Rajzoló program

A felhasználói felülettel kapcsolatos elemek összefoglalásaként elkészítünk egy egyszerű rajzoló programot (`Rajz`). Ennek során újabb rajzolási lehetőségeket vizsgálunk meg, valamint kitérünk két „könytári” párbeszédablak használatára: a fájl, illetve szín kiválasztó párbeszédablakokat ismerhetjük meg.

Az eddigiekből felhasználjuk a

- `StatusBar` és
- `OKCancelDialog` osztályokat.

A `StatusBar` osztályt módosítjuk. Bevezetünk egy új műveletet, amellyel egy mező ikonját lehet beállítani.

```
/**
 * Adott cella ikonjának állítása
 * @param index a cella indexe
 * @param icon a cella ikonja
 */
public void setIcon(int index, ImageIcon icon)
{
    cellák[index].setIcon(icon);
}
```

1. Követelmények

- A programmal hozhassunk létre egyenes szakasz, téglalap és ellipszis alakzatokat!
- Az alakzatokat egy ábrához vehessük fel!
- Lehessen új ábrát kezdeni, elmentett ábrát betölteni, ábrát elmenteni!
- A rajzot lehessen nagyítani!
- A státuszsor tartalmazza az aktuális alakzat jellemzőit: típust, a vonalvastagságot, a kitöltöttséget, illetve átlátszóságot, a nagyítás mértékét, az első és második végpont koordinátáit!

A soron következő (új) alakzat jellemzőit be lehet állítani. Ezek:

- a vonal vastagsága,
- a vonal stílusa (folytonos, szaggatott, pontozott, szaggatott–pontozott),
- a kitöltöttség,
- a vonal színe és
- a kitöltési szín.

Az alakzatokat két ponttal adhatjuk meg, amelyket az egérrel határozhatunk meg.

A gomb lenyomása adja meg az első végpontot, a gomb felengedése a második végpontot adja meg.

2. Az alkalmazás kerete

A `Rajz` osztály adja meg a program keretét, és ez lesz egyben a fő osztály is, azaz ebben lesz a `main` függvény.

Hivatkozik a rajzolás területére (`Terület` osztály `terület` objektuma), nyilvántartja az aktuális rajzolási attribútumokat, az aktuális alakzatot, illetve megfelelő ikonokat. Ezeken kívül az eseménykezelőket is itt adjuk meg.

Az alakzatokból mindig egy prototípusra mutatunk, és amikor egy új alakzatot kell felvennünk, akkor az aktuális prototípust másoljuk le a megfelelő rajzolási attribútumok megadásával.

2.1. Események kezelése

A rajzolási jellemzőket állító események kezelése a színek, a vonalvastagság és a nagyítás állításán kívül értelemszerű.

Színek esetén a megfelelő könyvtári párbeszédablakot (`JColorChooser`) kell meghívni. A visszatérési érték a kiválasztott szín, illetve `null`, ha nem választottak ki színt.

A vonalvastagságot, illetve a nagyítást egy-egy saját párbeszédablakban adhatjuk meg csúszka segítségével.

A fájlműveleteket a rajzolás területének kell kezelnie, akárcsak az alakzatok listázását, ezért ilyenkor a terület objektum megfelelő műveleteit kell meghívni. Az eredmény figyelembevételével a keret (cím), illetve státuszsor (nagyítás) jellmezőit kell változtatni.

Kilépéskor ellenőrizzük, hogy az utolsó mentés óta változott-e a rajz, és ha igen, akkor erre figyelmeztetjük a felhasználót, felajánlva a mentési lehetőséget.

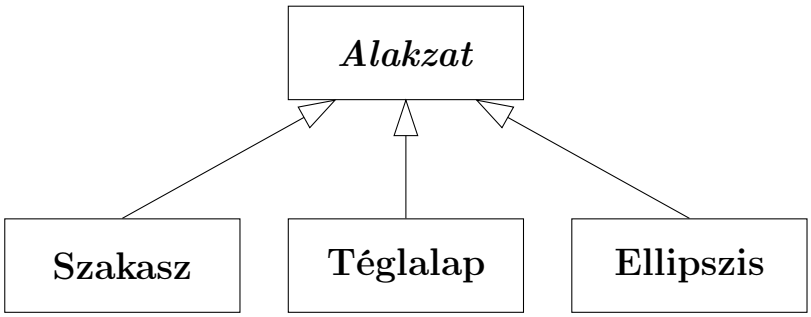
2.2. Szolgáltatások

Két műveletet vezetünk be, amelyeket a rajzolási terület használ:

- a státuszsor elemeinek változtatása (`felirat`), az egér helyzetének kiíratása miatt;
- az aktuális (prototípus) alakzat alapján megadott ponttal és aktuális jellemzőkkel alakzat létrehozása (`alakzat`).

