a JRootPane teljes területét lefedi, de a layeredPane-nek a JLayeredPane osztály leszármazottjának kell lennie. A layeredPane tartalmazza a contentPane-t, ami a RootPaneContainer gyermekeit tartalmazza és a menuBar-t amely a RootPaneContainer menüje. Továbbá a layeredPane lehetőséget add a gyermekek rétegelt kezelésére is, öt különböző elfedési szinten álló réteget definiálva (default, palette, modal, popup, drag). A menuBar null vagy egy JMenuBar példány, míg a contentPane alapértelmezve JPanel, egyébként pedig tetszőleges komponens. Tehát top-level komponensek felépítése a RootPaneContainer interfész által megszabott felépítés és az ezeknek a tartalmát befolyásoló eljárások delegálódnak a JRootPane-nek, illetve annak a megfelelő elemét módosítják. Megjegyzendő, hogy a négy top-level kompensen kívűl még JInternalFrame is implementálja ezt az interfészt. Továbbá a contentPane alapértelmezett JPanel-e a JPanel alapértelmezettével szemben nem FlowLayout-ot, hanem BorderLayout-ot használ LayoutManager-ként.

4.2 A WindowConstant intefész

A másik az előbbinél kevésbé fontos interfész a WindowConstants interfész. Ez az interfész mindössze négy konstanst tartalmaz (DO_NOTHING_ON_CLOSE,

HIDE_ON_CLOSE, DISPOSE_ON_CLOSE, EXIT_ON_CLOSE). A WindowCosntants feladata pedig, hogy lehetőséget add a setDefaultCloseOperation(int) metódus alkalmazására, mely az alapértelmezett műveletet határozza meg, ha a hordozó bezárására kattintunk. Az interfészt a JFrame, a JDialog és a nem top-level JInternalFrame implementálja. Fontos, hogy alapértelmezve a JInternalFrame-t leszámítva az alapértelmezett bezárási művelet az implementálóknál a HIDE_ON_CLOSE. A JInternalFrame-nél ez az EXIT_ON_CLOSE.

4.3 Egyéb interfészek top-level komponensek által használt, de nem csak hozzájuk kötődő interfészek

A fenti két interfészen kívül a top-level komponensek még kivétel nélkül implementálják az ImageObserver, a MenuContainer, a Serializable és az Accesible interfészeket. Ezek közül a MenuContainer nyilvánvalóan adódik a RootPaneContainer-ből és a JRootPane-nek a menuBar részéből. Az Accesible-nek a JFC Accesibility részével való kapcsolat a feladata, és a legtöbb Swing komponensre jellemző.A Serializable-t és az ImageObserver-t a top-level komponensek mellett pedig a JComponent is implememtálja és így minden Swing komponens is. A Serializable intefész a JavaBeans által elvárt perzisztencia feltételét hívatott teljesítenni. Az ImageObserver a komponensek képi megjelenítésének frissítését végzi.

4.4 JFrame

A javax.swing része. Az AWT-beli Frame közvetlen leszármazottja és Swing-beli megfelelője. A fent említett interfészek mindegyikét implementálja. Használatát tekintve egy fő hordozó szokott lenni. A JFrame rendelkezik saját címsorral, ablakvezérlő gombokkal és szegéllyel, továbbá átméretezhető és bezárható (van alapértelmezett bezárási művelete is).

4.5 JWindow

A javax.swing része. Az AWT-beli Window közvetlen leszármazottja és Swing-beli megfelelője. A fent említett interfészek közül a WindowConstants-ot leszámítva mind implementálja. A JWindow egy olyan az asztalon bárhol megjeleníthető hordozó, amely nem

rendelkezeik saját címsorral, ablakvezérlő gombokkal és szegéllyel a JFrame-mel szemben. Továbbá nem átméretezhető vagy bezárható. Természeténél fogva ideiglenes megjelenítésre használjuk (bejelentkező logo, stb…). Ezenkívűl a JWindow több képernyős rendszerek esetén különböző megjelenítőkön is definiálható.

4.6 JDialog

A javax.swing része. Az AWT-beli Dialog közvetlen leszármazottja és Swing-beli megfelelője. Az összes fent említett interfészt implemetálja. Tulajdonságai megegyeznek a JFrame-ben leírtakkal, azzal a kivétellel, hogy a JDialog lehet modális is, illetve megadható neki egy tulajdonos Frame, vagy Dialog. A modális ablak nem engedi, hogy más ablak aktív legyen, illetve az aktuális végrehajtást is blokkolja. A modalitás a setModal(boolean) eljárással állítható be. A JDialog-ot a felhasználóval való párbeszédeknél alkalmazzák (pl.: megerősítesek).

5. JComponent és az örökölt tulajdonságai, viselkedése

A Swing-ben a JComponent minden könnyűsúlyú komponens őse és a GUI építés gyakorlatilag egy top-level komponensbe történő hierarchikus elhelyezése a könnyűsúlyú komponenseknek. Ennek következményeként célszerű megvitatni a JComponent definiált és örökölt metódusait és tulajdonságait, hiszen ezek tovább öröklődnek minden könnyűsúlyú komponensben és használhatóságukat is nagymértékben meghatározza. Itt megemlítendő, hogy mivel a JComponent nem a Component közvetlen leszármazottja, hanem a Container-é, az AWT-vel szemben minden Swing komponens egy hordozó szerepet is ellát, habár ezt sok esetben nem szokásos használni (pl.: egy gombot nem használunk hordozóként, mert attól csak zavaros lesz a felület).

Az alábbiakban a JComponent tulajdonságait fogom részletezni, néhány helyen kitérve a top-level hordozókra, melyek számos pontban hasonlítanak az itt elhangzottakhoz és az előző fejezetbe nem fért bele.

5.1 Az add() metódus és validate()

A JComponent számos add() metódust örökölt. Ezek közül egy a Component-é, amely popup menü hozzáadását teszi lehetővé a komponenshez. A többi számszerint öt a Container-hez tartozik. Ezek az add metódusok adják a hordozó funkcionalitást. Az add() szintaktikája: add(Component), add(Component, int), add(Component, Object), add(Component, Object, int). Ezeknél a Component típusú a gyermek komponens, míg az opcionálisan megadhatóak közül az Object típusú bizonyos megszorításokat takar, melyek a LayoutManager-mek szólnak, illetve megadható a komponensek pozíciója/indexe az int értékkel. Ezentúl még az add(String, Component) metódust is örökölte a Container-ből, de az elavult.

A Container-ből örökölt metódusok közül fontos még a validate() is. Ennek a metódusnak a meghívása nélkül ugyanis nem jeleníti meg a hozzáadott komponenst. Ezzel egy időben viszont arra is lehetőségünk nyílik, hogy ha több komponenst is hozzá adunk azokat egyszerre jelenítsük meg, ami egyrészről kíméli az erőforrásokat, másrészről pedig bonyolultabb számolás igényesebb komponensek hozzáadásánál (pl.: helyben átszerkesztett képek)

lehetőség nyílik az egyszerre történő megjelenítésére és a számolás alatt például egy készültség jelzőt kirajzolni.

5.2 Méretek és poziciók kezelése

A grafikus megjelenítés során fontos a méretek kezelése. Ehhez négy fajta méretet kezel a Swing: size, maximum size, minimum size és preferred size. A JComponent-ben mind a négy mérethez van lekérdező és beállító metódus, de a size-hoz csak a Component-ből örökölt. Ez és sejteti, de követendő elvként kimodhatjuk, hogy a setSize()metódus használata általábban nem a kívánt (legtöbbször a semmilyen) hatást éri el, így nem is alkalmazandó (kivéve a top-level hordozók). Ellenben a többi méret beállító metódus alkalmazható. Az aktuális méret általában, a setPrefferedSize(Dimension)-zal beállítható, de mivel ez a LayoutManager felé csak javaslatként/kérésként továbbítódik, ezért elképzelhető, hogy nem éri el a kívánt hatást. Ilyenkor a minimum és maximum méretek állításával próbálhatjuk kényszeríteni a Java-t a kívánt eredmény kirajzolására. Az előbb említett metódusok a méretet Dimension példányként kezelik, ez a paraméter illetve a visszatérési érték típusa.

A komponenseknél fontos lehet a pozíció beállítása, hiszen például a top-level hordozóknál nincs LayoutManager, ami meghatározza azt, vagy, hogyha a LayoutManager-t null-ra állítjuk (habár ez elég komoly problémákhoz vezethet). A pozíció beállítására a setLocation() metódus használható, melynek a paraméter vagy egy x és egy y koordináta, vagy egy Point objektum.

A pozíció és a méret egyszerre is beállítható a setBounds() elljárással, amely paraméterként egy Rectangle-t vagy egy x,y és szélesség-magasság int négyest vár.

Az átméretezhetőség pedig csak a JInternalFrame-re tiltható le, hiszen felhasználó értelemben csak arra értelmezhető (plusz a top-level komponensekre).

5.3 A JComponent figyelői

A Swing-ben számos esemény és ezekhez kapcsolódó figyelő jelenik meg, melyek egy része már az ősökben is jelen volt mások pedig csak itt jelentek meg. A figyelők közül pedig nem egy a JavaBeans követelmények miatt került be. A JComponent figyelői az alábbiak: AncestorListener, VetoableChangeListener*, ContainerListener, PropertyChangeListener*,

ComponentListener, FocusListener, HierarchyBoundListener, HierarchyListener, InputMethodListener, KeyListener, MouseListener, MouseMotionListener, MouseWheelListener. Ezek közül a *-gal jelöltek JavaBeans követelmények, így ezekhez tartozik fire metódus is (fireVetoableChange() és firePropertyChange()). A JComponent továbbá rendelkezik egy getListeners(Class) eljárással, amely visszaadja a megadott osztályú ehhez a komponenshez regisztrált figyelőket. A figyelőket itt ennél részletesebben nem tárgyalom.

5.4 Interfészek

A JComponent három interfészt implementál: az ImageObserver-t, a MenuContainer-t és a Serializable-t. A MenuContainer szerepe abban az elvben rejlik, mely szerint a minden Swing komponens hordozó is, a többi pedig megegyezik a 4.3-as részben a leírtakkal.

5.5 Egyéb

Fontos még megjegyezni, hogy a JComponent rendelkezik egy setVisible(boolean) eljárással, amely a komponens láthatóságát szabályozza. A JComponent és leszármazottai alapértelmezve láthatók, míg a top-level komponensek nem.

A JComponent-nek ezenkívül beállítható a szegélye is a setBorder(Border) eljárással, illetve rendelkezik print(Graphics) eljárással a nyomtatáshoz, állítható a színe (előtér, háttér), betűtípusa.

6. Hordozó célú könnyűsúlyú komponensek

Itt azok a hordozók lesznek megbeszélve, amelyek ellentétben a top-level komponensekkel, könnyűsúlyúak és feladatuk nem a gyökér elem képezése a GUI struktúrában, hanem az egyéb könnyűsúlyú hordozókba, illetve top-level komponensekbe való beépülés hordozó célzattal. Tehát, ha egy GUI, mint egy fát tekintünk, ezek lesznek a köztes elemek, amelyek sem levél, sem gyökér szerepet nem töltenek be. Ezek a hordozók és a tartalmuk nem lesznek láthatóak, ha nincsenek behelyezve egy látható hordozóba, azaz nincs a felmenőik között egy nehézsúlyú hordozó.

6.1 JPanel

Az első és legfontosabb könnyűsúlyú hordozó a JPanel. A JPanel-t általában komponensek csoportosítására szokták használni. Ennek következtében az alapértelmezett LayoutManager-re FlowLayout.

6.2 JLayeredPane

A JLayeredPane-ről már említést tettem a JRootPane tárgyalásánál, ahol a JRootPane a gyermekeit kezelte vele. A LayoutManager-re null, így a komponensei tetszőlegesen pozíciónálhatóak, méretezhetőek. A JLayeredPane egy olyan könnyűsúlyú hordozó, mely rétegesen kezeli a gyermek komponenseit.

A különböző mélységű rétegeket int változókkal kezeli, ami az add() metódusnál megszorításként adható meg. Ehhez van még hat konstansa, amely hat speciális réteget definiál. Ezek rendre (a legalsóbbal kezdve): FRAME_CONTENT_LAYER (-30000), DEFAULT_LAYER (0), PALETTE_LAYER (100), MODAL_LAYER (200), POPUP_LAYER (300) és DRAG_LAYER (400). Az int indexelésnél a legfelsőbb a legnagyobb értékű réteg. A JLayeredPane a rétegein belül is definiál egy sorrendet, melyet szintén int indexekkel határozzunk meg, de az elhelyezés sorrendje pont fordított a rétegek index szerint sorrendjéhez, illetve a pozíció 0-től n-1-ig értelmezett, ahol n a rétegben

található komponensek száma (azaz 0 reprezentálja a legfelső elemet, illetve a –1-et a legalsó elem reprezentálására használjuk). Az add() metódusnak így van egy olyan formája is, hogy add(Component, int, int), ahol a hozzáadandó komponens után nem csak a réteg indexét, hanem az azon beüli pozíciót is megadjuk.

Ha egy komponenst már hozzáadtunk a JLayeredPane példányunkhoz, annak a pozíciója és a rétege a setLayer(), a moveToFront(), a moveToBack() és a setPositon() metódusokkal állítható. A setLayer()-nek és a setPosition-nek a paraméterei a komponens, illetve a réteg, vagy pozíció index megadva (setLayer()-nél akár mindkettő). A moveToBack() és moveToFront() eljárások paramétere pedig egy komponens, amelyet az eljárás a réteg legtetejére, vagy legaljára helyez, azaz a komponens a -1, illetve 0 pozícióba helyezi.

A JLayeredPane hatékonyan használható, ha például magunk akarjuk elhelyezni, méretezni a komponenseinket és a mélységüket is meg akarjuk határozni (pl.: háttérképet berakni).

6.3 JDesktopPane és JInternalFrame

A JDesktopPane a JLayeredPane leszármazottja. A belehelyezett komponensek JInternalFrame-k. A JInternalFrame egy olyan könnyűsúlyú hordozó, mely viselkedésében, illetve tulajdonságaiban is nagymértékben hasonlít a JFrame-re. A fő különbség viszont az, hogy a JInternalFrame, akárcsak az összes többi könnyűsúlyú komponens, más hordozóba helyezve használt. A JDesktopPane és a JInternalFrame rendeltetése, hogy virtuális asztalt, vagy több-dokumentumos felületet (Multiple-Document Interface - MDI) teremtsenek. A JDesktopPane biztosítja a virtuális asztalt, míg abban az ablakokat, azaz kereteket a JInternalFrame-k, vagy a több-dokumentumos felületet és abban a dokumentumokat.

A JDesktopPane a JLayeredPane-ben definiált lehetőségeken kívül még további eszközöket is nyújt a JInternalFrame-k kezelésére. Ezek a getAllFrames(), getAllFramesInLayer(int), getSelectedFrame(), amelyek az összes keretet, az összes keretet egy adott rétegben és az aktív keretet adják vissza. A getSelectedFrame-nek van set metódus is.

