1. A testekkel kapcsolatos feladat megoldásának módosítása

1. Hasáb jellegű testek esetén az alapterület kiszámítását egy beágyazott objektumra delegáljuk.

   Így elérhető, hogy négyzet alapú hasáb, illetve gúla esetén ugyanazt a műveletet használjuk fel, nem kell ugyanazt a megvalósítást elkészítenünk két különböző osztályban.

   Ugyanez igaz háromszög, illetve kör alap esetén is.

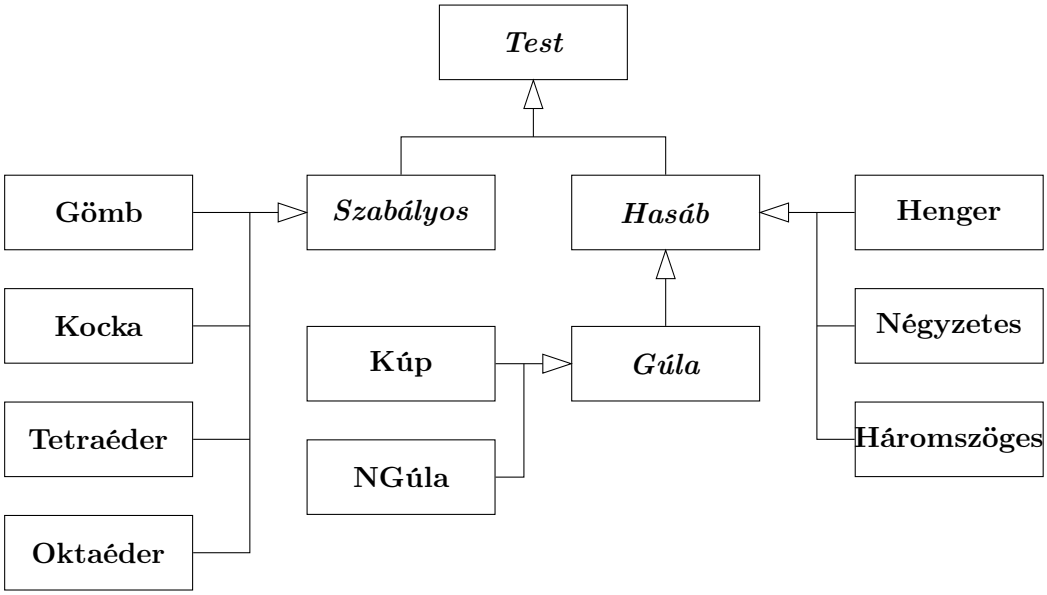
2. A testeket interaktív módon, egy párbeszédablak segítségével adhatjuk meg.