3. Alakzatok

A rajzokban használt alakzatokat – szakaszt, téglalapot, ellipszist – egységesen akarjuk kezelni. Ezért bevezetünk egy absztrakt `Alakzat` osztályt, amelyből a konkrét típusokat származtatjuk.



3.1. `Alakzat` osztály

A fájlműveletek (betöltés, mentés) miatt megvalósítja a `Serializable` interfészt.

Tartalmazza a különböző vonalstílusoknak megfelelő konstansokat.

Attribútumai az alakzatok rajzolási jellemzői.

A vonalstílus numerikus értékként, és nem kiírandó `BasicStroke` komponensként is szerepel.

(A `BasicStroke` osztály nem valósítja meg a `Serializable` interfészt, ezért objektumai nem írhatók, olvashatók egyszerűen. Ezért nem mentjük ezt el, hanem a numerikus értéket írjuk ki, és betöltéskor gondoskodunk arról, hogy létrejöjjön ez a komponens.)

3.1.1. Műveletek

- Konstruktor a megadott jellemzőket beállítja, és létrehozza a vonalstílust.
- `vég`: a végpontot állítja.
- `másol`: absztrakt, a másolás létrehozásának felülete.
- `rajzol`: absztrakt, az alakzat rajzolásának felülete.
- `név`: absztrakt, az alakzat típus nevének lekérdezési felülete.
- `ikon`: absztrakt, az alakzat típus ikonjának lekérdezési felülete.
- `keret`: belső, a befoglaló téglalap meghatározása.
- `vonalstílus`: belső, a vonalstílus létrehozása a többi jellemző alapján.

3.2. `Szakasz` osztály

Tartalmazza az osztály összes objektumára vonatkozó típus ikont.

Minden alakzathoz tárolja a megjelenítéshez szükséges elemet, amit nem mentünk fájlba.

A konstruktorban az ős osztály konstruktorával beállítja a jellemzőket, és létrehozza a megjelenítési elemet.

Értelemszerűen megvalósítja az `Alakzat` osztály összes absztrakt műveletét.

Megadja a betöltés műveletét (`readObject`), amelyben az alapértelmezett olvasás mellett a nem mentett komponenseket is be kell állítani.

3.3. `Téglalap`, `Ellipszis` osztályok

A `Szakasz` osztálynak megfelelően kell ezeket is származtatni az `Alakzat` osztályból.

4. Rajzterület

A `Terület` osztály kezeli a rajzolási területet: felelős az alakzatok kezeléséért, megjelenítéséért. Az osztályt a `JPanel` osztályból származtatjuk, és megvalósítjuk vele a görgetéshez szükséges `Scrollable` interfészt.

4.1. Attribútumok

- hivatkozik a keretre, amit szülség esetén értesít;
- megadja az aktuális rajz méretét, kiterjedését;
- tárolja a rajz alakzatait;
- külön hivatkozik az éppen rajzolás alatt álló, változó alakzatra;
- tartalmazza a nagyítás értékét;
- megadja, hogy változott-e a rajz.

Ezenkívül hivatkozik egy fájlkiválasztáshoz szükséges párbeszédablakra, illetve tartalmazza az egér kezeléséért felelős objektumot.

4.2. Műveletek

A konstruktorban az attribútumokat kell létrehozni, illetve megfelelő kezdőértékkel ellátni.

A megjelenítés (`paintComponent`) során a tárolt alakzatokat és az aktuális alakzatot kell kirajzolni a `rajzol` művelet segítségével.

A nagyítás változása az attribútum állításán kívül a megjelenítéshez használt méret változását is jelenti, amiről a felületet értesíteni kell.

Új rajz kezdésekor meg kell kérdezni a méretet, a jelenlegi tartalmat törölni kell, a nagyítást alapállapotba hozni, és jelölni, hogy a rajzon nincs változás. A méret meghatározásához bevezetjük a `DimDlg` osztályt

Rajz betöltésekor a fájlválasztó párbeszédablak beállítása után, annak segítségével ki kell választani a fájlt. Ha ez megvan, akkor a fájlból be kell olvasni a méretet és az alakzatokat. Ha ez sikerült, akkor a nagyítást alaphelyzetbe kell állítani, és jelölni, hogy nincs változtatás.