A JInternalFrame implementálja a RootPaneContainer interfészt és tulajdonságai nagyjából egyeznek a JFrame-nél leírtakkal. A megemlítendő különbségek közül az első, hogy a JInternalFrame „ikonizálható”. Ekkor létrejön egy JDesktopIcon példány és ez fog szerepelni a JDesktopPane-ben. Ennek a funkcionalitását a későbbi Swing verziók során bele akarják

olvasztani a JInternalFrame-be, azaz közvetlen használata nem ajánlott. Ezentúl a JInternalFrame még rendelkezik a JDesktopPane-nel történő kommunikációra készített eszközökkel: toBack(), toFront(), setLayer(), getDesktopPane(). Illetve állítható a kinézete a setUI()-val.

Minden JDesktopPane-hez tartozik egy úgynevezett DesktopManager. A DesktopManager egy interfész, mely a keretekkel elvégezhető műveletekhez vár el egy-egy metódust (pl.: keret bezárása, mozgatása, átméretezése…). Ennek egy alapértelmezett implementálása a DefaultDesktopManager. Ha viszont egy virtuális asztal kereteitől eltérő viselkedést várunk el, akkor a JInternalFrame-khez rendelhető egy az InternalFrameListener interfészt implementáló osztály, amely a keret egyes viselkedését speciálisan megadhatja. Ennek egy implementálása az InternalFrameAdapter, ha csak adott metódusakt szeretnénk speciális működéssel megvalósítani. Illetve még itt említhető meg, hogy a JInternalFrame-knek van saját esemény osztályuk az InternalFrameEvent, ami az AWTEvent közvetlen leszármazottja.

6.4 JTabbedPane

A JTabbedPane egy olyan hordozó, amely a komponensek hozzáadását és kezelését „fülekkel” oldja meg. Ez azt takarja, hogy különböző rétegek vannak, melyek közül a látható réteget az őt jelképező fülre kattintva érhetjük el. Ezek a rétegek egy-egy komponenst takarnak.

A komponensek a szokásos add() metódus mellett addTab() és az insertTab() metódusokkal is hozzáadhatjuk a JTabbedPane-hez. Az addTab()-nél meg kell adni a fül nevét String-ként és a komponenst, illetve a komponens előtt megadható még egy Icon típusú ikon is a fülnek, illetve ekkor, utána egy úgynevezett tipp String-ként. Ez a tipp egy felirat, ami megjelenik, ha az egeret a fülre mozgatjuk és várunk. Az insertTab() megegyezik az addTab() metódussal, annyi különbséggel, hogy az addTab() összes paraméterét várja és ezek mellett még egy indexet is, ami a beszúrás helyét jelképezi a fülek sorában. Az add() esetén pedig a szokásos formában adhatunk hozzá elemeket, ahol a megszorításoknál megadhatjuk a fül indexét, nevét és ikonját, illetve ha nem adunk meg nevet, akkor a komponens alapértelmezett neve lesz az.

A fülek helye a setTabPlacement() metódussal állítható be és négy int konstans (ezek a JTabbedPane által implementált a SwingConstants-ban találhatóak) használható hozzá, ezek a

TOP, a BOTTOM, a LEFT és a RIGHT. Az alapértelmezett a TOP. Az elhelyezés beállításánál a fülek tartalmának alakját és méretét, a fülek számát és hosszát, illetve a szükséges egérmozgást szokás figyelembe venni. A setTabLayoutPolicy() metódussal az is beállítható, hogy ha a fülek száma meghalad egy adott számot görgetve lehessen előhívni a nem láthatókat, vagy törje több sorba/oszlopba azokat.

A fülek kezelése indexekkel történik. Az index adott címke, név és komponens alapján lekérhető az indexOfTab() és indexOfComponent() metódusokkal. A fülek eltávolíthatók a remove()-val, aminek a paramétere egy komponens vagy a fül indexe, removeAll()-lal, ami az összes fület eltávolítja, illetve a removeAt()-tel, ami megegyezik a remove() index-alapú verziójával. A fül ikonja, címe, tippje és komponense beállítható és lekérdezhető index alapján.

Ezentúl beállítható és lekérdezhető az indexe alapján a fül előterének és hátterének a színe, gyorsbillentyűje.

6.5 JSplitPane

A JSplitPane olyan hordozó, amely két komponensből áll, melyek között az L&F-nek megfelelő határoló van és a két komponens aránya a felhasználó által dinamikusan változtatható anélkül, hogy a hordozó mérete változna.

A JSplitPane két komponense megadható a konstruktorban a felosztás iránya után, illetve setLeftComponent(), setRightComponent(), setTopComponent(), setBottomComponent() metódusokkal, melyek paramétere a komponens. Mivel a JSplitPane két komponenst tárol a felső és a bal illetve a lent és a jobb komponensek beállító metódus felülírja egymást. Ha valamelyik komponens nincs megadva, akkor alapértelmezve egy JButton van a helyén.. A setOrientation() a felosztás irányát adja meg a JSplitPane-ben definiált VERTICAL_SPLIT és HORIZONTAL_SPLIT konstansokkal.

A határoló mérete állítható a setDividerSize()-zal, aminek egy int a paramétere. A helyzete is állítható a setDividerLocation-nel(), aminek a paramétere egy double változó és a százalékos helyzetét határozza meg a határolónak (0 – fent/bal, 1.0 – lent/jobb). A setContinousLayout() metódus paramétere egy boolean és azt lehet megadni vele a JSplitPane-nek, hogy a méret változtatása alatt állandóan frissítse-e a két oldal tartalmát.

A fentebb leírt metódusokhoz természetesen tartozik lekérdező metódus is.

6.6 JScrollPane, JViewport és JScrollBars

A JScrollPane feladata a görgetősáv megadása, így a túl nagy méretű dokumentum kisebbként történő kezelésének megvalósítása. Ennek a megvalósítása úgy történik, hogy a JScrollPane áll egy JViewport típusú részből, ami a görgetősávval ellátott komponens látható része, négy JScrollBars-ból, amik a görgetősávokat és fejléceket reprezentálják, illetve négy további komponensből, amik a sarkok.

A JScrollPane-ben állítható és lekérhető vertikális és a horizontális görgetési politika (set/getVertical/HorizontalScrollBarPolicy()), melynek az implementált ScrollPaneConstans interfészben található konstansok (mindig, soha és ha szükséges) adhatóak meg. Ezentúl megváltoztatható, lekérhető illetve generáltatható a két görgetősáv példány is (set/get/createVertical/HorizontalScrollBar()). A nézőpont és a fejlécek is beállíthatóak és lekérdezhetőek.

Ha azt szeretnénk beállítani, hogy mit az a komponens, amit görgetünk, azaz a JViewport minek a nézete, akkor ezt a komponenst megadhatjuk a JScrollPane konstruktorában, vagy beállíthatjuk a setViewportView() paranccsal, aminek a paramétere a komponens.

A nézet komponense beállítható a JViewPort-ban közvetlenül is, a setView() metódussal. A JViewport-ban ezentúl még állítható a nézőpont aktuális pozíciója és mérete is. A nézet komponens görgetésének szükségességéről annak a preferált méret attribútum alapján dönt.

A JScrollPane alapértelmezve a számára készített ScrollPaneLayout-ot használja, amely rendelkezik egy addLayoutComponent() metódussal. Ezzel a JScrollPane fentebb leírt összes elemét megváltoztathatjuk a komponensen kívül a ScrollPaneConstans megfelelő konstansát megszorításként megadva (pl.: vonalzót adhatunk meg fejlécként).

Szokásos a görgetősávok használatát kerülni, ha nem szükséges, illetve ha mégis szükség van rá a grafikus felület adott részén, akkor igyekezünk kerülni a mindkét irányú görgetést.

A JScrollPane felépítéséből adódóan nagymértékben testreszabható, de ennek a részletezésre itt nincsen hely.

7. Adatkivitelre szolgáló komponensek

Az előző fejezetben különböző könnyűsúlyú hordozókat tárgyaltunk, így itt az ideje arról is szót ejteni, hogy mik azok a komponensek, amelyeknek a rendeltetése az, hogy a hordozókba helyezzük őket. Ebben a fejezetben az adatkiviteli célt szolgáló Swing komponenseket tárgyaljuk, illetve itt tárgyaljuk az ikonokat, amik habár nem komponensek, de szorosan idekapcsolódnak.

7.1 JLabel

Az első itt tárgyalt kiviteli komponens a JLabel, ami szöveg, kép, vagy mindkettő megjelenítését teszi lehetővé. Ezentúl gyakran használják más komponensek azonosítására is.

A címke által hordozott képnek, illetve szövegnek megadható és lekérdezhető a vízszintes és függőleges igazítása (set/getHorizontal/VerticalAlignment()), a címke és a kép távolsága, szöveg relatív pozíciója a képhez képest (a SwingContants implementálja és annak a LEFT, CENTER, RIGHT, LEADING és TRAILING konstansaival).

Mint azt az elején is írtam a JLabel-t gyakran használják komponensek azonosítására is és azok funkcionalitásának kijelzésére. Erre szolgála get/setLabelFor() eljárás is, ami lekéri/beállítja, hogy az adott címke mely komponenst azonosítja. Továbbá megadható úgynevezett „mnemonic” is, aminek a megjelenése, a megadott indexű karakter, vagy a megadott karakter első előfordulásának aláhúzása és ha az ALT mellett megnyomjuk ezt a karaktert, akkor a hivatkozott komponens requestFocus() eljárása emghívódik. A „mnemonic” a get/setDisplayedMnemonic() paranccsal lekérdezhető/megadható a JLabel-nek. Ugyanebben a hozzárendelési esetben előfordulhat, hogy a hivatkozott komponens funkcionalitása le van tiltva és ezt is ki szeretnénk jelezni, anélkül, hogy lecseréljük a címkét. A JLabel erre is szolgáltat megoldást úgy, hogy a JComponent-től örökölt setEnabled() eljárás használata esetén a szöveget elszürkíti, illetve a setDisabledIcon-nal megadható a letiltása esetére eltérő kép is.

7.2 Az Icon interfész és az ImageIcon

A JLabel-t, mint azt az előbb leírtam képek megjelenítésére is használt eszköz, a képeket pedig egy az Icon interfészt implementáló osztály példányaként jeleníti meg. Így habár nem a JComponent leszármazottai az Icon-t implementáló osztályok, de mindenként megemlítendőek. Az Icon alapértelmezve egy kis és fix méretű kép, azaz ikon reprezentálására szolgál és három metódust vár el az implementáló osztályoktól, melyek ezekkel a szélességüket és a magasságukat adják vissza, illetve kirajzolják maguk.

Az Icon-t implementáló osztályok közül a legfontosabb az ImageIcon osztály, mely az AWT-ben definiált Image-k megjelenítésére szolgál. A hivatkozott Image megadható/lekérdezhető a set/getImage() eljárással illetve a konstruktorban. A konstruktorban történő megadás előnye, hogy a kép megadható URL objektumként, képfájl elérési útjaként String-ben, amely URL-é konvertálódik és byte tömbként, aminek az AWT által támogatott formátumú képadatnak (gif, jpeg és png) kell lennie, illetve megadható még az előbbiek után egy tömör leírása a képnek String-ben.

Itt jegyezném meg, hogy a JLabel-ben a korábban leírtaknak megfelelően megadható egy letiltott kép is a címkéhez és előfordulhat, hogy csak az eredeti képet akarjuk elszürkíteni. Erre nyújt egyszerű megoldást a Swing GrayFilter osztály statikus createDisabledImage() metódusa, amely az Image paraméterű képéből egy szürkített Image-t képez. Ezt az ImageIcon lekérdező és beállító metódusaival kombinálva könnyen megkapjuk a kívánt eredményt.

7.3 JProgressBar

A JProgressBar egy folyamatos és bizonyos intervallum és az abban mozgó érték kijelzésére szolgál, tipikusan egy adott folyamat aktuális készenléti állásának kijelzésére.

Az intervallum definiálásához a BoundedRangeModel interfész valamely példányát használja, ami megadható a konstruktorban, vagy a setModel() eljárással. Az intervallum határai és az aktuális érték a set/getMaximum/Minimum/Value() eljárásokkal beállíthatóak és lekérdezhetőek int-ek formájában. Ezentúl a készültség százalékban is lekérhető (getPercentComplete) illetve beállítható ismeretlen állapot is (set/isDeterminate()).

A JProgressBar állapotának kijelzésének orientációja állítható, illetve megadható, hogy (a hozzárendelt L&F által definiált megjelenítésen mellett) az állapotát szövegesen is kiírja (set/isStringPainted()). Ez a szöveg a setString() metódussal megváltoztatható (alapértelmezve null és a BoundedRangeModel által megadott érték íródik ki). Ezeken kívül a keret kifestése is kérhető (is/setBorderPainted()).

7.4 JToolTip

Ha egy komponens-re rávisszük az egeret kérhető, hogy megjelenjen egy tipp szöveg a komponens felett. Ez egy JToolTip komponens. Általában ha egy komponenshez tippet akarunk megadni, akkor a JComponent definiált setToolTipText() metódust használjuk melynek a paramétere a kívánt tipp String formájában. Ekkor a továbbiakban a tipp megjelenítését egy ToolTipManager végzi.

8. A vezérlésre és az adatbevitelre szolgáló komponensek

Ebben a fejezetben a vezérléshez és az adatbevitelhez szükséges komponensek ismertetése kap, majd helyett. A vezérlésre szolgáló komponensek többsége az AbstractButton alatt található az osztályhierarchiában, így annak a tárgyalásával nyitom ezt a fejezetet.

8.l Az AbstractButton és a ButtonModel interfész

Az AbstractButton, mint neve is mutatja egy absztrakt osztály, amely a különféle gombok, menük és jelölőnégyzetek létrehozásához nyújt egy örökítendő alapot, viselkedésmódot.

Az AbstarctButton osztályban először definiálva vannak lehetőségek figyelők hozzáadására. Ezentúl a JLabel-nél leírtakkal egyező módón mind szöveg, mind pedig ikon hozzáadását lehetővé teszi a gombokhoz. Ezeknek pozícióját, elrendezését a JLabelnél leírtaknak megfelelően lehet változtatni. JLabel-nél megadható ikon száma is nőtt, itt még az ott látott típusokon (normál, letiltott) kívül megadható kijelölt, egér felette van és benyomva verziók és ezek egymással és a letiltottal való különféle értelmes kombináció. Ezentúl itt is megadható „mnemonic”.

Az előbbieken kívül még megadható és lekérdezhető egy Action is a gomboknak (set/getAction()), illetve program szinten is szimulálható megnyomásuk (doClick()). Lekérdezhető még a kijelöltsége is a gombnak és a setContentAreaFilled() paranccsal állítható, hogy a gomb átlátszó legyen-e (ez akkor használható, ha például ikont adtunk meg neki és nem akarjuk, hogy a gomb többi része is látszódjon, de ilyenkor az sem árt, ha figyelünk a szegély letiltatására is).