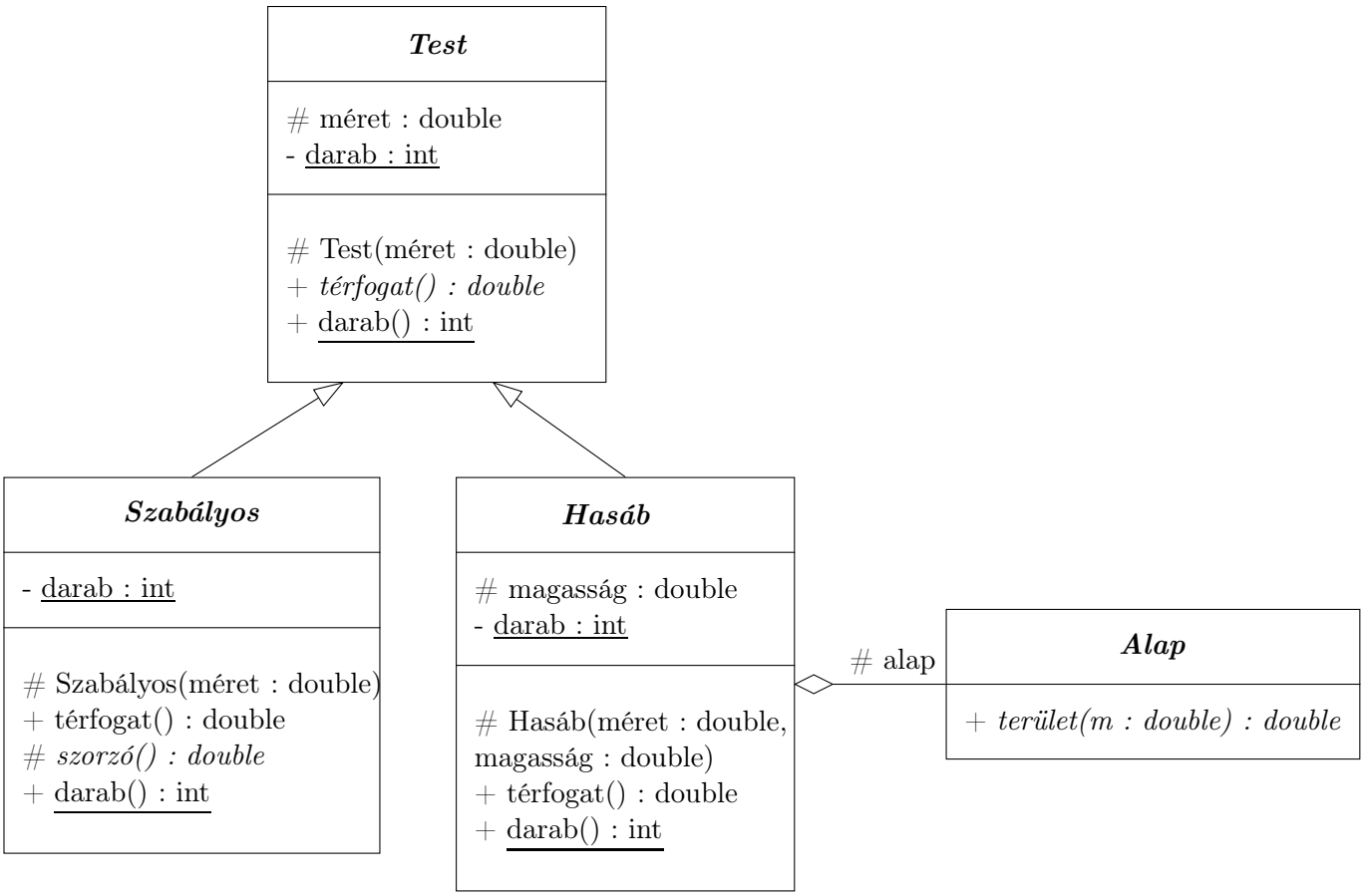
1.1. Alapok használata

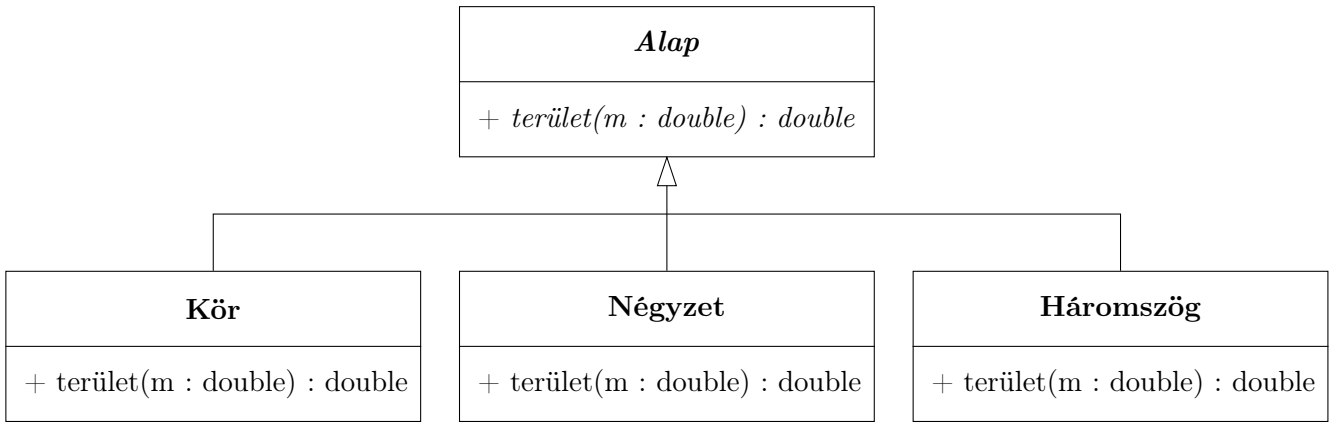
Az alapterület kiszámítását az ***Alap*** absztrakt osztály deklarálja, amit a konkrét **Kör**, **Négyzet**, **Háromszog** osztályokban valósítunk meg.