Mentéskor a fájlválasztó párbeszédablak beállítása után, annak segítségével ki kell választani a fájlt. Ha ez megvan, akkor a fájlba ki kell írni a méretet és az alakzatokat. Ha ez sikerült, akkor jelölni kell, hogy nincs változtatás.

4.3. Egérkezelés

A nagyítás miatt az egéresemény koorditátáit konvertálni kell, erre szolgál az `alakít` művelet, ami a konvertált koordinátát a hely attribútumban tárolja. Minden esemény feldolgozása tartalmazza ezt a konverziót.

- Mozgatás esetén a keret státuszsorát kell módosítani.
- Húzás esetén az aktuális alakzat végpontja változik, újra kell rajzolni az ábrát, és a keret státuszsora változik.
- Gomb lenyomása esetén, új alakzatot kell létrehozni a megfelelő ponttal, újrarajzolni az ábrát, és a státuszsort változtatni.
- Gomb felengedésekor az aktuális alakzatot tárolni kell, újrar kell ajzolni az ábrát, és a státuszsort kell állítani.

5. Párbeszédablakok

A programban felhasználtunk három saját, és két könyvtári párbeszédablakot.

A saját párbeszédablakok megvalósításához felhasználtuk a már ismert `OKCancelDialog` osztályt.

5.1. `ZoomDlg`

Egy egyszerű csúszkát tartalmazó párbeszédablak, amely megfelel az előző órán megismert csúszkát tartalmazó dialógusnak.

5.2. `VastagságDlg`

Ebben az esetben is egyetlen csúszkát használunk, azonban a csúszka értékeinek értelmezése nem egész szám, hanem valós. Ezért a fő értékek címkéit külön meg kell adnunk egy `Hashtable` objektumban (`setLabelTable` művelet), és a csúszka egész értékeit valósakká alakítanunk érték lekérdezésekor, és a fordított konverziót kell elvégeznünk érték beállításakor.

5.3. `DimDlg`

Két címkézett soreditor használunk a szélesség és magasság bekérésére. Az `OK` gomb megnyomásakor ellenőrizni kell, hogy a soreszerkesztők tartalma egész szám-e (kivételkezelés), illetve megfelelő érték-e.

5.4. `JColorChooser`

Az osztály statikus `showDialog` műveletét hívjuk a párbeszédablak megjelenítéséhez. (Nem akarunk minden híváskor új objektumot létrehozni.) A művelet paraméterei: a hívó komponens, a dialógus címe, és a kezdeti szín, amit induláskor mutat a párbeszédablak. A függvény visszatérési értéke a kiválasztott szín, illetve `null`, ha a mégsem opciót választjuk.

5.5. `JFileChooser`

Fájlok kiválasztására szolgáló párbeszédablak. Beállítható jellemzők:

- a kezdő mappát, ahonnan a fájl kiválasztása indul;
- fájlok és könyvtárak közül mik jelenjenek meg;
- milyen fájlok jelenjenek meg, amihez egy `FileFilter` típusú szűrő objektum kell;
- mi legyen a kiválasztó gomb neve (jelentése), tulajdonságai.

Lehet külön megnyitási, illetve mentési párbeszédablakot megjeleníteni.

A `FileFilter` objektumban meg kell adni, hogy milyen fájlokat fogadunk el, és milyen leírás tartozzon ezekhez. Egy párbeszédablakhoz több szűrőt is fel lehet venni (`addChoosableFileFilter` művelet), illetve le lehet tiltani az összes lehetőséget tartalmazó szűrőt.

Egyszerűbb lehetőség:

```
FileFilter filter = new FileNameExtensionFilter("Rajz fájl", "rajz");
```

Átneveztük az elfogadó gombot (Open helyett Megnyitás, illetve Save helyett Mentés), de az ablak többi felirata még angol. Ezeket is át lehet nevezni, de ezt nekünk kell megtennünk. Vezessük be erre az `átnevez` műveletet.

Figyelem, a párbeszédablak nevét állítsuk be külön a megnyitás előtt, és ott ne adjunk meg címet! (Ha szerepel cím a megnyitáskor, akkor nem minden mezőt ír át.)

Ezek a változtatások láthatóak a `Rajz2` projektben.