Mivel a nyomógombokon túl idetartoznak többek között a jelölőnégyzetek is, szükség lehet a gomb állapotának tárolására. Ennek következtében még megadható egy ButtonModel típusú modell is a set/getModel() eljárások segítségével.

A ButtonModel egy olyan interfész, amely az alábbiakat várja a nyomógomb modelljétől: be tudja állítani, és meg tudja mondani, hogy élesített, engedélyezett, be van nyomva, felette van az egér, kijelölt vagy mi a „mnemonic”-ja, továbbá hozzáadható és eltávolítható legyen Action/Change/ItemListener, beállítható legyen a billentyűcsoportja (ButtonGroup) és vissza tudja adni a hozzá generált eseményt azonosító String-et. A Swing-ben két a ButtonModel-t implementáló osztály van a DefaultButtonModel és JToggleButton.ToggleButtonModel.

8.2 JButton

A JButton a Swing nyomógombot megvalósító osztálya. A ButtonModel-je a DefaultButtonModel. A JButton kinézetének beállítása nagyjából, megegyezik a JLabel-évél és a fentebb leírtaknak megfelelően a gomb L&F-nek megfelelő megjelenése is láthatatlanná tehető a setContentAreFilled() metódussal.

A JButton megnyomásakor bekövetkező eseményeket, pedig ActionListener formájában szokták megadni a setActionListener() eljárással. Illetve előfordul, hogy egy adott területén a GUI-nak (kifejezetten dialógusoknál), ha megnyomjuk az enter billentyűt elvárjuk, hogy valamely gomb „benyomódjon”. Ez a JRootPane setDefaultButton() metódusával megadható a RootPaneContainer-eknek.

8.3 JToggleButton, JCheckBox, JRadioButton és a ButtonGroup

A JToggleButton egy kétállapotú gomb. Az aktuális állapota az isSelected() metódussal kérhető le. Megjelenése JButton-ban leírtaknak megfelelő. A JCheckBox és a JRadioButton a JToggleButton leszármazottai és az egyetlen nagyobb különbség köztük és az ősük között a megjelenésük, az, hogy miként rajzolódnak ki és az ebből és a megszokásokból adódó eltérő használat. Ezt pontosítva, míg a JToggleButton megjelenése egyezik a JButton-éval, addig a JCheckBox-nál és a JRadioButton-nál nincs kifestve a háttér és egyedi ikon is van hozzájuk rendelve (ebből adódóan nem is szokás megadni neki ikont, mert az felülírná azt, amit adtak nekik). A használatbeli eltérés, pedig abban áll, hogy míg a JCheckBox-ot egy opció ki vagy bekacsoltságának beállítására szokták használni, addig a JRadioButton-t több lehetséges mód közül pontosan egy kiválasztására szokás használni ButtonGroup-ba helyezve. A JToggleButton használatára pedig mindkettő jellemző az előbbiek közül.

Az előbb említettem a ButtonGroup-ba helyezést, de nem árt, ha az is itt áll, hogy pontosan mi is az. A ButtonGroup a Swing egy eszköze, melybe a gombokat tudjuk csoportosítani. Ez a csoportosítás logikai szintű. Az egy csoportban szereplő gombokra az alábbi igaz: a csoportból, pontosan egy gomb lehet kijelölve se több, se kevesebb, kivéve a kezdeti állapotot, amikor még egy sem lett kijelölve, ez programszinten sem szeghető meg (ez persze

megkerülhető, ha az adott gombot előbb töröljük a ButtonGroup-ból, majd megszűntetjük a kijelölését és aztán visszatesszük csoportba).

A ButtonGroup-ba elemet az add()-dal tehetünk, a remove()-val távolíthatunk el. ButtonGroup-ba az AbstractButton bármely példány, leszármazottjának példány helyezhető, de csak a JToggleButton-nal és annak leszármazottaival van értelme használni.

8.4 JMenuBar

A programok grafikus felületeinek napjainkban fontos részét képezik a menük. A JMenuBar a menüsort megvalósító osztály a Swing eszköztárában. A JMenuBar a JComponent közvetlen leszármazottja. Menüsor hordozóhoz való hozzáadásához nem az add() metódust, hanem a setJMenuBar() eljárást szokás alkalmazni, aminek oka, hogy menüsorból jellemzően maximum eggyel rendelkezik egy hordozó. Ezt a metódust a RootPaneContainer osztályok (a JWindow kivételével) és a JRootPane maga implementálja.

A JMenuBar jellegéből adódóan tartalmaz egy kijelölési modellt is, aminek a SingleSelctionModel interfészt implementáló osztálynak kell lennie. Ez az interfész azt mondja ki, hogy maximum egy eleme lehessen kijelölve a menünek. A JMenuBar alapértelmezve a DefaultSingleSelctionModel implementációt használja.

A menüsor, mint azt már megszokhattuk, itt is vízszintesen kerülnek kirajzolásra az elemek. Ehhez a JMenuBar egy BoxLayout-ot használ. Az elemek hozzáadása az add() metódus segítségével történik, lekérdezésük pedig index alapján a getMenu()-vel. A menüsornak elvileg megadható külön súgó menü is, amit a megjelenés külön kezelne a többi menütől, de ezt még nem implementálták még a Java 1.5-ben sem.

Az előbbieken túl megadható/lekérdezhető, az elemek és a menüsor kerete közti távolság (set/getMargin()), a menüsor valamely elemének kijelöltsége (set/isSelected()), a keret kirajzolása (set/isBorderPainted()).

Ezentúl még itt említendő meg, hogy a különböző menüelemek ideértve a JMenuBar-t implementálják a MenuElement interfészt, amely az egyedi menük könnyebb építését hivatott lehetővé tenni. A MenuElement interfész öt eljárás implementálását várja el: getComponent, getSubElements, menuSelcetionChanged, processKeyEvent és processMouseEvent. Ezek közül a JMenuBar az elsőre önmagát, a másodikra az elemeit adja vissza, a többi nem csinál semmit. A MenuElement-et implementálja még a JMenuItem, a JMenu, a

JCheckBoxMenuItem, a JRadioButtonMenuItem és a JPopupMenu az itt tárgyalt eszközök közül.

8.5 JMenuItem

Az AbstractButton közvetlen leszármazottja, amely egyszerű menüelemet valósít meg. A JMenuItem nem más, mint egy gomb, amely egy listában jelen esetben egy menüsorban, vagy menüben található gomb, amelyet, ha megnyomunk valamilyen funkcionalitás bekövetkezik.

A JMenuItem, mint azt az előbb is írtam az AbstractButton közvetlen leszármazottja, így mint ott is leírtam a megjelenése a JLabel-éhez hasonlóan változtatható és adható meg hozzá „mnemonic" is. A működése hasonló a JButton-éhoz. A funkcionalitás implementálásához itt is ActionListener használata a jellemző, habár Action-t is használhatunk. A JButton-hoz képest még fontos különbséget képez a JMenuItem-ek működésében a menübe helyezettségen kívül az is, hogy gyorsbillentyű, úgynevezett „accelerator" adható meg. Ez egy billentyűkombináció, amely a „mneomonic"-kal szemben, akkor is működik, ha a menüelem éppen nem látható. Az ilyen gyorsbillentyűket a menüelem neve után szoktak, gyakran kisebb betűkkel kiírni. Hozzáadásuk a setAccelerator eljárással történik, aminek a paramétere egy KeyStroke példány. A KeyStroke egy olyan osztály, melynek csak statikus metódusai vannak és példányosítani is csak ezeken át lehet, általában a getKeyStroke eljárással. A getKeyStroke eljárásának több változata van. Általában a egy karakter kódját (az AWT-s KeyEvent konstansaival, pl.: VK_A, VK_ENTER) és egy módosító kódját (az AWT-s InputEvent konstansaival, pl.: CTRL_MASK).

8.6 JMenu

A JMenu a JMenuItem leszármazottja és egy almenüt valósít meg. Ennek a gyakorlati szintű megvalósítása a JMenu típusú menü elemhez egy JPopupMenu rendelésével történik, azaz, ha rákattintunk egy JMenu-re egy megfelelően pozícionált felugró menü jelenik meg.

A JMenu-t általában egy JMenuBar-hoz, vagy egy másik JMenu-höz szokás hozzáadni. A JMenu-höz elemeket az add és insert eljárásokkalal adhatunk, aminek paramétereként megadhatunk Action-t, String-et, amik az adott funkcionalitással vagy névvel adnak hozzá egy új menüelemet a menühöz, vagy közvetlen egy JMenuItem-t is megadhatunk. A menühöz

az elválasztó vonal adható hozzá a menü végére/tetszőleges pozícióba az add/insertSeparator eljárással (ez egy JSeparator példányt generál és ezt adja hozzá a menühöz). Ezentúl hozzáadható menüfigyelő is lekérdezhetőek a pozíciójuk alapján az elemek, az hogy a legfelső szinten (közvetlen a menüsor gyermeke) és mivel a megvalósítás egy JPopupMenu-vel történik az is lekérdezhető.

A JMenu-ben a gyorsbillentyű hozzáadásának JMenuItem-es verzióját felülírták ezzel megakadályozva, hogy gyorsbillentyű legyen rendelhető egy almenühöz.

Itt fontos még megjegyezni, hogy egy menü tervezése során fontos szempontot képez az átláthatóság és a kezelhetőség. Ennek az eléréséhez két fontos szabály van: a menüket ne ágyazzuk egymásba három szintnél mélyebben, illetve egy menüben az adott szinten általában hét plusz-mínusz két elem legyen. Ezek közül az első szabály a fontosabb, ha nem tudjuk mindkettőt betartani.

8.7 JCheckBoxMenuItem és JRadioButtonMenuItem

Ez a két menüelem típus is a JMenuItem közvetlen leszármazottja, mint a JMenu, de feladtuk kevésbé fontos. Funkciójukat és megjelenésüket tekintve megegyeznek a JCheckBox és JRadioButton eszközökkel, a különbég mindössze annyi, hogy ezek menüelemek. Állapotuk a set/getState() eljárással kérhető le és ők is ButtonGroup-ba helyezhetőek.

Fontos még ha használjuk ezeket az elemeket funkcionalitásbeli eltérésük miatt az átlag menüelemekhez képest célszerű őket elkülöníteni elválasztó vonallal vagy pedig külön menübe helyezni őket.

8.8 JPopupMenu

Már korábban is említést tettem a JPopupMenu-ről a JMenu-ben, de most részletesebben is foglalkozni fogok vele. A JPopupMenu a JComponent leszármazottja és a felugró menüt valósítja meg. Mint azt már korábban leírtam általában a JMenu használja az almenük megvalósítására, de az is gyakran előfordulhat, hogy az egér jobb gombjára, vagy egyéb eseményre reagálva megjelenjen egy felugró menü. Ezt valósítja meg ez az osztály.

A JPopupMenu alapértelmezve GridBagLayout-ot használ, mint LayoutManager és a DefaultSingleSelectionModel-t használja kijelölési modellként. Menüelemek az add eljárás JMenu-nél leírt módján adhatóak hozzá (a JMenu gyakorlati szinten ezeket rögtön továbbítja a JPopupMenu-jének). Ezentúl az insert metódussal megadott pozícióba adható hozzá komponens, vagy Action, illetve remove-val eltávolítható a megadott indexű elem. A JPopupMenu-höz adható még elválasztó vonal is (addSeparator()). Ezentúl lekérdezhető és beállítható a címkéje (get/setLabel), a margói (get/setMargin()), a kiváltója (get/setInvoker()), a keretének kifestése(is/setBorderPainted), továbbá beállítható a mérete (setPopupSize()) és a helye (setLocation()).

A JPopupMenu különlegessége, hogy képes könnyű és nehézsúlyú komponensként egyaránt működni, ez az set/isLightWeightPopupEnabled eljárással állítható be és kérdezhető le az aktuális komponensre, míg a statikus set/getDefaultLightWeightPopupEnabled metódus az alapértelmezésre vonatkozik.

8.9 JSeparator

A JSeparator egy általános célú elválasztó vonal, amit általában a menükben használunk, de a JComponent közvetlen leszármazottja. Amikor a menükben használjuk az elválasztókat az addSeparator eljárás automatikusan generálja őket, de előfordulhat, hogy nem menüben akarjuk használni, ekkor explicit módon kell létrehozni őket. Ebben az esetben állítható az orientációja a setOrintation() eljárással a SwingConstants HORIZONTAL és VERTICAL értékeire. A JPopupMenu és a JToolBar ennek egy-egy speciális alosztályát használja.

8.10 JToolBar

A JToolBar-ral eszköztárat lehet létrehozni. Implementálja a SwingConstans interfészt, alapértelmezett LayoutManager-re a BoxLayout, állítható az orientációja (setOrientation(int)), a keret kirajzolását (setBorderPainted(boolean)), a margó szélessége (a gombjai és a kerete közti táv – setMargin(Insets)), a keret kirajzolásának ideje (csak akkor rajzol keretet, amikor az egér rajta van, vagy mindig – setRollover(boolean)).

Az eszköztárak általában a menü alatt találhatóak a GUI-ban, de előfordulhat, hogy áthúzhatóak az egérrel az ablak más oldalaira is. Ez a JToolBar-ban is implementálva van és a setFloatable(boolean) paranccsal engedélyezhető. Ha ezt engedélyezzük ajánlott a JToolBar-t egy úgynevezett négyoldalú LayoutManager-ű hordozóba tenni, tipikusan BorderLayout-ba és a négy oldalát szabadon hagyni. Ha az eszköztárat mozgathatóra állítjuk, akkor arra figyelni kell, hogy bizonyos L&F-ek nem támogatják ezt a művelet és letiltják az állíthatóságát.

Az eszköztárhoz található a Swing-ben még a JToolBar.Separator, ami JSeparator leszármazottja és a feladata az eszköztár elemeinek elválasztásával történő vizuális csoportosítása.

Ha egy programban eszköztárat teszünk, akkor a tervezésénél nem árt bizonyos szempontot figyelembe venni, hogy az eszköztár tényleg használható legyen és a rendeltetése is az legyen, ami egy eszköztáré. Ezek közül az egyik legfontosabb az eszköztár szerepe, ami alapján az eszköztár három főbb csoportba tartozhatnak: eszköz/módválasztás, funkciók gyors elérése és navigáció. Az elsőnél célszerű a már hasonló piacon lévő szoftverek jelölésrendszeréhez közelálló jelöléseket/ikonokat használni. A másodiknál a túlterhelést célszerű megakadályozni úgy, hogy csak a legfontosabb, leggyakrabban használt funkciókat hozzuk ki a menüből. A harmadiknál akárcsak az első esetnél a legfontosabb az érthetőség, azaz a megfelelő ikonok kiválasztása a fontos. Az elválasztók használata pedig a használhatóságát az eszköztárnak az előbbiekkel kombinálva tovább növeli.