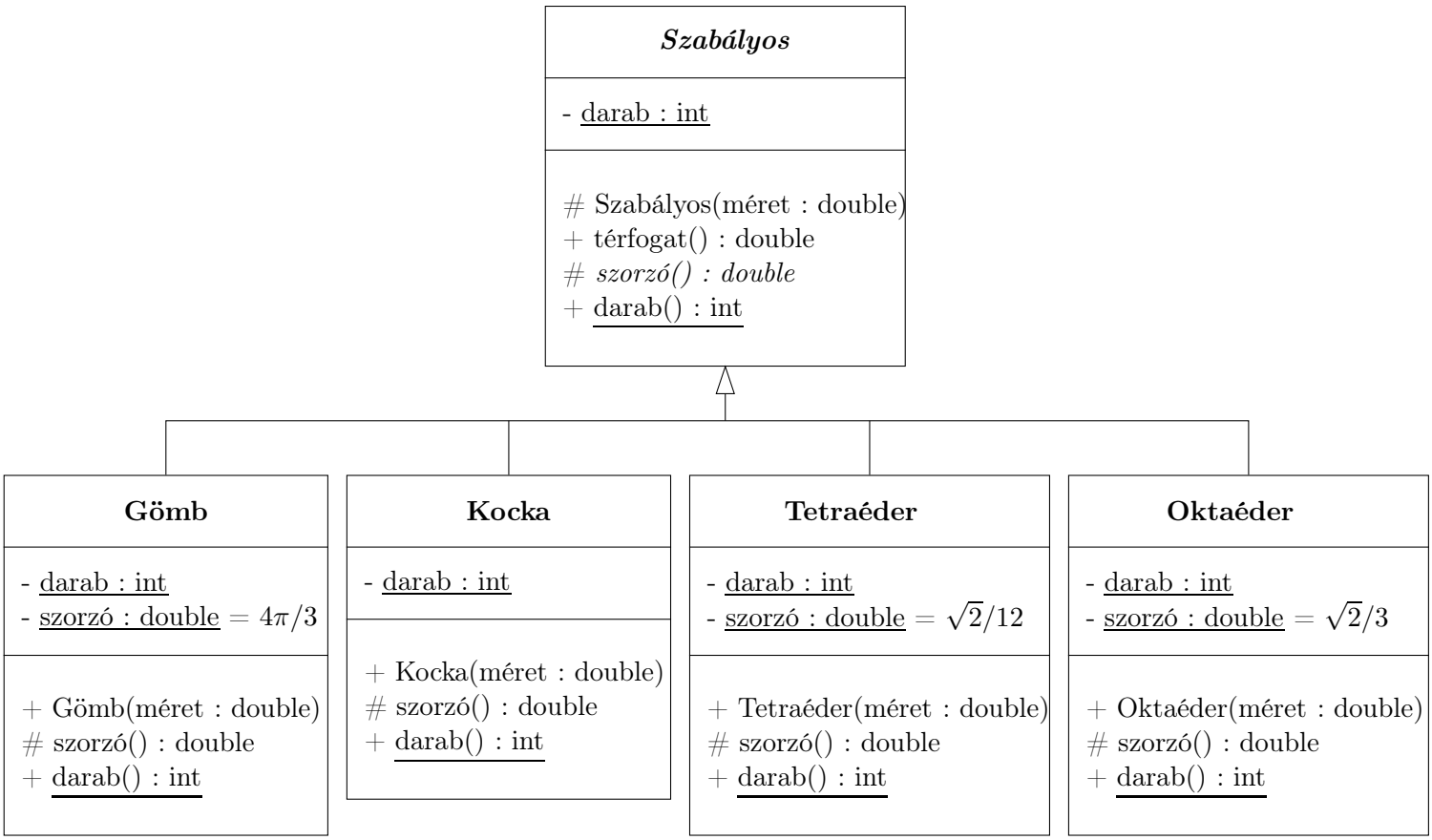
A beágyazott objektum ennek megfelelően az ***Alap*** osztályból származtatott konkrét osztály példánya lesz.

