Harmadik programozás házi feladat

Math Function Visualizer: Dokumentáció

Adatok:

- Név: Morvai Barnabás
- Neptun: E0HB0N
- Státusz: Kész
- Dátum: 2024.11.25.

Program leírása

A kész alkalmazás a specifikáció szerint lett megvalósítva. Képes matematikai függvények megjelenítésére, azok elmentésére, majd a mentett állapot visszatöltésére.

Az ablak

A megnyíló ablak egy *JFrame*-ből leszármazó saját *Frame* osztály. Azért származtattam le, mert így konstruktoron belül be tudom állítani minden paraméterét. Ebben az osztályban találhatóak meg a grafikus elemek példányai, mint a kirajzolt függvények, az oldalsó panel és az ablak tetején található menü. Ez az osztály kezeli a függvények hozzáadását és eltávolítását, valamint a menüsorban található mentés, betöltés funkciót mivel ezek a *Frame*-ben található függvényeket és a hozzájuk tartozó grafikus elemeket módosítják.
Az ábrázolt függvényeket úgy lehet jobban megvizsgálni, ha vagy közelebbről vagy éppen távolabbról nézzük őket, amit az egér görgőjével lehet vezérelni. Ezért ezt a funkciót az ablak implementálja a *MouseWheelListener* interfészből és definiálja, hogy mi történjen ha megmozdul a görgő.

public class Frame extends JFrame implements MouseWheelListener

Függvények

Minden függvény kirajzolásáért a *java.awt* könyvtárban található *Polygon* felel, ami úgy működik, hogy egy ***x*** ponthoz rendel egy ***y*** koordinátát, amit később majd egy *drawPolyline* függvény ki fog tudni rajzolni bármilyen *JPanel*-re. Kívülről nézve, ha ki akarunk rajzolni egy függvényt elég csak a *Polygon* objektuma, nincs szükség arra a tudásra, hogy az most a függvényeken belül éppen melyik. Ezért minden függvény egy absztrakt *Function* osztályból származik le.

public abstract class Function implements Serializable
    public class LinearFunction extends Function
    public class PowerFunction extends Function
    public class ExponentialFunction extends Function
    public class Sinus extends Function
    public class Cosine extends Function

A Function osztály implementálja a Serializable interfészt, mert ez az aktuális állapot elmentéséhez elengedhetetlen. Mivel a különböző függvények ebből származnak le így ők is meg fogják kapni ezt a tulajdonságot.

Először úgy gondoltam, hogy a *Polygon* kiszámítása majd absztrakt lesz és a függvények külön külön csinálják majd, de belegondolva minden függvénynél ugyanúgy működik, kiszámolom az ***x*** és ***y*** koordinátát. Ebből kiindulva a *calculatePolygon* függvény nem absztrakt, hanem a *Function* részre, ami

viszont absztrakt, hogy az egyes függvények, adott pixel-hez a képernyőn milyen ***x*** és milyen ***y*** koordinátát rendelnek. Így van két függvény, a *calculateXCoordinate* és a *calculateYCoordinate* és minden leszármazott függvény csak ezeket definiálja felül.

```
protected abstract float calculateXCoordinate(int position)
protected abstract float calculateYCoordinate(int position)
```

Első nekifutásra minden függvény egy *JPanel*-ből származott le és a *paintComponent* rögtön ki is rajzolta a függvény *Polygon* objektumát, de így nem valósult meg az, hogy a kirajzolni kívánt objektumok önmagukban is megállják a helyüket. Így viszont ezt a függvény osztályt máshol is lehetne alkalmazni hiszen nem egy grafikus elem.

Az oldalsó menü

Az oldalsó menü, avagy panel a *Frame* része és az ablak jobb oldalán foglal helyet, a *Frame* BorderLayout.*EAST* részén, rálógva a függvényekre, amik a BorderLayout.*CENTER* részen vannak.

```
sideMenu = new SideMenu(this)
this.add(sideMenu, BorderLayout.EAST)
layeredPane.add(coordinateSystem)
this.add(layeredPane, BorderLayout.CENTER)
```

Leglényegesebb funkciója az új függvények hozzáadása, ami gombokkal történik. Minden függvény típushoz tartozik egy gomb, amit megnyomva felugrik egy ablak, ahol meg lehet adni az adott függvény paramétereit. Az oké gombra kattintva az ablak bezárul és a paramétereknek megfelelő függvény létrejön a képernyőn.

```
JPanel buttonPanel = new JPanel()
String[] buttons = new String[] {
        „Add Linear Function”,
        „Add Power Function”,
        „Add Exponential Function”,
        „Add Sinus Function”
        „Add Cosine Function”
}
```

Emellett a hozzáadó gombok alatt helyet kapott egy lista, ami gombokból áll és egy-egy gomb a képernyőn látható függvények egyikét reprezentálja, a gomb lenyomásával a gombhoz tartozó függvény törlődik a modellből. A gombokon található szöveg a ábrázolandó függvény típusának megfelelő nevet kap és a színe pedig a tényleges függvény színével megegyező.

```
for (Function function : frame.functions.keySet()) {
        Button button = new Button(function.getName(), buttonWidth, function.color);
        button.addActionListener(new ButtonListener(frame, buttonsOfFunctions);
        buttonsOfFunctions.put(button, function);

        functionPanel.add(button);
}
```