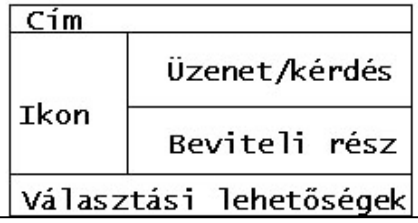
8.11 JOptionPane

A JOptionPane a JComponent közvetlen leszármazottja, de viselkedését tekintve komoly hasonlóságokat mutat a JDialog-gal és mindig modális. Az ok amiért ebben a fejezetben kerül megvitatásra, hogy habár egy hordozóról van szó, de felhasználása főként adatbeviteli vagy vezérlési feladatokra korlátozódik.

A JOptionPane-nek négy fő felhasználása van: megerősítés kérése, input kérése, üzenet kiírása és lehetőségek felkínálása. Ezek mind dialógus formájában történnek és mind a négy lehetőség gyors megjelenítésére külön statikus eljárás van írva a JOptionPane-ben (showConfirm/Input/Message/OptionDialog,). Ezek mindegyike egy-egy JDialog-ot hozz fel, ha a szülőkomponensük JFrame, illetve egy-egy JInternalFrame-t, ha a szülő JDesktopPane, vagy abba van beleágyazva. Ebből következően ezeknek a metódusoknak meg kell adni egy

szülőkomponenst, továbbá kötelező megadni egy üzenet komponenst is, ami a kiírt üzenetet vagy kérdést tartalmazza. Ezentúl még egy JOptionPane áll egy címből, egy input részből (amikor szükséges) és egy gombokból álló opciók részből (4.-es ábra). A cím opcionális mind a négy esetben, akárcsak az ikon. Ha nem adunk meg ikont, akkor megadható egy üzenettípus int-tel, ami egy L&F-től függő automatikus ikont rendel a generált dialógushoz. Ehhez a konstansok: XXX_MESSAGE, ahol az XXX lehet ERROR, INFORMATION, WARNING, QUESTION, vagy PLAIN.

4-es ábra: JOptionPane szerkezete



A megerősítést kérő verzió egy ablakot hozz fel, ahol igen, nem és esetleg mégse opciók közül lehet választani. Itt a szokásoshoz képest megadható egy bonusz int változó is, amely a kiírt opciókat változtatja. Ehhez a konstansok: YES_NO_OPTION, YES_NO_CANCEL_OPTION. Beviteli rész nincs a visszatérési értéke int, amelyhez az alábbi konstansok valamelyike feleltethető meg: YES_OPTION, NO_OPTION, CANCEL_OPTION, OK_OPTION, CLOSED_OPTION.

A következő a beviteli eset. Itt egy String-et kérhetünk be vagy egy választást egy listából a felhasználótól. Ebben az esetben a szokásoshoz képest megadható még kezdőérték, vagy a választási lehetőségek egy tömbje, illetve lehetőség van egy szülőnélküli csak az üzenetet, vagy még mellé a kezdőértéket tartalmazó paramétereket megadni.

Az üzeneti esetnél csak a szokásos inputok között választhatunk, nincs beviteli mező és egy OK gomb van mindössze az opcióknál.

Az utolsó az opciók kiírására szolgáló eset. Ennek az eddigiekkel ellentétben egy formája van. Ebben megadandó a szülőkomponens, az üzenet, a cím, az opciók típusa (a korábban említett int értékek, csak akkor fontos, ha nem adunk meg a későbbiekben opciókat), üzenettípus, ikon, az opciók tömbje és a kezdőérték. Visszatérése a kiválasztott opció indexe, vagy a CLOSED_OPTION (-1) érték, ha bezárták.

A JOptionPane lehetőséget nyújt egyedi dialógusok képzésére, de ezeknél már elveszti az egyszerűségének és gyorsaságának az előnyét, illetve a megjelenés rovására is mehet, így ilyenkor célszerűbb sajátmagunknak megírni azokat JDialog, vagy JInternalFrame formájában.

8.12 JSpinner

A JSpinner elemek egy sorozatából való értékválasztásra szolgáló eszköz. Megjelenítését tekintve két részből áll, az egyik az értékek megjelenítésért felelős, míg a másik két nyílból, melyekkel az értéket léptethetjük (ezeknek a funkcionalitása a fel és le nyilakkal is elérhető). Az értéket tartalmazó rész alapértelmezve egy JTextField, de ez manipulálható a set/getEditor metódussal. Négy ilyen előre definiált szerkesztési rész van, ezek a JSpinner-be ágyazott DateEditor, ListEditor, NumberEditor, DefaultEditor statikus osztályok. Ezek dátum, szöveg és szám bevitelre szolgálnak, illetve az utolsó egy csak-olvasható beviteli eszköz.

A JSpinner eszköz mögött egy a SpinnerModel interfészt implementáló osztály, modell áll. Ez az interfész a modelltől azt várja el, hogy képes legyen visszaadni, a JSpinner aktuális, előző és következő elemét, az aktuálisat meg is tudja változtatni, illetve ChangeListener-k kezelését is elvárja. A JSpinner a saját az előbbiekkel megegyező metódusait (a ChangeListener-re vonatkozót kivéve), mind delegálja a modelljének. Az eszköz modellje a get/setModel metódusokkal kezelhető. A JSpinner modelljeihez is tartozik négy már megírt osztály, ezek az AbstractSpinnerModel, a SpinnerDateModel, a SpinnerListModel és a SpinnerNumberModel , melyek közül az első egy absztrakt osztály, a további modelleknek, mely csak a ChangeListener kezelését oldja meg, míg a másik három a dátum, szöveg és szám bevitelhez tartozó modellek. A dátum és szám bevitel modelljénél állítható az intervallum és a léptetés mérete, míg a szöveges listánál egy List adható meg, amely az értékeket tartalmazza.

8.13 JSlider

A JSlider numerikus értékek intervallumból kiválasztására szolgáló grafikus eszköz. Megjelenítését tekintve egy „csúszka”, melynek megadható az orientációja, rovátkázható két szinten, melyek sűrűsége állítható, illetve kiírhatók értékek is adott pontjaihoz. A csúszka értéke egérrel, irány gombokkal illetve a PgUp és PgDn gombokkal változtatható. A kiírandó címkék szövege a set/getLabelTable metódussal állítható, melyek Dictionary típusként kezelik a címkéket, de a createStandardLabel eljárás segítségével megadott inkremenssel esetleg kezdőértékkel generálható egy HashTable, mely az intervallumba tartozó a paraméterekkel behatárolt címkéket tartalmazza, mint számokat. Az orientáció állíthatósága mellet még az értékek sorrendje is invertálható ám ennek a két tulajdonságnak a

megváltoztatása nem ajánlott, mivel már megszokott a vízszintes és balról jobbra növekvő csúszka. A csúszkák sűrűsége a set/getMinor/MajorTickSpacing metódussokkal kérdezhető le/adható meg a két rovátka közti távolság int-ben történő megadásával.

A JSlider mögött egy BoundedRangeModel osztályú modell áll, ami a set/getModel-lel kezelhető. A JSlider saját minimum, maximum és aktuális értékek beállítására/lekérdezésére szolgáló metódusai (set/getMinimum/Maximum/Value) delegálódnak a modellnek.

8.14 JFileChooser

Ez az eszköz a könyvtárak közötti navigálásra jött létre, azon belül a fájlok, könyvtárak kiválasztására mentéshez, vagy megnyitáshoz. Megjelenítését tekintve eléggé összetett, számos egyéb Swing elemből épül fel.

Működését tekintve a JFileChooser a showOpen/-/SaveDialog parancsokkal jeleníthető meg. Az Open és Save verziók a nevüknek megfelelő elfogadó billentyűvel jelennek meg, míg a szimpla verziónál meg kell adni annak a nevét. Ezeknek az eljárásoknak a viszzatérési értéke int és a JFileChooser-ben definiált konstansokkal térnek vissza a fájlnavigálás eredményével (CANCEL_OPTION, APPROVE_BUTTON, ERROR_OPTION). A kijelölt fájl(ok) a get/setSelectedFile(s) parancsokkal érhetőek el, míg az aktuális könyvtár a set/getCurrentDirectory-val kezelhető. Ezeken túl állítható az elfogadó gomb mnemonic-ja, szövege, tippje, a fájlok áthúzásának engedélyezése, a több fájl kijelölésének lehetősége.

A megjelenítendő fájlok szűrhetőek, erre FileFilter absztrakt osztály szolgál, amelynek meg kell tudnia mondani, hogy egy adott fájlt elfogad-e, illetve a saját leírását. Egy ilyen szűrő a set/getFileFilter-rel kezelhető, ha csak egyet akarunk belőle használni, illetve az add/removeChoosableFileFilter-rel és a getChoosableFileFilters-szel, ha többet is.

A fájlokról megjelenítendő információ is állítható. Ez lehetőséget ad többek között a platform függő fájlinformációk elrejtésére/megjelenítésére, ami növeli a swing platform függetlenségét. A fájlinformációk megjelenítéséért egy FileView példány a felelős (set/getFileView). A FileView egy absztrakt osztály mely öt eljárást tartalmaz, ezek paramétere egy fájl és fájl leírását, ikonját, nevét, típusleírását, illetve még egy, amely a könyvtáraknak a fájlstruktúrában levélelemként kezelését teszi lehetővé.

Az előbbiek mellett még egy a FileSystemView (ez az osztály is absztrakt) objektum is áll, amely az operációsrendszer specifikus adatokhoz történő hozzáférésért felelős és fájlrendszer kezelésért felelős.

A FileFilter, FileView és FileSystemView külön csomagot alkot a swing-en belül javax.swing.filechooser néven.

Megjegyzendő még, hogy fájlrendszer navigáláshoz ajánlott ennek az osztálynak a használata saját írása helyett, mivel amellett, hogy ez egy jól megírt osztály, a már megszokott megjelenítéseknek megfelelően íródott, így a felhasználónak feltételezhetően nem okoz majd problémát a kezelése.

8.15 JColorChooser

A JColorChooser egy színválasztásra alkalmas eszköz. Megjelenítését tekintve egy három elemű JTabbedPane-ből, egy előnézeti részből és a vezérlő gombokból áll. A JTabbedPane három füle három külön színválasztási lehetőséget takar (Swatches, HSB és RGB). Az osztály tartalmaz egy statikus metódust egy színválasztó megjelenítéséhez (showDialog), amely a kiválasztott színnel tér vissza, illetve egy olyan statikus metódust, amely egy JDialog-ot ad vissza a megadott paraméterek alapján.

Ha példányosítjuk a színválasztót állítható az előnézeti panel (set/getPreviewPanel), az aktuális szín (set/getColor), a kijelölési modellje (set/getSelectionModel) és kezelhetőek a színválasztó panelek (add/removeChooserPanel, set/getChooserPanels).

A kijelölési modell a ColorSelectionModel interfésznek megfelelő osztály egy példánya, amelynek állítani kell tudnia a színt (set/getSelectedColor) és kezelnie kell ChangeListener-t.

A színválasztási lehetőségeket kínáló felületeknek az AbstractColorChooserPanel absztrakt osztály leszármazottainak kell lennie. Ez a JPanel közvetlen leszármazottja és öt absztrakt metódussal rendelkezik, amelyeket implementálnunk kell, ha saját színválasztó felületet akarunk készíteni. Ezek a metódusok a build/updateChooser, melyek a felület inicializálásáért és a változások frissítéséért felelősek, a getDisplayName és a getSmall/LargeDisplayIcon, melyek közül az első a nevét adja vissza, a másik kettő pedig a kis és nagy ikonját a színválasztónak. Az alapértelmezett színválasztó felületek és az előnézeti rész ColorChooserComponentFactory osztály statikus metódusaival generálódnak.

9. LayoutManager-ek

Ebben a fejezetben az úgynevezett LayoutManager-ek lesznek tárgyalva. Ezek az AWT-ben jelentek meg, de a Swing is őket használja és a szerepük a megjelenítésben kifejezetten fontos, mivel ők a felelősek a hordozók gyermekeinek megjelenítésért (elhelyezés és méretezés a komponens minimum, maximum és preferált méretének megfelelően). Amennyiben egy hordozó LayoutManager-e null, az összes komponens elhelyezését, manuálisan kell megadni, azok mérete pedig statikus lesz.

Általánosságban megállapítható, hogy a LayoutManager-ek használata ajánlott.

9.1 LayoutManager és LayoutManager2 interfész

Az összes Layout-nak (elrendezésnek) implementálnia kell a LayoutManager interfészt, amely öt metódus megvalósítását követeli meg. Ezek az add/removeLayoutComponent, melyekkel komponensek adhatóak az elrendezéshez, a minimum/preferredLayoutSize, amely megadja adott hordozónak a komponenseiből következő minimális és ajánlott mértetét és a layoutContainer, amely elrendezi az adott hordozót (kiszámolja és beállítja a komponensek pozícióját és méretét).

A LayoutManager interfésznek van egy LayoutManager2 nevű alinterfésze is. A LayoutManager2 interfész a komponensek megszorítás-alapú elrendezésére ad lehetőséget. Ennek fényében a LayoutManager-ben megkövetelt metódusokon túl még további öt jelent meg. Ezek az addLayoutComponent-nek egy megszorítással kiegészített alternatívája, a maximumLayoutSize-t, amely adott hordozó maximális méretét hivatott megadni (a megszorítások miatt ennek az ismerete is szükségessé válhat). A getLayoutAlignmentX/Y, amely a komponensek egymáshoz viszonyított távolságát hivatott visszaadni (egy 0 és 1 közé eső float szám, amely a lehető legközelebb és legtávolabb közti intervallumot reprezentálja). Végezetül pedig az invalidateLayout, amely az esetlegesen a komponensekről eltárolt információk érvénytelenné válását jelzi az elrendezőnek.

Futási időben a LayoutManager-eket váltogatni nem ajánlott, mivel a megszorítások miatt gyakran nem jelennek meg a megszorítással el nem látott (vagy a elrendező megszorításainak nem megfelelő) elemek a LayoutManager2-ben.

Ezeknek az interfészeknek a felhasználásával létrehozhatóak saját Layout-ok, de mivel ez nem minden esetben egyszerű az AWT számos előre megírtat tartalmaz és néhány új a Swing-ben is megjelent. A fejezet további részében ezeknek a funkciót, használhatóságát, előnyeit és hátrányait fogom tárgyalni.

9.2 FlowLayout (AWT)

A FlowLayout egy megszorítás nélküli LayoutManager, melyet a JPanel alapértelmezve használ. Elrendezési elve elég egyszerű: a hordozó orientációjának (balról jobbra, vagy jobbról balra) és a Layout elrendezésnek (közép, balra, jobbra, a kezdő és a befejező elemhez igazított) megfelelően sorjában kirajzolja az elemek és, ha az aktuális sor már megtelt, új sort kezd.

