Párbeszédablakok, vezérlő elemek

Cél: párbeszédablakok használata, vezérlő elemek megismerése.

Nyomógombok: az eddigiek során megismertük, külön nem foglalkozunk velük.

A programokban színekkel dolgozunk. Felhasználjuk az előzőekben elkészített `Szín` osztályt:

- A szín mellett nyilvántartjuk annak nevét is.
- A konstruktorral hozhatunk létre egy adott nevű színt.
- Lekérdezhetjük a szín nevét, magát a színt, és megadjuk a kiírási formátumot is (`toString`), ami a szín neve.
- A `toString` művelet akkor lehet hasznos, ha listákat, vagy comboboxokat akarunk ilyen elemekkel feltölteni, és nem adunk meg megjelenítési előírást. Ilyenkor ezt a műveletet használja a rendszer a megjelenítendő információ meghatározásához.

Az egyes vezérlő elemek használatát külön-külön párbeszédablakokban vizsgáljuk meg.

Modális párbeszédablakokkal foglalkozunk.

A legtöbb párbeszédablak tartalmazza az `OK` és a `Mégsem` nyomógombokat.

Ennek megfelelően célszerűnek tűnik egy olyan osztály létrehozása, amely ezt a folyamatot támogatja a gombok létrehozásával és az alapvető eseménykezelés elvégzésével.

OK és Mégsem gombot tartalmazó párbeszédablakok

Funkcionalitás:

- Az OK gomb megnyomása esetén ellenőrizni kell, illetve lehet hogy az ablak tartalma megfelelő-e, és ha a válasz igen, akkor le kell zárni az ablakot.
- A Mégsem gomb megnyomásakor esetlegesen vissza kell állítani a kezdeti értékeket, és le kell zárni az ablakot.

Ezek az eseménykezelők feladatai.

Az eseménykezelés miatt be kell vezetnünk két absztrakt műveletet:

- Az `OK` nyomógomb megnyomásakor szükséges ellenőrzések elvégzése és az ellenőrzés eredményének megadása.
- A `Mégsem` gomb aktivizálásakor szükséges teendők végrehajtása.

További művelet:

- A bezárást okozó gomb azonosítása.

Az azonosításhoz szükséges két konstans, amelyek megfelelnek az `OK`, illetve a `Mégsem` gomboknak.

Az absztrakt műveletek miatt a létrehozott `OKCancelDialog` osztály absztrakt lesz. Ebből kell származtatni a konkrét esetnek megfelelő párbeszédablakot.

Származtatás az `OKCancelDialog` osztályból

- Meg kell határozni (implementálni) a két absztrakt műveletet.
- A párbeszédablakot fel kell tölteni a megfelelő vezérlő elemekkel.
- A vezérlő elemekhez hozzá kell venni a két nyomógombot, pontosabban el kell helyeznünk a gombokat tartalmazó panelt.

OKCancelDialog osztály

- Az absztrakt osztályt a JDialog osztályból származtatjuk.
- Az osztály tartalmazza a kívülről használható osztályszintű konstansokat, illetve a származtatott osztályokban használható panelt, és gombokat.
- Események kezelése:
  - az absztrakt műveletek meghívása,
  - gombkód beállítása,
  - eltüntetés.

OKCancelDialog osztály (folyt.)

- Konstruktor:
  - Minden esetben megadjuk a keretet, ahonnan a párbeszédablakot aktivizáltuk, és az ablak címét. (Ezek segítségével lehet egy modális párbeszédablakot létrehozni.)
  - A konstruktorban létrehozzuk a paramétereknek megfelelő modális párbeszédablakot, és beállítjuk az alapértelmezett bezárási műveletet. (Az alapértelmezett az ablak elrejtése, ezért ez most tulajdonképpen felesleges.)
  - Ezután létrehozzuk a megfelelően paraméterezett gombokat, amelyeket egy panelban helyezünk el.
  - Az `OK` gombot állítjuk be alapértelmezett nyomógombként. (`Enter` billentyűvel aktivizálható.)
  - A `Mégsem` gombhoz hozzárendeljük az `Escape` billentyűt.