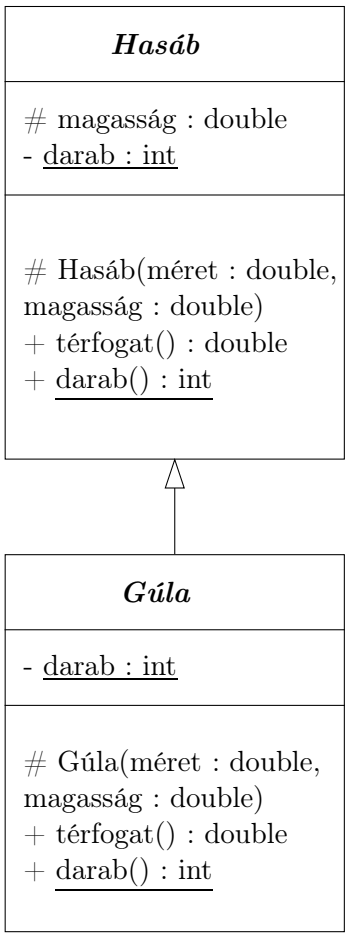
Az új osztálydiagram

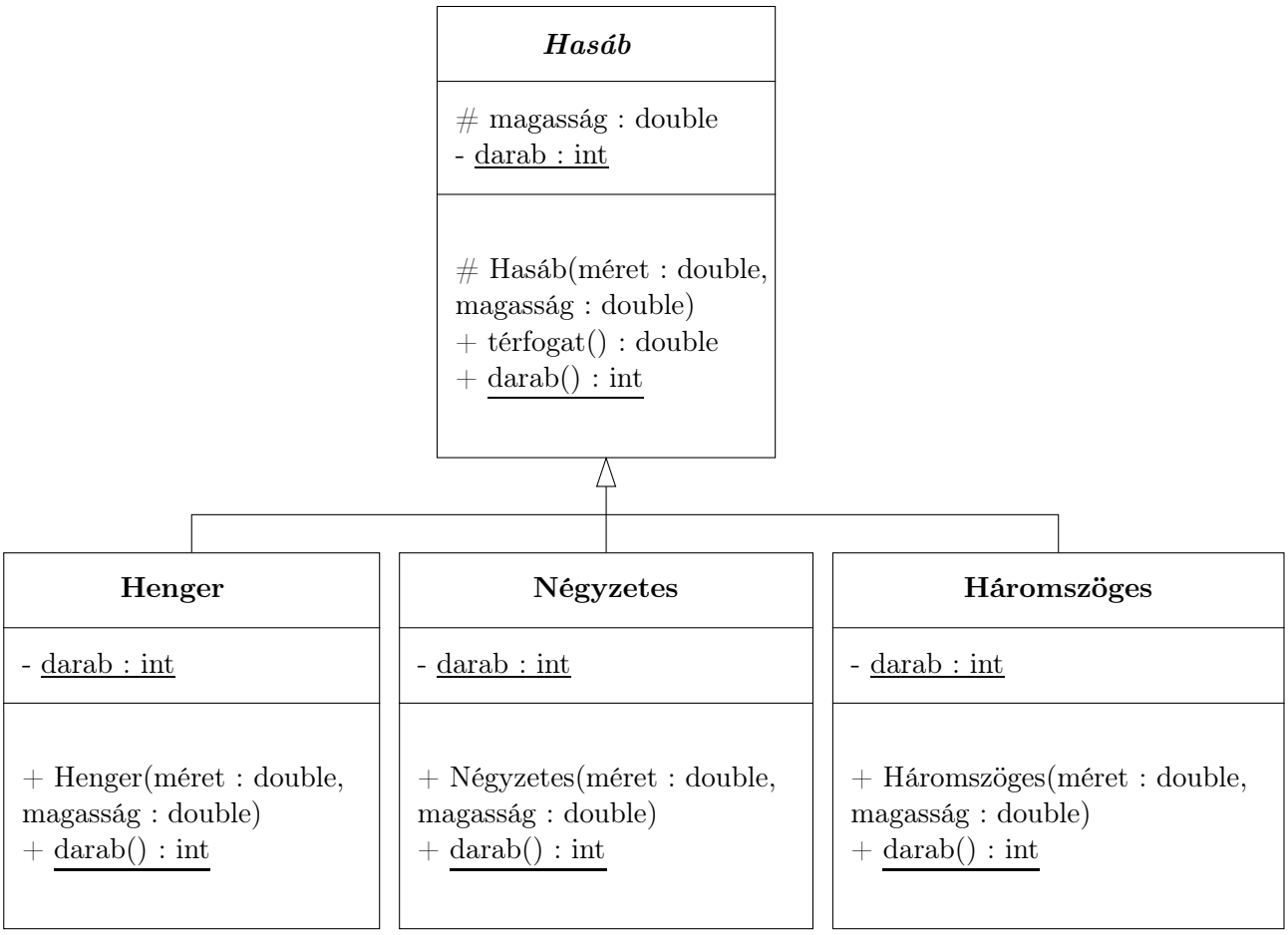


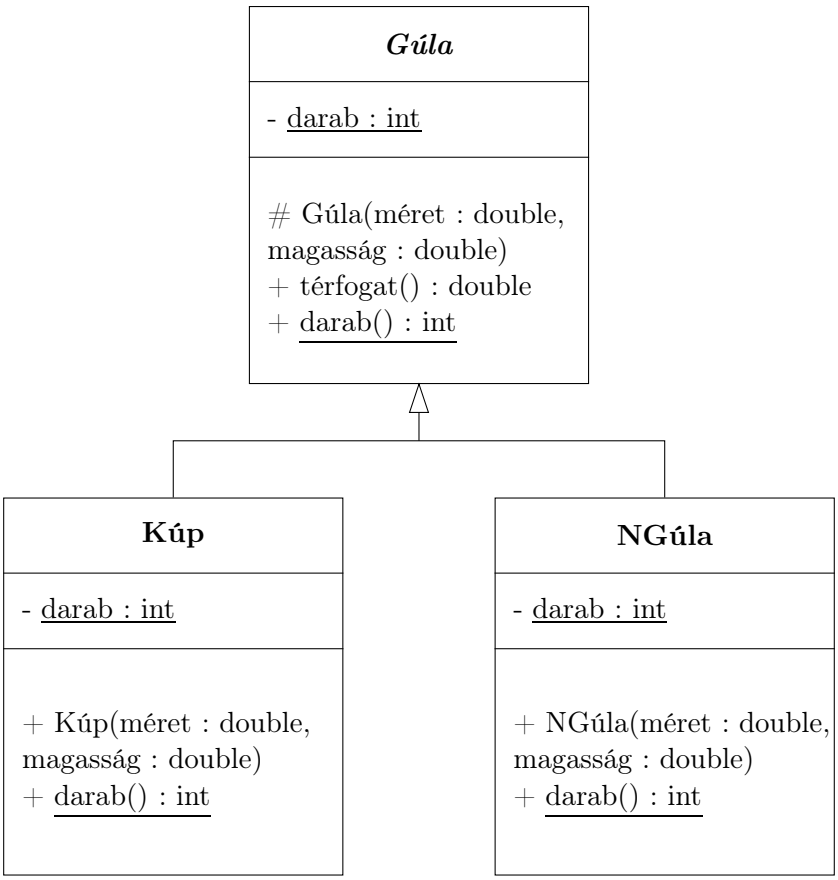

Test
méret : double
- darab : int
Test(méret : double)
+ térfogat() : double
+ darab() : int
Szabályos
- darab : int
Szabályos(méret : double)
+ térfogat() : double
# szorzó() : double
+ darab() : int
Hasáb
magasság : double
- darab : int
Hasáb(méret : double, magasság : double)
+ térfogat() : double
+ darab() : int
# alap
Alap
+ terület(m : double) : double