Állítható az elemköz (set/getH/VGap) és az elrendezés (set/getAlignment). Gyakran használják gombok kirajzolásához, de összetett alkalmazásoknál nem mindig célszerű a használata, mert kérdéses lehet, hogy minden komponens kifér-e.

9.3 BorderLayout (AWT)

A BorderLayout az egy gyakran használt LayoutManager, már alapértelmezve is. Komponenskezelése statikus és megszorítás-alapú, azaz a LayoutManager2 interfészt implementálja. A megszorításoknak megfelelően öt különböző részre bontja a megjelenítést: North, South, West, East, Center. Ezekhez tartozik megfelelő String konstans is, alternatív konstans nevekkel a BorderLayout-ban definiálva, melyek a megszorításokat képezik. A szokásosak mellet állítható a vízszintes és függőleges köz is (set/getH/VGap). Egy megszorítási területre csak egy komponens kerülhet, és az újabbak elfedik a régieket.

Fontos még megjegyezni, hogy a komponensek méretezésénél a hordozó teljes területének a felhasználása az elsődleges cél és csak a komponensek egymáshoz képest felvett arányánál veszi figyelembe a komponens ajánlott méretét.

9.4 BoxLayout és Box (Swing)

Ez LayoutManager a komponenseket a hordozónak a megadott tengelye (X/Y/LINE/PAGE_AXIS) szerint a komponensek optimális méretének megfelelő méretben egymás után rajzolja ki. Implementálja LayoutManager2-t, de nem használ megszorításokat. Sort nem tör még akkor sem, ha nem fér ki az összes komponens.

Itt jegyzendő meg, hogy található egy Box nevű hordozó is a Swing-ben, amely a BoxLayout-t használja és annak a könnyebb használhatóságát hívatott elősegíteni. Ezt különféle statikus eljárásokkal hajtja végre, melyekkel Box és különféle láthatatlan komponensek példányosíthatóak (createVertical/HorizontalBox/Glue/Strut és createRigidArea). A láthatatlan komponensek az elemek pozicionálását segítik. Három fajtájuk van: glue, strut és rigid area. Ezek közül a glue egy olyan komponenst generál, mely akkor amekkora hely a hordozó végéig hiányzik (több glue esetén ez osztódik), a strut fix szélességű vagy magasságú komponens, míg a rigid area-nek mind a magassága és szélessége is fixált.

9.5 GridLayout (AWT)

A GridLayout a hordozót rácshálóra bontja, majd az így kapott egyenlő méretű téglalapokba helyezi a komponenseket. A sorok és oszlopok száma megadható, a hordozó területét teljesen megtölti a komponensekkel és a komponensek méretét ennek megfelelően, azonos értékűre állítja. Állítható és a konstruktorban is megadható a köz (set/getH/VGap) és a sorok, illetve az oszlopok száma (set/getRows/Columns). Használatánál figyelni kell arra, hogy gyakran nem a kívánt eredményt szüli a komponensméretezési gyakorlata miatt.

9.6 CardLayout (AWT)

A CardLayout a komponensek, mint egy kártyapaklit kezeli, a nevét is innen kapta. Megjelenítését tekintve mindig csak egy komponenst láttunk és az teljesen kitölti a rendelkezésére álló területet.

A CardLayout lehetővé teszi a komponensei közötti választás lehetőséget, amihez azonosító String-eket, vagy pedig a hordozóban definiált sorrendet használja fel.

Implementálja a LayoutManager2-t és elvárja a megszorítás megadását, ami az előbb említett String az elemre történő hivatkozáshoz.

A fontosabb metódusok a komponensekhez történő hozzáféréshez a first, last, next és previous metódusok, melyek paraméter a szülőkomponens. Ezek a hordozó komponensei között felállított szekvenciális sorrendnek megfelelően megváltoztatják az aktuális kirajzolt elemet. Ez előbbi négy metóduson túl még a megszorításként megadott String-gel is meg lehet jeleníteni egy komponenst, ez a show eljárással történik.

9.7 OverlayLayout (Swing)

Az OverlayLayout olyan megjelenítést tesz lehetővé, melynek során komponenseket helyezhetünk egymásra, mint a CardLayout-nál, csak itt az optimális méretüket veszik fel és egyszerre több elem is látható, megfelelően igazítva.

A komponensek elrendezése a hozzájuk rendelt igazításnak megfelelően történik. Az igazítást a komponens set/getAlignmentX/Y metódusával kezelhetjük, amely egy képzeletbeli középponthoz viszonyít. A komponensek elfedése a kirajzolás sorrendjében történik, azaz a hordozóban definiált sorrend szerint, amíg valamelyik újra ki nem rajzolódik.

Az OverlayLayout gyakran használják például, ha egy gombnál a szöveget az ikonra akarjuk helyezni.

9.8 GridBagLayout (AWT)

A GridBagLayout hasonlóan a GridLayout-hoz rácshálósan felbontja a hordozót, a komponenseket viszont nem csak egy rácstéglába helyezhetjük, hanem akár több szomszédosat egyesítve újabb nagyobb téglalapot hozhatunk létre adott komponenseknek, illetve a komponenseket az optimális méretűknek megfelelően méretezi és az oszlopot vagy sort a legtöbb helyett elfoglaló komponens méretéhez igazítja. Ennek a megvalósításához szükséges volt, hogy a GridBagLayout implementálja a LayoutManager2.

A GridBagLayout-ba helyezett komponenseknek a meg kell adni egy GridBagConstraints típusú megszorítást. A megszorítások állíthatóak a set/getConstraint-tel, illetve megadhatóak LayoutManager2-nek megfelelően a komponens hozzáadásánál.

A megszorítás tulajdonságai megadhatóak a konstruktorban, de gyakorlatban egy GridBagLayout-ot használó hordozóknál általában egy GridBagConstraints példányt szoktak használni és ennek az értékeit módosítják mielőtt a komponens hozzáadásra kerül. Ez annak köszönhető, hogy csak a hozzáadásnál használja fel a Layout a GridBagConstraints értékeit, így egy részről rengeteg energiát spórolhat meg magának a programozó és az erőforrásokat is kevésbé erőlteti meg ez a megoldás. Ennek a megvalósításához természetesen nem árt a GridBagConstraints ismerete. A komponens rácsmezőben való elhelyezkedését az induló sarok a hálóban vett koordinátái alapján adhatjuk meg a gridx és gridy tulajdonságokkal, ahol 0 az első sarok. Ezeknek az induló érétke a RELATIVE konstans, amely az előző után helyezi el közvetlen az aktuális elemet. A magasság és a szélesség rácsokban értelmezve a gridwidth és gridheight értékekkel adhatóak meg, melyek alapértelmezve 1 értékűek és a REMAINDER konstanssal az aktuális sor/oszlop maradék celláit kitölti. A fill tulajdonsággal (NONE, VERTICAL, HORIZONTAL, BOTH) felülbírálhatjuk azt, hogy a komponensek az optimális méretüket vegyék fel. Az ipadx és ipady a komponens méretéhez add hozzá a megfelelő tengely szerint adott értéket (ehhez a komponens minimum méretét használja). Az insets tulajdonság pedig egy Inset típusú objektum, amely a komponensek közötti tér méretét állítja. Az utolsó publikus mezője, amit még nem részleteztem az anchor. Ez a komponens elrendezését állítja, ha az nem töltené be a rendelkezésére álló helyet. Lehetséges értékei: NORTH, SOUTH, WEST, EAST, ezeknek a kombinációi, CENTER (ezek voltak az abszolút értékek), PAGE_START/END, LINE_START/END, FIRST/LAST_LINE_START/END.

Az itt leírtakból is érződhet, hogy habár nem mindig egyszerű a GridBagLayout használata, de a benne rejlő lehetőségek az egyik legsokrétűbb és leghasználhatóbb Layout-á teszik.

9.9 SpringLayout (Swing)

A SpringLayout a Swing egyik a LayoutManager2-t implementáló újdonsága, amely a komponensek az eddigieknél is dinamikusabb elrendezését hivatott megvalósítani.

A SpringLayout komponenseinek megjelenítése SpringLayout.Cosntraints objektum segítségével történik. A SpringLayout.Constraints egy beágyazott osztály, amely négy úgynevezett Spring-et tartalmazz, ezek a komponens x, y szerinti pozícióját (megadott élhez képest), a szélességét és a magasságát reprezentálják.

Két él később is egymáshoz rendelhető a putConstraint metódussal, amelynek megadjuk, hogy melyik élt (NORTH, SOUTH, EAST, WEST) és komponenst linkeljük, majd egy fix értéket, vagy egy Spring-et a két él távolságára, végül pedig azt az élt és komponenst, amihez linkelünk. A megszorítások lekérhetőek a getConstraints metódussal adott komponenshez, vagy a getConstraint-tel adott élhez.

A megszorítások által használt Spring egy absztrakt osztály, amely egy távolság minimális, maximális, optimális és aktuális értékét reprezentálja. Ezek az értékek az absztrakt getMinimum/Maximum/Preffered/-Value metódusokkal lekérdezhetőek és az aktuális érték a szintén absztrakt setValue-vel állítható. A Spring ezeken túl viszont rendelkezik számos osztályszintű metódussal is. A fontosabb statikus metódusok a constant, amely egy Spring példányt add vissza és a paramétereként megadható a minimum, maximum és optimális érték, vagy egy szám mindnek. A height és width metódusok egy komponens magassága vagy szélessége alapján példányosít, a scale egy Spring skálázott verzióját, míg a minus az inverzét hozza létre. Ezeken túl még van max és sum függvények, amelyek két Spring adott értékei közül mindig a maximálisat kiválasztva példányosít egy új objektumot, illetve összeadd két Spring-et.

Fontos még megjegyezni, hogy ha nem adunk meg megszorításokat egy komponenshez, akkor a SpringLayout nem is rendel hozzá, így az automatikusan a (0,0) pozícióban az optimális méretének megfelelően jelenik meg, illetve a komponensek a kirajzolás sorrendjében takarják a többiek lefedett részeit.

Összegezve a SpringLayout egy dinamikus és nagy tudású Layout, amely bármely más Layout viselkedését képes megvalósítani.

9.10 Általánosságban a LayoutManager-ekről

Általánosságban nézve azt, hogy mikor milyen LayoutManager-t érdemes használni az alábbiakat állapíthatjuk meg.

Egyszerűbb esetekben, mindössze néhány komponens (2-3) megjelenítésénél BorderLayout-ot esetleg Flow vagy BoxLayout érdemes használni. Ha a komponensek elhelyezése nem bonyolult (például a gyorsképek megjelenítése egy fotógalériában) a Flow és BoxLayout használata a ajánlott attól függően, hogy akarunk-e sortörést. A CardLayout

használata speciális esetekben célszerű (például egy rövidebb válogatok és egy gyorsnézet részleget, akarok nagy sebességgel elérni), akárcsak a GridLayout-é.

Bonyolultabb felületeknél pedig a GridBagLayout, illetve a SpringLayout ajánlott (például összetettebb adatlapok beállítások).

10. Szöveges adatok kezelése

A swing szöveges adatok kezeléséhez számos osztályt nyújt, attól függően, hogy mire akarjuk használnia szöveges komponenst. Ezek az osztályok a JTextComponent és a leszármazottai. Ezen szöveges eszközök mögött, pedig egy külön csomag áll a javax.swing.text. A továbbiakban a JTextComponent, a leszármazottai és a hozzájuk tartozó csomag szolgáltatásai lesznek kifejtve.

10.1 JTextComponent

A JTextComponent egy absztrakt osztály, amely az alapját képezi a swing-ben felhasználható szövegkezelésre szolgáló osztályoknak és, mint ősosztálynak számos ehhez szükséges metódust kell definiálnia. Érdekesség, hogy a text csomagban található, míg leszármazottai közvetlenül a swing-ben.

A metódusok mellett egy struktúrát is definiál a JTextComponent, amely tovább öröklődik a leszármazottakban. Ezen struktúra szerint a modell és a nézet külön áll (függelékek 10.1-es ábra). A szöveg mögött pedig egy Document típusú modell áll, amiről lentebb lesz szó. A Document a get/setDocument metódusokkal kezelhető. Ebből a Document-ből egy ViewFactory példány készít egy View típusú nézetet.

A ViewFactory interfész egyetlen metódust definiál, amely egy Element típusú elemből készít egy View-ot. A View akárcsak a ViewFactory a javax.swing.text csomag része. A View absztrakt osztály felelős a grafikus megjelenítéséért adott szövegnek és ennek következtében számos leszármazottja van.

A generált View-hoz párosul még egy Highlighter implementációja (set/getHighlighter), amely adott szövegrészek megjelenítésénél kiemeléseket eszközölhet, például a szövegrész kijelölésénél, illetve még a kurzor is ráépül. A Highlighter interfész metódusokat biztosít ezeknek a kiemeléseknek a intervallumokként történő kezelésére.

A JTextCompoent lehetőséget nyújt szövegrészek kijelölésére (select, selectAll, getSelectedText, set/getSelctionStart/End) és ezeknek a szövegszerkesztőkben megszokott vágólapra helyezésére (copy), kivágására (cut) és beillesztésére (paste), illetve a kijelölt szöveg helyetesíthető (replaceSelection). Beállítható benne a kijelöléshez (set/getSelectionColor, set/getSelectedTextColor), illetve a letiltott szöveghez

(set/getDisabledTextColor) használt szín. Megadható és lekérdezhető a szöveg (set/getText), illetve inicializálható adatfolyamból és kiírható oda (read, write). Megadható margó (set/getMargin), a szövegszerkesztés iránya (set/getComponentOrientation). Engedélyezhető a szerkeszthetőség (set/isEditable), a drag&drop (set/getDragEnabled). A navigációhoz állítható az azt irányító NavigationFilter (set/getNavigationFilter) és a kurzor (set/getCaret), amikről később lesz szó, a kurzornak pedig állítható a színe és a pozíciója (set/getCaretColor/Position, moveCaretPosition). Lekérhető a megjelenítésben az adott pozícióhoz tartozó téglalap (modelToView) és adott ponthoz tartozó pozíció (viewToModel). A JTextComponent kezel még Caret és InputMethodListener-t, fókusz accelerator-t (set/getFocusAccelerator). A szöveghez még úgynevezett KeyMap-ek is adhatóak (add/remove/get/set/loadKeyMap), melyek adott billentyűhöz, billentyűkombinációhoz add meg valamilyen felülbírálást az alapértelmezett hatáshoz képest. Implementálja a Scrollable interfészt.

A swing-ben a szövegszerkesztés Action-ök sorozataként kezelődik. Ezek az Action-ök lekérhetőek a getActions metódussal, mint tömb. A text csomagban külön absztrakt implementációja is található az Action interfésznek, mégpedig a TextAction.