Az alkalmazás kerete

- A különféle párbeszédablakokat egy keretből fogjuk vezérelni.
- Erre szolgál a `Dialógusok` osztály.
- Ebbe kerülnek el a különböző dialógusok, és kezelésükhöz szükséges elemek, továbbá a menüpontok eseménykezelői.
- A teszteléshez a lehető legegyszerűbb felületet alakítjuk ki, nem hozunk létre eszköztárat, csak menüt.
- A színeket egy szövegszerkesztő mező háttér-, illetve betűszínében használjuk.
- Ebben a mezőben fogjuk naplózni a párbeszédablakok használatát.

Csúszka

- `SliderDlg` osztály egy csúszkát tartalmazó párbeszédablak.
- A csúszka aktuális értéke is jelenjen meg a csúszka fölött.
- A párbeszédablak megadásakor beállítható:
  - a csúszka intervalluma,
  - az aktuális érték,
  - és a fő értékek (a fő értékeket az értékkel és hosszabb vonallal jelöli a csúszka).
- A normál érték minden esetben 1.

Műveletek

- Az `OK` gomb mindig elfogadható (`processOK`).
- A `Mégsem` esetén semmit sem kell tenni (`processCancel`).
- Új műveletek:
  - A csúszka értékének lekérdezése a párbeszédablak lezárása után (`getValue`).
  - A csúszka értékének állítása (`setValue`) (a megjelenítés előtt).

Állapotváltozás

- Az érték megjelenítése miatt figyelni kell, hogy a csúszka értéke megváltozik-e.
- Ezért az osztály megvalósítja a `ChangeListener` interfészt.
- Ehhez a `stateChanged` műveletet kell implementálnunk:
  - a csúszka értékének lekérdezése,
  - a címke feliratának módosítása.
- Ahhoz, hogy a művelet meghívásra kerüljön az osztályt fel kell vennünk a csúszka állapotváltozásaira figyelők közé (a csúszka `addChangeListener` művelete).

Gombok

A `ButtonDlg` párbeszédablak checkbox és rádiógomb típusú gombokat tartalmaz.

A párbeszédablak segítségével határozhatjuk meg a szövegmező háttérszínét.

- Ha a checkbox-ot bekapcsoljuk a háttér a szövegszín inverze.
- Ha a checkbox-ot kikapcsoljuk, akkor a hátteret a kiválasztott rádiógomb határozza meg.

A rádiógombok megjelenítése speciális:

- a szöveg az adott szín neve,
- az ikon a színnel kitöltött kör, amelyben fekete pont jelzi a kiválasztottságot (fekete szín esetén a pont fehér).

Rádiógombok speciális megjelenítése

- A megjelenítési módot külön meg kell adnunk a gomb létrehozásakor az ikon, illetve a kiválasztottsági ikon specifikálásával.
- Erre szolgál a `gomb` függvény.
  - Ebben létrehozunk két képet (`BufferedImage`), amelyekben megrajzoljuk a kívánt ábrát, és a képekből létrehozzuk az ikonokat.
  - A rajzoláshoz szükséges a kép grafikus eszközkapcsolat leírója, amit a `getGraphics` függvény ad meg.
  - A rajzolás ezután értelemszerű.

Rádiógombok kezelése

- A létrehozott rádiógombokat egy csoportba kell foglalnunk.
- A rádiógombokat egy kerettel ellátott panelra kell helyeznünk.
- A gombokat nyilvántartjuk (`színgombok`) azért, hogy le tudjuk majd kérdezni a kiválasztottságukat.

Műveletek

- Az `OK` gomb lenyomása akkor fogadható el, ha van kiválasztott háttérszín.
  - Ez lehet a szövegszín inverze, vagy ha ez nincs kijelölve, akkor
  - egy színt kellett kiválasztani.
- A `Mégsem` gomb lenyomásakor nincs teendő.
- Szükséges az értékek lekérdezése
  - `inverzmód`,
  - `szín`, illetve
- a checkbox értékének megadása (`setValue`).

Lista

A `ListDlg` párbeszédablak egy listában tartalmazza a választható szövegszíneket.