Ez a részlet az *updateMenu* függvény része, ami update-notify rendszer megvalósítás egy fontos része. Ezek a gombok tartalmaznak egy *ButtonListener*-t, amik *ActionListener* interfészt implementálnak. Ez a Listener megkapja az ablak referenciáját, a függvényeket és azok gombjait tartalmazó *HashMap*-et.

```
public void actionPerformed(ActionEvent event) {
        Button clickedButton  = (Button) event.getSource();
        List<Map.Entry<Button, Function>> toRemove = new ArrayList<>();

        for (Map.Entry<Button, Function> entry : buttons.entrySet()) {
                Button button = entry.getKey();

                if (clickedButton.equals(button))
                        toRemove.add(entry);
                }
        }

        for (Map.Entry<Button, Function> entry : toRemove) {
                frame.removeFunction(entry.getValue());
                buttons.remove(entry.getKey());
        }
}
```

Ha egy ilyen gomb megnyomásra kerül azt a ButtonListener érzékelni fogja. A *getSource* metódussal lekérdezi, hogy melyik gombot nyomták meg, a *HashMap*-ből kikeresi, hogy ahhoz a gombhoz melyik függvény tartozik és az ablak referenciáján keresztül meghívja a *removeFunction* metódust és átadja neki a függvényt, amit az ablak a saját implementációjának megfelelően töröl a képernyőről. Viszont ha ez ennyi lenne, akkor a gomb amit megnyomtunk ugyanúgy ott maradna viszont törölt objektumra mutatna, a gomb ami nagy hiba lenne. Ezért a függvényt törlő metódusban a törlés után meg lesz hívva az oldalsó menü *updateMenu* metódusa, ami az ablakban található függvényeket listázza ki gombokként és adja hozzájuk a megfelelő függvényt és Listener-t, hogy működjön a törlés.

```
public abstract class PopupWindow extends JDialog
        public class LinearWindow extends  PopupWindow
        public class PowerWindow extends  PopupWindow
        public class ExponentialWindow extends  PopupWindow
        public class SinusWindow extends  PopupWindow
        public class CosineWindow extends PopupWindow
```

A függvényt hozzáadó gombok megnyomására megnyíló ablakok hasonlóan magukhoz a függvényekhez egy absztrakt osztály leszármazása utáni specializációk. Minden függvényhez tartozó ablak egy *PopupWindow* osztály, ami a *JDialog*-ból származik le. A konstruktor beállít mindent, hogy azt már ne kelljen a leszármazottaknak megcsinálniuk. Van még egy *saveParameters* nevű metódus, ami absztrakt mert azt minden függvény esetében máshogy kell, így azt majd minden függvény esetén a függvény paramétereinek megfelelően implementáltam az absztrakt metódus felüldefiniálásakor.

Működés

Egy példán bemutatva a program könnyen bemutatható.

Amikor elindítjuk a programot a belépési ponton, példányosulni fog egy *Frame* az előre meghatározott paraméterekkel amik nem mások, mint az ablak magassága, szélessége, a koordináta rendszeren található egy egység pixelben mért távolsága az ***x*** és ***y*** tengelyen, és az egész számot jelző vonal hossza, a két tengelyen külön-külön. Ezek a paraméterek a *Frame* konstruktor meghívása előtt szerepelnek, így ezeket átírva tudjuk megmondani ezek értékeit.

Az ablak konstruktora beállítja a kívánt méretet, az X gombbal való bezárás lehetőségét, háttérszínt, az ablak középre való elhelyezését, letiltja az átméretezhetőség. Létrehozza az oldalsó menüt, a fenti menüsávot, a koordinátarendszert, amit egy *LayeredPane*-be tesz, ahova később a függvények is kerülni fognak. Ezek létrejötte után minden az ablak része lesz és megjelenik minden a képernyőn.

Kezdetben egy üres koordinátarendszer fogadja a felhasználót. Függvényt hozzáadni a már említett gombokkal lehet vagy korábbi állapotot tartalmazó txt fájllal. A betöltést a menüből érjük el a File menüpont alatt a Load opcióval, ami egy könyvtérböngészőt hoz elő és az egész számítógépen kereshetünk ilyen szöveges fájlokat. Találat esetén a mentés gombbal, rögtön be is töltődik a fájlban mentett állapot.

Gombbal való hozzáadás esetén a gombok által tartalmazott *FunctionListener* kezeli le az eseményt hasonló logikát követve, mint a *ButtonListener* esetében. A Listener azonosítja a gomb szövege alapján, hogy milyen függvényt szeretnénk hozzáadni és a megfelelő ablakot előhozva és a paramétereket kitöltve hozzáadja az ablak referenciáján keresztül a kívánt függvényt, ami meghívja az oldalsó menü *updateMenu* metódusát, hogy az aktuálisak legyenek kilistázva. Természetesen tetszőleges mennyiséget hozzáadhatunk.

Abban az esetben ha az adott állapotot el akarjuk menteni, a File menüpont alatt található Save opcióval elmenthetjük, ami feldob egy könyvtárböngészőt és tetszőleges mappába tetszőleges névvel elmenthetjük azt, amit később visszatölthetünk.

Függvény törlés és hozzáadás esetén törölni kell a függvényt, viszont a függvény önmagában nem ábrázolható ezért kell egy objektum, amire lehet rajzolni. Ezt úgy oldottam meg, hogy csináltam egy *DrawablePanel* osztályt ami kap egy függvényt és a függvény *Polygon* objektumát kirajzolja. A kettőt pedig párba rendezve tárolom egy *HashMap*-ben, hogy törlés esetén tudjam, hogy melyik függvényhez melyik panel tartozik. Viszont itt figyelni kell arra, hogy ne csak magát a függvényt töröljem, hanem a panelt is, ami viszont egy *LayeredPane*-ben van.

A menü utolsó része a Help, ezen belül található a Documentation opció, amivel ezt a dokumentációt lehet megnyitni az adott számítógép alapértelemzett pdf nezegetőjével.

GitHub link: https://github.com/Leveskockaaa/Math-Function-Visualizer