10.2 A Document és a StyledDocument interfészek

A Document egy a javax.swing.text csomagban található interfész, amely egy strukturált, módosítható és megfigyelhető dokumentumhoz biztosít alapokat. A Document a tartalmát a Java ideológiájának megfelelően Unicode karakterekként kezeli.

Ezeknek a karaktereknek a dokumentumban elfoglalt helyük két fajta módon adható meg egy pozícióval vagy egy offset-tel. Az offset egy szám, amely azt takarja, hogy hányadik helyen áll a karakter a dokumentumban, míg a pozíció egy Position interfész implementáció, amely egyetlen metódussal rendelkezik, amely visszaadja a pozícióhoz aktuálisan tartozó offset-et. A Position is a javax.swing.text csomag része. A két mód között a különbség a strukturált dokumentumoknál jön ki, ahol a szerkesztés során az offset állandóan változik a szerkesztett rész után, míg a pozíció, például a bekezdés száma ritkábban. A karakterek helyének kezeléséhez a Document interfész biztosít adott offset pozíciójának (createPosition), a dokumentum kezdő- és végpozíciójának (getStart/EndPosition) lekéréshez függvényt.

A struktúra kezeléséhez a Document az Element interfész implementációit használja. Ezek az elemek, mint hierarchikus szerkezet definiálják a dokumentum struktúráját. Az Element intefész által meghatározott elemeknek a dokumentum egy adott részét fedik le, ebből adódóan rendelkeznek egy kezdő és egy vég offset értékkel (getStart/EndOffset). Az elemeknek van neve (getName) és tartozik hozzájuk egy AttributeSet, amely attribútumok egy halmaza (getAttributes) és egyetlen dokumentumhoz tartoznak (getDocument). Mivel a struktúrát definiáló elemek faszerkezetben épülnek rá a dokumentumra ezért ennek a szerkezetnek megfelelően lekérhető az elem szülője (getParentElement), a gyermekek száma (getElementCount), adott gyermek index alapján (getElement), az index offset alapján (getElementIndex) és az, hogy levélelem-e az aktuális elme (isLeaf). A dokumentumra ezekkel az elemekkel összetett logikai- és formátumstruktúra építhető. Egy dokumentumra akár több struktúra is építhető, így lekérhető a dokumentumban az alapértelmezett gyökér (getDefaultRootElement) és az összes gyökérelem (getRootElements).

Egy dokumentumnak még az eddigieken túl lekérhető a hossza (getLength) és adott részlete offset és hossz alapján (getText), illetve be is illeszthető bizonyos offset-re, adott AttributeSet-tel szövegrész (insertString), vagy eltávolítható offset és hossz alapján (remove). A dokumentum kezel még DocumentListener-t és UndoableEditListener-t. Ezek közül az utóbbi a dokumentum szerkesztésénél a visszavonás opció swing szintű támogatása miatt fontos. Ami még fontos lehet a Document interfésznél az, hogy feltételezi, hogy a dokumentum rendelkezik úgynevezett tulajdonságokkal, mint például az inicializáló adatfolyam. Ezek a tulajdonságok lekérdezhetőek és bővíthetőek (get/putProperty). A Document modellbe beleépítették még a konkurens szerkesztés támogatását, így rendelkezik a megjelenítéshez egy render metódussal, amelynek paraméterként megadandó egy Runnable implementáció.

A Document absztrakt implementációja az AbstractDocument. Ennek a leszármazottai közvetlenül a DefaultStyledDocument és a PlainDocument, a DefaultStyledDocument-en át a HTMLDocument. Ezek a PlainDocument kivételével a Document interfész StyledDocument szubinterfészét használják.

A StyledDocument interfész a Document által definiált metódusokon túl még számos metódust tartalmaz, amelyek a formázáshoz fontosak. Így AttributeSet alapján meg tudják adni a betűtípust (getFont), az előtér (getForeground) és a háttér színét (getBackground), illetve adott szövegrészre új AttributeSet-et adható meg (setCharacterAttributes). Ezentúl,

mivel szükséges a karakterszintű formátumkezelés, lekérhető az adott offset-ű karaktert reprezentáló elem (getCharachterElement). Kezel még paragrafusokat is, így lekérhető az adott pozícióhoz tartozó szövegrészt lezáró paragrafus elem (getParagraphElement), illetve adott szövegrészben átállítható az összes paragrafus AttributeSet-je (setParagraphAttributes). A StyledDocument az eddigieken kívül még úgynevezett Style-okat os kezel, ami egy speciális AttributeSet. A dokumentumhoz így hozzáadható Style, eltávolítható (add/removeStyle), illetve a paragrafusokhoz külön kezelendők (set/getLogicalStyle).

10.3 Az AttributeSet, Mutable AttributeSet és a Style interfészek

Az eddigiek során sok szó esett az AttributeSet interfészről. Az AttributeSet text csomag egy olyan interfész, melynek az implementálói attribútum kulcs-érték párosokat tartalmaznak és kezelnek. Ennek megfelelően elvárja, hogy meg tudja adni, hogy tartalmazza-e az adott kulcsot (isDefined), kulcs-érték párost (containsAttribute), vissza tudja adni adott kulcshoz tartozó értékeket (getAttribute). Ezentúl még az attribútumok száma és azok neve is lekérdezhető (getAttributeCount/Names), illetve képes a saját attribútumaiból egy új AttributeSet-et készíteni, plusz megmondani, hogy részhalmaza-e egy AttributeSet (copy/containsAttributes). Van még egy saját egyenlőségi metódusa is (isEqual), illetve az attribútumok visszafejthetőek egy hierarchia alapján (getResolveParent). A visszafejtésnek akkor van szerepe, ha egy attribútumot nem találunk meg az adott AttributeSet-ben attól az még vonatkozhat az adott szövegrészre, ilyenkor visszamegyünk egy szinttel feljebb a hierarchiában a visszafejtés segítségével.

Az AttributeSet tartalmaz még négy beágyazott interfészt, melyek speciális attribútumokat definiálnak: karakter (ChracterAttribute), betűtípus (FontAttribute), szín (ColorAttribute) és paragrafus (ParagraphAttribute) attribútumok.

Ezeket az AttributeSet-eket használják az Element-ek a szöveg formázására, aminek megfelelően számos implementációja van, akárcsak a beágyazott interfészeinek.

Az AttributeSet-nek két szubinterfésze található a javx.swing.text csomagban. Ezek közül az első a MutableAttributeSet. Az eddigiekből látszódott, hogy az AttributeSet nem biztosít lehetőséget az attribútumok módosítására. Ezt hívatott orvosolni a MutableAttributeSet, amihez öt új metódust definiál. Ezek az attribútum, attribútumok hozzáadása és eltávolítása (add/removeAttribute/Attributes) és a visszafejtendő szülő átállítása (setResolveParent).

A második szubinterfész a Style, amely MutableAttributeSet-nek is leszármazottja. Ez az interfész mindössze a ChangeListener kezelésével és a Style-t azonosító (getName) név használatával egészíti ki az ősét.

10.4 A Caret és a NavigationFilter

A szövegszerkesztésnél, ha navigálunk, akkor azt a képernyőn megjelenő kurzor segítségével valósítjuk meg. A Caret interfész a kurzor megvalósításához szükséges viselkedésmódot definiálja.

A Caret intefész két pozíciót társít a kurzorhoz. Az első a pont, ami a modellben elfoglalt helyét adja meg, a másik a kézjegy pedig, ha kijelölt szövegrész van a kijelölés másik vége. Ha nincs kijelölés ez a kettő egyenlő. A pontnak állítható a pozíciója a kézjeggyel együtt, vagy anélkül, illetve lekérdezhető mind a pont, mind a kézjegy (set/move/getDot, getMark). Beállítható még a megjelenített pozíció is (set/getMagicCaretPosition), a villogás sebessége (set/getBlinkRate), a láthatóság (set/isVisible), a kijelölés láthatósága (set/isSelectionVisible). Elvár a Caret az eddigieken kívül még metódust a kirajzolásra (paint), az UI JTextComponent-be való beépítéséhez és eltávolításához (install, deinstall), illetve a ChangeListener-ek kezelésére. Alapértelmezett implementációja a DefaultCaret.

A Caret tartalmazz metódusokat a dokumentumban való navigálásra, de elképzelhető, hogy ebben korlátozni akarjuk a felhasználót. Az ilyen korlátozások megvalósítására használható a NavigationFilter osztály. Amikor a JTextComponent-ben megváltozna kurzor pozíciója, akkor az a hozzárendelt NavigationFilter-t hívja meg először. A NavigationFilter három metódussal rendelkezik, ezek a setDot, moveDot, illetve a getNextVisualPositionFrom. Az első kettő megegyezik a Caret azonos nevű eljárásaival, annyi különbséggel, hogy megadandó nekik még egy NavigationFilter.FilterByPass és egy Position.Bias. A Position.Bias a mozgás irányát adja meg, míg a NavigationFilter.FilterByPass módosítja, ha szükséges a mozgást. A getNextVisualPositionFrom pedig a következő látható pozíciót adja vissza.

10.5 JTextField, JFormattedTextField és JPasswordField

A fejezet eddigi részeiben megismerhettük a szöveges komponensek alapjait, de mivel a JTextComponent egy absztrakt osztály, ezért közvetlen nem használható.

A JTextField egy olyan megvalósítás, amely a sima egysoros szöveg kezelésére alkalmas. A céljai miatt egy PlainDocument áll mögötte, mint modell. Állítható a szöveg igazítása (set/getHorizontalAlignment), megadható a mérete oszlopokban, ami az m betű szélességével egyezik meg (set/getColumn), kezel ActionListener-t, ami az Enter megnyomására sül ki. Ennek következtében megadható neki Action is (set/getAction).

A JTextField-nek két alosztálya van, amelyek plusz funkcionalitást adnak az ősükhöz. Ezek a JFormattedTextField és a JPasswordField.

A JFormattedTextField a bevitt szöveg formázására ad lehetőséget, mint például, ha dátumot viszünk be. A formátum kezeléséhez a beágyazott AbstractFormater leszármazottait használja. Ennek implementációja a DefaultFormatter, melynek két alosztálya van az InternationalFormatter és a MaskFormatter. A MaskFormatter egy maszk String segítségével formázza a szöveget. Az InternationalFormatter hívatott az egyéb specifikusabb formázásokat megvalósítani, így leszármazottai a DateFormatter és a NumberFormatter is, amelyek dátum és szám bevitelének formázására hivatottak. A formázók generálására még rendelkezik a JFormattedTextField egy másik beágyazott osztállyal a AbstractFormatterFactory.

A JFormattedTextField-nek fontosabb metódusai még a formázó megadása (set/getFormatter), a formátumgyár megadása (set/getFormatterFactory) az értékének beállítása (set/getValue), az aktuális bevitel érvényesítése (commitEdit) és az érvényesség lekérése (isEditValid).

A másik leszármazottja a JTextField-nek a JPasswordField, amely a jelszó bevitelre alkalmas. A bevitt szöveget nem írja ki és a helyette megjelenített karakter beállítható (set/getEchoChar), illetve lekérdhető, hogy van-e megadva, azaz nem 0 (echoCharIsSet). Az értéke char tömbként lekérhető (getPassword), illetve a kivágás és vágólapra másolás eljárások módosítva lettek úgy, hogy ne működjenek, csak visszajelzést küldjenek az UI-nak.

10.6 JTextArea

A sima többsoros szöveg kezelésére a JTextArea eszköz szolgál. Az ehhez a JTextField-hez hasonlóan a PlainDocument-et használ modellnek, és nem használhat többféle betűtípust és színt. A többsoros megvalósítás miatt engedélyezhető az automatikus sortörés (set/getLineWrap), illetve beállítható, hogy szavak, vagy betűk szintjén törjön

(set/getWrapStyleWord), lekérhető adott sornak a kezdő és vég offset-je (getLineStart/EndOffset), illetve adott offset-hez a sor (getLineOfOffset). A sorok száma is lekérhető (getLineCount), beállítható a betűtípus (setFont). A szerkesztésnél hozzáfűzhető (append), beszúrható (insert) és helyettesíthető (replaceRange) szöveg továbbá állítható a tabulátor mérete (set/getTabSize).

10.7 JEditorPane és JTextPane

A JEditorPane a szöveges beviteli megvalósítások közül az egyik legösszetettebb. Számos többsoros tartalom megjelenítésére és szerkesztésére nyújt lehetőséget. Alapértelmezve háromféle tartalmat ismer, ezek a text/plain, a text/html és a text/rtf. A tartalom típusa a set/getContentType-pal állítható be. A különféle tartalmak kezelését EditorKit-ekkel (set/getEditorKit) végzi és, amennyiben megváltoztatjuk a tartalomtípusát, automatikusan meghívódik a getEditorKitForContentType metódus, majd az általa visszaadott EditorKit állítódik be a JEditorPane-hez. Megjegyzendő, hogy az integritás érdekében az EditorKit átállításakor a dokumentum modellje is megváltozik. Adott tartalomtípushoz meghatározható a hozzárendelendő EditorKit is a setEditorKitForContentType-pal. Az alapértelmezett hozzárendelés is átállítható (registerEditorKitForContentType). A sima tartalomként értelmezi a fel nem ismert típusokat is. Adott tartalomtípushoz lekérhető a hozzárendelt EditorKit osztály neve is (getEditorKitClassNameForContentType).

A különféle EditorKit-ek teszik lehetővé, hogy eltérő típusú szöveges dokumentumokat szerkeszthessünka JEditoPane-nel. Ezek mind az EditorKit absztrakt osztály leszármazottai. Az EditorKit osztály legtöbb metódusa absztrakt, így azokat saját készítésénél mind implementálni kell. Ezek a metódusok kurzor (createCaret), az inicializálatlan új dokumentum (createDefaultDocument) generálása, a tartalomtípus (getContentType) és a ViewFactory (getViewFactory) lekérése és az írás, illetve olvasás adatfolyamból vagy adott íróból, olvasóból (write, read). Nem absztrakt metódusoként jelen van még a JEditorPane-be való installálás és eltávolítás (install, deinstall). Az EditorKit alapértelmezett megvalósítása a DefaultEditorKit, amely text/plain, illetve ismeretlen tartalomtípus esetén hívódik meg. Ennek leszármazottja a StyledEditorKit, amely a stílusban is formázott dokumentumok kezeléséhez nyújt plusz lehetőségeket őséhez képest többek között beágyazott stílust érintő Action

implementációkkal. Neki a leszármazottai text/html és text/rtf típusokhoz tartozó EditorKit-ek, a HTMLEditorKit és az RTFEditorKit.

A JEditorPane kezel még HyperLinkListener-t, illetve a szövegkezeléséhez is rendelkezik még pár saját metódussal, amelyek nem tartoznak az EditorKit-ek hatáskörébe. Ezek az aktuálisan megjelenített URL (set/getPage), szöveg (set/getText) kezelése, a kijelölt szövegrész lecserélése (replaceSelection) és megadott referenciára ugrás (scrollToReference).