A listában nem csak a szín nevét, hanem egy színnel kitöltött téglalapot is megjelenítünk.

- Egy speciális megjelenítési forma szükséges (`elemforma`), amit a `setCellRenderer` művelettel rendelhetünk a listához.
- (Ha csak szöveget akarunk a listában, erre nincs szükség, az elemek `toString` művelete szerinti szöveg jelenik meg.)

Ha azt akarjuk, hogy a lista egy elemén duplán kattintva az elemet válasszuk ki és zárjuk be az ablakot, akkor a listához fel kell vennünk egy egér esemény figyelőt, amelyben a duplakattintást figyeljük. (Az esemény megegyezik az `OK` gomb megnyomásával.)

Az `OK` gomb megnyomása elfogadható, ha van kiválasztott elem a listában.

Combobox

A `ComboDlg` párbeszédablakban egy szöveget adhatunk meg combobox segítségével.

- A combobox előre definiált értékei a színek nevei.
- A neveket most a szerkeszthetőség (pontosabban kiválaszthatóság miatt) `String` objektumokként helyezzük el a comboboxban az `addItem` művelettel.
- A combobox alapértelmezésben nem engedi az elemek szerkesztését. Ezt állíthatjuk a `setEditable` művelettel.
- A szerkesztett elemet a sorszerkesztő `getItem` műveletével kaphatjuk meg.
- A sorszerkesztőt a `getEditor` függvény adja meg.

Combolista

A combobox alapértelmezett viselkedésekor egy listából választhatunk elemet.

A `ComboListDlg` dialógus combolistájában a színeket jelenítjük meg, a nevük mellett a megfelelő színű körlemezzel.

- Az ikonok megjelenítése a listához hasonlóan érhető el.
- Most a `setRenderer` művelettel lehet a megfelelő megjelenítőt a combolistához rendelni.
- (Ha csak szöveg kell, akkor erre nincs szükség, az elemek `toString` művelete adja a szöveges információt.)

A kiválasztott elemet a `getSelectedItem` függvénnyel kaphatjuk meg.

Alapértelmezettként a lista első (nulla indexű) eleme kiválasztott. Így mindig lesz kiválasztott elem, azaz az `OK` gomb mindig elfogadható.

Sorszerkesztő

Egysoros szöveg bevitelét támogatja a `EditDlg` párbeszédablak.

Ebben egy `JTextField` objektummal valósítjuk meg a szerkesztőt.

- A szerkesztett szöveget a `getText` művelettel kérdezhetjük le.
- (A kezdeti szöveget a `setText` művelettel lehetne beállítani, mi ezt nem használjuk, így üres szöveg lesz kezdetben a tartalom, a későbbiekben pedig a párbeszédablak megelőző lezárásakor tartalmazott szöveg.)
- A soreditort szokásos módon használhatjuk szerkesztésre, értelmezettek a vágólapot használó műveletek: kivágás, másolás, beillesztés.

Spinner

A `SpinnerDlg` párbeszédablakban egy spinner segítségével változtathatjuk meg ugyanazt az értéket, mint a csúszkával.

- A spinner létrehozásakor megadhatunk egy modellt, ami leírja a spinner értéktartományát, aktuális értékét, és a rákövetkezés módját.
- Ha nem adunk meg modellt, egész számokat enged meg a tartomány korlátozása nélkül.
- Meg lehet adni egy listát, illetve tömböt (`SpinnerListModel`), ekkor az értékek értelemszerűek.
- Választható számokat tartalmazó modell (`SpinnerNumberModel`) a megfelelő beállításokkal.

Nekünk most ez utóbbira van szükségünk.

Ha a szerkesztőbe érvénytelen értéket írunk, akkor az `Enter` billentyű lenyomásakor nem fogadja azt el. Ha az `OK` gombot nyomjuk meg, az utolsó érvényes értéket veszi figyelembe!

Táblázat

A `TáblázatDlg` párbeszédablakban egy táblázatban jelenítjük meg a színek jellemzőit: a nevet, és a komponens (rgb) értékeket.