A JTextPane a JEditorPane leszármazottja, amely amellett, hogy megőrizte azt a képességét, hogy számos tartalomtípust kezeljen, lehetőséget ad a különféle swing komponensek beágyazására a JTextPane-be. Ennek következtében lehetőség adódik a tartalom régiókra bontására és ezekhez külön Style-ok rendelésére. Ezeknek a fényében a JEditorPane-hez képest új metódusok vannak a Style-ok (add/get/removeStyle), a logikai stílus (set/getLogicalStyle), a karakter és paragrafus (set/getCharachter/ParagraphAttributes), illetve az input attribútumok (set/getInputAttributes) kezelésére. A dokumentumokat, mint StyledDocument-eket kezeli, így ezekhez is külön beállító metódusok rendelődnek (set/getStyledDocument). A JTextPane-be beilleszthetőek még komponensek is és ikonok is, így ezeknek a beillesztésére is van eljárás (insertComponent/Icon).

11. Adatstruktúrák reprezentálása

Ebben a fejezetben a különféle adatstruktúrák reprezentálására szolgáló eszközökről lesz szó. A Swing három adatstruktúra megjelenítéséhez add eszközkészletet, ezek a listák, a táblázatok és a fák, de ezeknek az eszközkészleteknek a mérete indokolja, hogy külön fejezetet érdemeljenek.

11.1 JList

A JList elemek egy listájának megjelenítését és annak az elemeinek a kiválasztását teszi lehetővé.

A JList mögött két modell áll: egy ListModel és egy ListSelectionModel.

A ListModel interfész felelős a listának megfelelő működésért. Ehhez két metódus implementálását várja el: a getElementAt-ét és getSize-ét. Ezek az adott indexű elemet és a lista méretét hivatottak visszaadni. Ezentúl még elvárja a ListDataListener-ek kezelését is. A ListModel a get/setModel-lel kezelhető, illetve a konstruktorban is megadható. A konstruktorban egy Vector vagy egy tömb is megadható, ekkor erre épül fel egy DefaultListModel. Ugyanez később is elérhető a setListModel eljárással, ekkor egy új modell példány jön létre. Ezen implementáció mögött egy Vector áll, aminek a legtöbb metódusa közvetlen meg is jelenik delegált metódusok formájában.

A másik JList mögött álló modell a ListSelectionModel, ami a kijelölések kezelését végzi. A SelectionModel három kijelölési politikát kezel, ezek a SINGLE_SELECTION (csak egy elem lehet kijelölve egyszerre), a SINGLE_INTERVAL_SELECTION (csak egy intervallum lehet egyszerre kijelölve, de az egynél több elemből is állhat) és a MULTIPLE_INTERVAL_SELECTION (több tetszőleges számú elemből álló intervallum is ki lehet egyszerre jelölve), amelyek a set/getSelectionMode-dal állíthatóak. A kijelölések kezelését illetően számos metódust előír a ListSelectionModel interfész, melyekkel intervallumokat jelölhetünk, adhatunk hozzá indexek szerint, illetve hossz és index szerint, beállíthatjuk az intervallumok kezdő, illetve befejező indexét, törölhetjük és lekérdezhetjük a létezését a kijelöléseknek. Összesen 14 ilyen metódust vár el az interfész. Az eddigieken túl még kezelni kell tudnia ListSelectionListener-t (add/removeListSelectionListener), illetve még specifikál egy set/getValueIsAdjusting() metóduspárt, amelyeknek a szerepe az

eseménykezelés időzítésében van. A ListSelectionModel-nek alapértelmezve a DefaultListSelectionModel implementációját használja a JList.

A modellek legtöbb metódusa delegálva megtalálható a JList metódusai között, illetve a kettő kombinációjaként a kijelölt elemek közvetlen elérésére is van függvény (getSelectedValue/Values). Ezekből is látható a JList dinamikus, ellenben nem árt megjegyezni, hogy az elemek közvetlen szerkesztésére nem add lehetőséget.

Az elemek vizuális megjelenítéséért egy ListCellRenderer implementáció felelős, hasonlóan. A ListCellRenderer interfész egyetlen függvény megvalósítását várja el: a getListCellRendererComponent-ét. Ennek öt paramétere van: a JList, amelyhez tartozik, a lista kirajzolandó eleme, annak az indexe, a kijelöltség és a fókusz kérdése. A visszatérési értéke pedig egy Component, melynek a paint metódusa lesz felhasználva a kirajzolásnál és az optimális mérete, ha nem fix méretű a lista. A JList alapértelmezve a DefaultListCellRenderer implementációt használja. A ListCellRenderer a get/setCellRenderer metódussal állítahtó.

Az elemek megjelenítésénél a JList-ben állítható a háttér és a betű színe, ezeknek a metódusoknak a működése ListCellRenderer-től függ (a DefaultListCellRenderer figyelembe veszi őket), így annak az egyedi implementációknál erre sem árt odafigyelni. A celláknál beállítható fix szélesség és magasság is (set/getFixedCellWidth/Height), vagy prototípus Object (set/getPrototypeCellValue) a lista elemek méretének beállításához. Ha a fix szélesség és magasság –1, illetve a prototípus null, akkor az elemek optimális mérete alapján számolja ki a JList a méreteit. Ez nagy listánál időigényes lehet, ekkor az előbbi metódusok használata ajánlott.

A JList még implementálja a Scrollable interfészt is, amely habár nem add közvetlen görgethetőséget, de ha egy JScrollPane-be helyezzük, akkor a görgetés léptetésének méretét az elemek szélességének és magasságának megfelelőre állítja. Ezentúl az ensureIndexVisible eljárás segítségével elérhetjük, hogy a megadott indexű elem látható legyen, illetve ha nincs a képernyőn, akkor odagördül rá. A látható elemek száma beállítható a set/getVisibleRowCount-tal, illetve a getCellBounds függvénnyel lekérhető egy indextartomány megjelenítéséhez szükséges terület.

Érdekes és említésre méltó metódusai a JList-nek még a getNextMatch, ami adott indexből, adott irányba indulva visszaadja az első a megadott prefix-nek megfelelő elem indexét, illetve a locationToIndex és indexToLocation függvények, amelyek egy képernyőn

értelmezett pontot egy indexhez rendelnek. A locationToIndex kifejezetten hasznos, ha például adott menü elemen az egérrel dupla-klikkhez valamilyen eseményt akarunk hozzárendelni.

A JList a JComponent getToolTipText metódusát is felüldefiniálja, hogy listaelemenként egyedi szöveg jelenhessen meg. Ehhez egy úgynevezett ToolTipManager-t használ.

A listaelemek megjelenítésének elrendezése is állítható a setLayoutOrientation-nel. A lehetséges elrendezések a VERTICAL (a listaelemek egymás alatt rajzolódnak ki), a HORIZONTAL_WRAP (a lista elemek egymás mellett rajzolódnak ki és, ha már nincs hely nekik, sortöréssel folytatódik elölről) és a VERTICAL_WRAP WRAP (a lista elemek egymás alatt rajzolódnak ki és, ha már nincs hely nekik, sortöréssel folytatódik fentről).

A JList használatára felállítható pár általános szabály. Többek között célszerű a lista méretét alacsonyan tartani. Ha viszonylag sok elem (20-nál több) található benne, akkor valamilyen logika alapján rendezni kell a használhatósághoz (például abc sorrend). Ezentúl, ha már nagyon nagy a listaelemek száma (több 100), akkor szükséges lehet valamilyen kereső rész is. Másik fontos követendő szokás az, hogy célszerű a lista kiegyensúlyozott szélességére figyelni. Ez azt jelenti, hogy amennyiben legtöbb elem jelentősen rövidebb, mint a maximális hosszúságú, akkor inkább kevésbé széles listát érdemes létrehozni és a leghosszabb elemet csonkolni. Ekkor a leghosszabb elem megjelenítése például történhet tooltip segítségével, vagy egér dupla-klikkjére írt figyelővel megnyitott részlekenél.

11.2 JComboBox

A JComboBox egy legördülő szerkeszthető listát megvalósító eszköz. Megjelenítését tekintve áll egy megjelenítő vagy szerkesztő részből illetve egy gombból, amelynek hatására a legördülő menü előjön (mivel a két komponense egy gombba van ágyazva ezért a legördülés, ha nem szerkeszthető a menü, akkor az értékre kattintva is elérhető). Az érték megjelenítésért egy ListCellRenderer felelős (set/getRenderer), ha az érték nem szerkeszthető, egyébként egy ComboBoxEditor (set/getEditor). A JList-hez hasonlóan a megjelenítésnél minden egyes értékhez kiszámolja a szükséges helyett, hogy megállapítsa a kívánt szélességet. Ez a setPrototypeDisplayValue metódussal küszöbölhető ki, amellyel egy prototípus adható meg a megjelenítés méretéhez.

A JComboBox mögött egy ComboBoxModel áll (set/getModel), amely a JList mögött álló ListModel leszármazott interfész és az őséhez képest mindössze azt várja el, hogy beállítható és lekérhető legyen a kijelölt elem (set/getSelectedItem). A ComboBoxModel-nek van még egy alinterfésze, a MutableComboBoxModel, ami még az elemek hozzáadására és eltávolítására is lehetőséget ad. Példányosításnál a JComboBox-nak megadható a modellje, vagy egy tömb esetleg Vector, ekkor egy DefaultComboBoxModel épül fel a tömbből, vagy Vector-ból. A modellből adódóan a JComboBox-ban megtalálhatóak a getItemAt, getItemCount, set/getSelectedIndex/Item, illetve az add/removeItem, removeItemAt metódusokkal (habár az utóbbi három csak a MutableComboBoxModel esetén működik).

Az általános használatához a JList-nél leírtak javasolhatóak, annyi módosítással, hogy a nagy elemszámú listához írandó keresőt itt a MutableComboBoxModel-lel könnyen megoldhatjuk.

11.3 JTable

A JTable a táblázatot reprezentáló eszköz a swing-ben. Lehetőséget nyújt az adatok táblázatban történő megjelenítésére és szerkesztésére. Mint eszköz a JTable rendkívül sok lehetőséget nyújt a táblázat kialakítására, de ennek fejében nagyon komplex is, aminek köszönhetően a mögötte álló modelleket és a swing-ben található implementációit külön csomagba gyűjtötték (javax.swing.table).

A JTable megvalósítása mögött számos modell és osztály áll. Ezek közül az első a TableModel interfész. Ez a modell felelős a táblázatban szereplő adatok tárolásáért. Ennek a megvalósításához metódust specifikál a sorok és oszlopok számának lekérdezéséhez (getRow/ColumnCount), adott oszlop osztályának, illetve nevének index alapján történő visszaadására (getColumnClass/Name), TableModelListener-ek és a cellák közvetlen kezelésére. A cellák kezelése sor és oszlop index alapján történik és a beállíthatóságon és lekérdezhetőségen (set/getValueAt) túl lehetőség adódik a szerkeszthetőség állítására is (isCellEditable). A table csomagban található egy absztrakt megvalósítása AbstractTableModel és egy alapértelmezett a DefaultTableModel. A DefaultTableModel implementáció mögött egy Vector-okból álló Vector áll, ami a sorokat tartalmazza és az oszlopok neve is egy külön Vector-ban tárolódik.

A következő modell interfész az oszlopok kezeléséért felelős, de a tárgyalásához szükséges először az oszlopok megvalósításáról is ejteni néhány szót. A JTable oszlopait TableColumn példányok reprezentálják. A TableColumn felelős az oszlopok méretezéséért, illetve a megjelenítésének részleteiért is. Az oszlopok méretezését illetően megadhatjuk, hogy az oszlop átméretezhető-e (set/getResizable), illetve beállíthatjuk a minimális, maximális, optimális és aktuális szélességét is (set/getMin/Max/Preffered/-Width). A megjelenítés kezelésére saját magunk adhatjuk meg az oszlop fejéhez, illetve celláihoz a megjelenítésért felelős TableCellRender-t (set/getHeader/CellRenderer) és a cellák szerkesztéséhez használt TableCellEditor-t (setCellEditor). Megadható még az oszlopazonosító és a fejnek az értéke (set/getIdentifier és set/getHeaderValue). Alapértelmezve az azonosító megegyezik a fej értékével. A megjelenítésért felelős TableCellRenderer interfész, hasonlóan a ListCellRenderer-hez egyetlen metódusból áll, melynek a megjelenített komponenssel kell visszatérni a JTable, a cella értéke, a kijelöltség, a fókusz és a sor- és oszlopindex alapján. Alapértelmezett implementációja a DefaultTableCellRenderer. A TableCellEditor elvében megegyezik a TableCellRenderer-rel, csak a szerkesztésért felelős komponenst adja vissza.

A TableColumn megismerése után a következő modell, amely a JTable mögött áll a TableColumnModel, melynek a feladata a táblázat oszlopait reprezentáló TableColumn-ok kezelése. Az oszlopok kezelését index-ek alapján várja el. Az ehhez implementálandó metódusok az oszlopok hozzáadása, eltávolítása, áthelyezése (add/removeColumn, moveColumn), oszlop, oszlopok lekérése (getColumn/Columns), adott oszlop indexének lekérése azonosító vagy hely alapján (getColumnIndex, getColumnIndexAtX). Az oszlopok kijelöléséhez ListSelectionModel-t használ (set/getListSelectionModel), a kijelölt oszlopok lekérhetőek (getSelectedColumn/Columns) és a kijelölés engedélyezhető (set/getColumnSelectionAllowed). A megjelenítés terén ez a modell kezeli az oszlopok közötti margó (set/getColumnMargin) és itt kérhető le a teljes szélesség (getTotalColumnWidth). Továbbá még elvárja a TableColumnModelListener-ek kezelését. A javax.swing.table csomagban egyetlen implementációja az alapértelmezett DefaultTableColumnModel.

Mint az az előbbiekből is kiderült az oszlopok kijelölésért a TableColumnModel felelős. Ezzel szemben a sorszintű kijelölést, mivel az jelentősen gyakoribb, mint az oszlopszintű, felhozták a JTable szintjére, így azokhoz a JTable közvetlen tartalmazz egy ListSelectionModel-t (set/getSelectionModel). A JTable szintjén található még egy

setSelectionMode metódus is, amely a kijelölési politikát állítja be sor- és oszlopszinten. Ahhoz, hogy ez a kettő eltérő legyen közvetlen a ListSelectionModel-ekben kell azt beállítani. Mindkét szintre hat még a clearSelection eljárás, amely megszünteti a kijelöléseket, illetve a selectAll, amely mindent kijelöl. Lekérdezhető még, hogy mindekét szinten engedélyezett-e a kijelölés, azaz jelölhető-e ki cella (getCellSelectionEnabled), illetve szintenként is lekérhető az engedélyezettség (getRow/ColumnSelectionAllowed).