- Egy táblázat megjelenítésére szolgál a `JTable` osztály.
- A táblázat tartalmát egy modell segítségével adhatjuk meg, ami az adatokat szolgáltatja.
- Ezt legegyszerűbben az `AbstractTableModel` absztrakt osztály felhasználásával tehetjük meg.
- A táblázat oszlopai egyenlő szélességűek. Ha ettől el szeretnénk térni, akkor a táblázat oszlop modelljét kell módosítanunk egy `DefaultTableColumnModel` osztályból származtatott osztály példányával. Mi most ettől eltekintünk.

Táblázat modell

Az `AbstractTableModel` példányosítása (`táblamodell`) során meg kell adnunk a következő műveletek jelentését:

`getRowCount` a táblázat sorainak száma,

`getColumnCount` a táblázat oszlopainak száma,

`getValueAt` a táblázat adott sorának és oszlopának értéke.

Táblázatok tulajdonságai

- Ha az oszlopok neveit is meg akarjuk adni, akkor a `getColumnName` függvényt kell felüldefiniálnunk.
- Ha görgetni akarjuk a táblázatot, akkor azt egy `JScrollPane` objektumba kell ágyaznunk. (Ha ezt nem tesszük meg, akkor külön gondoskodni kell az oszlopcímek kiíratásáról.)
- Táblázatok tartalma változtatható (mi most csak megjelenítést használtuk).
  - (`isCellEditable`, `setValueAt`, `fireTableDataChanged` ... műveletek a szerkesztéshez.)
- Táblázat elemeinek megjelenítése is szabályozható, ekkor az oszlop `setCellRenderer` műveletével állítható be egy `DefaultTableCellRenderer` típusú objektum.

Az utolsó 3 lehetőségre (az oszlopok szélességének szabályozásával együtt) később visszatérünk.

Egér kezelése

A felület elemei közül több komponens bizonyos formában kezeli az egeret (pl. gombok, listák a kattintást, görgetést a görgetősávok).

Ha eltérő kezelés, illetve új esemény figyelése szükséges, akkor egéresemény figyelőt kell a komponenshez rendelni:

`addMouseListener` az egérgomb (lenyomás, felengedés, kattintás) és területet váltó (belépés, elhagyás) eseményeinek kezelését végző objektum, amely megvalósítja a `MouseListener` interfészt;

`addMouseMotionListener` az egér mozgását, húzását kezelő objektum, amely megvalósítja a `MouseMotionListener` interfészt;

`addMouseWheelListener` a görgető gomb tekerését kezelő objektum, amely megvalósítja a `MouseWheelListener` interfészt.

`MouseAdapter` osztály

Mindhárom interfészt megvalósítja (üres műveletekkel).

Ebből származtathatunk kezelőt, ha nem akarunk mindenhez megvalósítást írni, illetve több interfész műveleteire van szükségünk, és nem akarunk külön objektumokat felvenni.

Az `EgérTeszt` projekt szemlélteti az egér kezelését.

- Felveszünk egy komponenst (`terület`), amelyen figyelni akarjuk az egeret, és az eseményeket három címke feliratában jelenítjük meg: a pozíciót, a gombot, és az eseményt (kattintás, mozgatás, ...). Görgetés esetén a gomb helyett a görgetés iránya jelenik meg.
- A komponenshez felveszünk egy objektumot (`egér`), amelyben figyeljük a bekövetkezett eseményeket. Ehhez az interfészek megfelelő műveletei szükségesek, amelyeket a `MouseAdapter` osztályhoz képest felüldefiniálunk. A műveletek nevei magukért beszélnek.
- A görgetésen kívül minden műveletnek egy `MouseEvent` típusú paramétere van, amely tartalmazza az eseménnyel kapcsolatos információkat: pozíció, gomb kódja ....

- Görgetés esetén egy `MouseWheelEvent` objektumot kapunk, amelytől lekérdezhető például a görgetés iránya. (Figyelem: -1 vagy $+1$ értéket kapunk, és minden egyes elforgatás hatására meghívódik ez a művelet, tehát 5 „kattanásos” lefelé görgetésnél ötször hívódik meg $+1$ értékkel.)
- Az események kezelése a megfelelő címkék feliratának változtatását jelenti esetünkben.