Az összetett és rugalmas táblázatok készítésénél fontos szerepe a van a JTableHeader osztálynak. Minden JTable egyetlen ilyen példányt tartalmazz (a JTableHeader is csak egyetlen JTable-höz tartozhat) és ez felelős az oszlopok drag&drop-jáért és a felhasználó által történő dinamikus oszlop átméretezésért. Ehhez ez az osztály külön kezek egy TableColumnModel-t, a fejekhez. A fontosabb metódusai a különböző rugalmasságot adó opciók engedélyezésére és azok megvalósítására szolgálnak. Ezek az átméretezésnél az engedélyezés (set/getResizingEnabled) és az átméretezendő oszlop kezelése (set/getResizingColumn). A drag&drop-nál pedig az engedélyezés (set/getReorderingAllowed), az oszlop (set/getDraggedColumn) és az áthúzás távolságának (set/getDraggedDistance) kezelése. Ezeken túl még számos metódust biztosít különféle Event-ek generálásához. Fontos megjegyezni, hogy a JTableHeader nyújtotta lehetőségek használata nem mindig ajánlott. Bizonyos táblázatoknál a drag&drop lehetőség gyakran csak a felhasználót zavarja meg, illetve lehetnek oszlopok melyeknél elvárt, hogy fix méretűek legyenek, így ezeknél a funkcióknál mindig érdemes megfontolni, hogy mikor és milyen mértékben engedélyezzük őket

Hasonlóan a JList-hez a JTable is implementálja a Scrollable interfészt, így JScrollPane-be helyezve, a görgethetőség könnyen megvalósítható.

Az eddig leírt modellek metódusai közül legtöbbel rendelkezik a JTable, melyeket közvetlen delegál a felelős modellnek. További metódusai még az osztálynak a nézetbeli és modellbeli oszlopindexek közti konverzió (convertColumnIndexToModel/View), a drag&drop engedélyezése (set/getDragEnabled), lekérhetőek az éppen szerkesztett cella indexei (set/getEditingRow/Column), illetve, hogy van-e egyáltalán ilyen cell (isEditing). A megjelenítésnél még állítható a sormagasság (set/getRowHeight) a rácsháló színe és kirajzoltsága (set/getGridColor, set/getShowHorizontal/VerticalLines, setShowGrid).

11.4 JTree

A hierarchikus adatstruktúrák reprezentálására szolgáló eszköze a swing-nek a JTree. A JTree lehetőséget nyújt ezeknek az adatszerkezetnek a megjelenítésére, szerkesztésére és a bennük történő navigálásra. A JTable-höz hasonlóan ez is egy nagyon komplex eszköz, így külön csomagot szántak neki a swing-en belül (javax.swing.tree).

A JTree mögött is több modell áll. A fák jellegzetes tulajdonságainak és viselkedésének a megvalósításáért a TreeModel a felelős. Ez a modell a set/getModel metódussal állítható be. A TreeModel által elvárja fában való navigálásához szükséges gyökérelem (getRoot), illetve megadott szülő gyermekének lekérését (getChild) index alapján. Az indexalapú gyermekkezelés miatt szükség van egy gyermekszámlálóra (getChildCount), illetve lekérhető az indexe egy elemnek adott szülőn belül (getIndexChild). Ezentúl a navigációhoz szükség lehet még arra, hogy eldönthessük adott elemről, hogy levélelem-e. A TreeModel kezeli még TreeModelListener-eket is (add/removeTreeModelListener). A navigációs metódusokon túl a TreeModel interfész még az elemek módosításához is elvár egy metódus (valueForPathChanged), amely a megadott TreePath-hez új értéket rendel. A swing-ben egyetlen a implementációja található meg a TreeModel-nek, ez az alapértelmezett DefaultTreeModel. Ez a megvalósítás a fa pontjaihoz úgynevezett TreeNode-kat használ. A DefaultTreeModel-ben az 1.4 óta lehet a gyökérelem null.

A TreeModel-nél szó esett egy a javax.swing.tree csomagban található interfészről és osztályról is, melyek a faszerkezet megvalósításában játszanak fontos szerepet. Ezek a TreePath és a TreeNode.

A TreeNode a fa egy csúcsát reprezentálja és, mint interfész az ezekhez minimálisan szükséges viselkedésmódok implementálását várja el. Ezek a szülő- (getParent), illetve index alapján az adott gyermekcsúcs lekérése (getChildAt), a gyermekek számának visszaadása (getChildCount) a levélelemség eldöntése (isLeaf), adott gyermek indexének megállapítása (getIndex), illetve az összes gyermek Enumeration-ként történő lekérése (children). A TreeNode-nak létezik egy MutableTreeNode nevezetű szubinterfésze. Ez az interfész már nem csak a navigáláshoz, hanem a módosításhoz is elvár bizonyos metódusokat. Ezek az adott indexen új gyermek hozzáadása (insert), index alapján, vagy direkt módon megadott gyermek eltávolítása (remove), saját magának az eltávolítása (removeFromParent) a szülőből, új szülő megadása (setParent), illetve a csúcs értékének megváltozatása (setUserObject)

A TreePath egy csúcshoz vezető utat tartalmazó osztály. Ennek megfelelően képes visszaadni az út által tartalmazott elemeket (getPath), illetve a szülő utat, ami az utolsó elem nélküli utat takarja (getParentPath), ennek a párját az új gyermekkel szaporított utat (pathByAddingChild). Le lehet még kérni az út utolsó elemét (getLastPathComponent), az adott elemet index alapján (getPathComponent) és az út hosszát (getPathCount). Ezeken túl még újradefiniálja az equals-t, ami itt az út elemeinek és a sorrendjüknek egyezése esetén igaz, plusz el tudja dönteni, hogy leszármazottja-e az aktuálisnak egy adott TreePath.

Ahogy azt megszokhattuk az adatstruktúrákat reprezentáló swing eszközöknél a fa kijelöléséért is egy külön modell a felelős. Itt a TreeSelectionModel interfész látja el ezt a feladatot. Ez számos a kijelölésre vonatkozó metódust vár el. A modell három kijelölési módot ismer, ezek a SINGLE_TREE_SELECTION, a CONTIGUOUS_TREE_SELECTION és a DISCONTIGUOUS_TREE_SELECTION. Ezek közül az első egyszerre csak egy útvonal kijelölését engedélyezi, míg a második többnek is, de azoknak összefüggőnek kell lennie, míg a harmadik tetszőleges kijelölést engedélyez. A kijelölési mód a set/getSelectionMode metódussal állítható. Azon túl, hogy a kijelöléshez utakat használ, mint elemeket, a metódusai közel ugyanazokat a műveleteket takarják le, mint az eddigi SelectionModel-ek. A fontosabbak ezek közül út, utak hozzáadása a kijelöléshez, az eltávolítása onnan és a beállítása (add/remove/set/getSelectionPath/Paths), lekérhető a legutoljára hozzáadott út (getLeadSelectionPath). Megszüntethető minden kijelölés (clearSelection), lekérhető a kijelölések száma (getSelectionCount), illetve, hogy van-e egyáltalán kijelölt rész (isSlectionEmpty). Az eddig leírtak mellett a TreeSelectionModel ad egy másik megközelítést is a fák kijelölésének a kezelésére, ez a sor alapú megközelítés, aminek az igazi szerepe a megjelenítésnél jelenik meg. A soros kezelés alatt a fa kibontott ágai alapján történő listaszerű megjelenését értjük. A soros megközelítésben lekérhető a listában a legutoljára hozzáadott, a legkisebb, illetve legnagyobb kijelölés indexe (getLead/Min/MaxSelectionRow), az összes kijelölt sor (getSelectionRows), illetve megállapítható, hogy adott sor ki van-e jelölve (isRowSelected). A sorok és utak közötti megfeleltetést egy úgynevezett RowMapper végzi. A RowMapper egy interfész, amely egyetlen metódus implementálását várja el, mégpedig a getRowsForPaths-t, amely megadott TreePath tömbhöz a neki megfelelő sorindex tömbbel tér vissza. TreeSelectionModel RowMapper-je a set/getRowMapper eljárással állítható. Az út és sor közötti megfeleltetések pedig a resetRowSelection eljárás segítségével bármikor újraszámoltathatóak.

A fák gyakran nagyon nagy elemszámmal rendelkeznek, például a fájlrendszerek. Emiatt a fákban történő navigálásban, mely listaszerűen történik, fontos a csomópontok kibontása és összezárása. Ezek a műveletek a fa modelljét illetően nem rendelkeznek akkora szereppel, hogy azt a TreeModel szintjére kelljen vinni, így ez a JTree-ben közvetlenül jelenik meg. A JTree metódusokat szolgáltat adott csomópont kibontására és összezárására út, illetve sor alapján (expand/collapsePath/Row), illetve le is kérhető az állapotuk (isExpanded/Collapsed). A sorok és utak közti konverzióra a getPathForRow és a getRowForPath függvények használhatóak.

A megjelenítésnél a fa kibontott elemeit listaszerűen sorokként látjuk. Ezeknek az elemeknek a megjelenítését egy TreeCellRenderer, illetve szerkesztés esetén egy TreeCellEditor implementáció végzi, ami a set/getCellRenderer/Editor-ral állítható be. A megjelenítés szerint három állapota lehet egy csomópontnak: rejtett, látható és megjelenített. A rejtett csomópont szülője össze van zárva, míg a látható ki van bontva, a megjelenített pedig egy olyan látható, amely ki is van rajzolva éppen. Ezeknek a kezelésére a JTree definiál pár metódust. Adott csúcs láthatósága állítható az útjával azonosítva a make/isVisible-lel. A megjelenítettsége a gyökérnek az is/setRootVisible, míg adott csúcsé s scrollPath/RowToVisible eljárással állítható. A megjelenített sorok lekérhetőek (set/getVisibleRowCount).

A getToolTipText felül lett definiálva e JTree-ben, hogy a renderer-ek elemenként egyedi tooltip-et adhassanak. Állítható a sormagasság (set/getRowHeight, isFixedRowHeight), aminek az értéke ha 0 vagy kevesebb, akkor változó a sormagasság, egyébként a fix megadott érték.

A JTree implementálja a Scrollable interfészt, azaz JScrollPane-ben az elemeinek megfelelő inkrementumokkal működik majd és rendelkezik az ehhez szükséges metódusokkal.

A JTree természetesen számos a modellekhez delegált metódusokkal rendelkezik, amelyeket nem írok le külön, hiszen már valamilyen formában szerepeltek korábban.

A JTree rendelkezik még néhány tulajdonsággal, amelyekről még nem esett szó eddig. Ezek közül a fontosabbak a szerkeszthetőség fa szinten (is/setEditable) és út szinten (isPathEditable) és az éppen szerkesztettség (isEditing). A szerkesztés leállítható a stopEditing függvénnyel.

12. Összefoglalás

Ezen szakdolgozat keretein beül szó esett a Java GUI fejlesztésre adott lehetőségeiről. Ezen lehetőségek az 1.5-ös Java-ra a Swing keretein belül igen széleskörűre duzzadtak fel, így ennek a tárgyalásához kevés volt ez a dolgozat. Ennek következtében logikai meggondolások révén egy általános GUI megépítéséhez szükséges támpontokat nyújtotta a dolgozat.

A Swing az itt leírtakon kívül még lehetőséget nyújt a grafikai megjelenés abszolút testreszabásához. Ezt nevezték el plaf-nak, ami a Pluggable Look & Feel rövidítése. Ebben a grafikai megjelenítés abszólut átszabható.

Másik fontos téma, amiről csak érintőlegesen esett szó azok a különféle Listener-ek voltak. A GUI manapság eseményközpontúan működnek és a különféle események bekövetkezésétől függően történek a reakciók. De mivel az eseményorientált működés az egész Java-t átjárja, így ahelyett inkább a Swing egyedi viselkedésére összpontosított a dolgozat.

Összegezve a dolgozat először tárgyalta az egész Swing mögött álló működési alapelveket, mindezt összehasonlítva az elődjének az AWT-nek az elveivel (2. és 3. fejezet). Ezalatt értve a könnyű és nehézsúlyú komponensek előnyeit, hátrányait és a hierarchikus szerkezetükből összeálló grafikus felület logikai felépítését. Majd miután ennek az alapstruktúráját részletezte először a top-level, azaz a grafikus felület hierarchiájának alapját képező nehézsúlyú Swing komponenseket tárgyalta meg (4. fejezet). A top-level hordozók után következett az erre ráépítendő elemek közös őse és viselkedésüknek meghatározója a JComponent osztály elemzése (5. fejezet). Ezek után a különféle konkrét Swing eszközök tárgyalása következett, feladatuk és megvalósításuk alapján külön fejezetekre bontva. Itt kerültek sorra a nem top-level hordozók (6. fejezet), a különféle output, input és vezérlő komponensek (7. és 8. fejezet). Ezek megvitatása után sor került a megjelenítés során ezeknek az elrendezését és méretezését végző elrendező osztályokra a LayoutManager-ekre (9. fejezet). Végül az utolsó két fejezetben az eddigieknél komplexebb eszközök tárgyalása történt meg. Ezekhez az eszközökhöz legtöbbször az összetettségük miatt külön csomagok is társultak a Swing-en belül. Ilyen eszközök voltak a szövegszerkesztésre, szöveges adatbevitelre szolgáló eszközök, melyek a javax.swing.text csomagot kapták meg, mint külön csomagot és teljesen az egyszerű szövegdoboztól a különféle dokumentumok (html, rtf) kezeléséig terjedtek (10. fejezet). Illetve hasonlóképpen összetett eszközök voltak a különféle adatszerkezetek (listák, táblázatok, fák) kezelésére szolgáló osztályok és a hozzájuk tartozó csomagok (11. fejezet).

Ezeknek a használatához igyekezett ez a szakdolgozat egyrészről alapvető ismereteket nyújtani, másrészről gyakorlati szabályokat, melyekkel a grafikus felületen ezek az eszközök hatékonyabban működhetnek és ezáltal maga a GUI is használhatóbb és hatékonyabb lesz, mind felhasználói, mind pedig teljesítmény szempontból.

Köszönetnyilvánítás

Itt szeretnék köszönetet mondani az összes tanáromnak, akik előadásiakkal segítették szellemi épülésemet. Külön szeretném még megköszönni a segítségét Simon Gyula tanár úrnak, aki témavezetője volt ennek a szakdolgozatnak és dr. Juhász István tanár úrnak, akinek órái ennek a dolgozatnak is fontos alapokat szolgáltattak.

Irodalomjegyzék

- Java™ 2 Platform Standard Edition 5.0 API Specification – java.sun.com
- The Java™ Tutorials – java.sun.com
- Matthew Robinson & Pavel Vorobiev: Swing, 2nd Edition, Manning, 2005
- Brian Cole, Robert Eckstein, James Elliott, Marc Loy & David Wood: Java™ Swing, 2nd Edition, O'Reilly, 2002

Függelékek

1.1-es ábra: A főbb AWT és Swing komponensek osztályhierarchiája