Alap
+ terület(m : double) : double
Kör
+ terület(m : double) : double
Négyzet
+ terület(m : double) : double
Háromszög
+ terület(m : double) : double


Szabályos
- darab : int
Szabályos(méret : double)
+ térfogat() : double
# szorzó() : double
+ darab() : int
Gömb
- darab : int
- szorzó : double = 4π/3
+ Gömb(méret : double)
# szorzó() : double
+ darab() : int
Kocka
- darab : int
+ Kocka(méret : double)
# szorzó() : double
+ darab() : int
Tetraéder
- darab : int
- szorzó : double = √2/12
+ Tetraéder(méret : double)
# szorzó() : double
+ darab() : int
Oktaéder
- darab : int
- szorzó : double = √2/3
+ Oktaéder(méret : double)
# szorzó() : double
+ darab() : int


Hasáb
magasság : double
- darab : int
Hasáb(méret : double,
magasság : double)
+ térfogat() : double
+ darab() : int
Gúla
- darab : int
Gúla(méret : double,
magasság : double)
+ térfogat() : double
+ darab() : int


Hasáb
magasság : double
- darab : int
Hasáb(méret : double, magasság : double)
+ térfogat() : double
+ darab() : int
Henger
- darab : int
+ Henger(méret : double, magasság : double)
+ darab() : int
Négyzetes
- darab : int
+ Négyzetes(méret : double, magasság : double)
+ darab() : int
Háromszöges
- darab : int
+ Háromszöges(méret : double, magasság : double)
+ darab() : int


Gúla
- darab : int
Gúla(méret : double, magasság : double)
+ térfogat() : double
+ darab() : int
Kúp
- darab : int
+ Kúp(méret : double, magasság : double)
+ darab() : int
NGúla
- darab : int
+ NGúla(méret : double, magasság : double)
+ darab() : int

1.2. `TestDialog` párbeszédablak

A felületet `swing` elemek segítségével készítjük el.

A testeket egy párbeszédablakban (**JDialog**) adjuk meg.

1.2.1. Elemek

- A test típusát egy legördülő listából választhatjuk (**JComboBox**). Ehhez tartozik egy címke is (**JLabel**). Változók nevei: `típus`, `típuscímke`.
- A test méretét egy szövegszerkesztő mezőben adhatjuk meg (**JTextField**), hozzá címke tartozik (**JLabel**). Változók nevei: `méret`, `méretcímke`.
- A test magasságát egy szövegszerkesztő mezőben adhatjuk meg (**JTextField**), hozzá címke tartozik (**JLabel**). Változók nevei: `magasság`, `magasságcímke`.
- Kell két gomb (**JButton**) az aktuális adatoknak megfelelő test felvételére, illetve a felviteli folyamat befejezésére. Változók nevei: `felveszgomb`, `végegomb`.

1.2.2. Funciók

- A konstruktorban létre kell hozni a párbeszédablakot az elemeivel együtt. Az elemeket el kell helyezni, erre használhatók a `Layout`-ok. A legördülő lista létrehozásakor meg lehet adni az elemeket, amelyeket a a konstans `típusok` tömb tartalmaz.
- A `mutat` művelettel jeleníthetjük meg a párbeszédablakot.
- A `gombKód` függvénnyel kérdezhető le, hogy melyik gombot használtuk. A használathoz bevezetünk két konstanst: `FELVESZ` és `VÉGE`, illetve egy attribútumot a tárolásra: `gombkód`.
- A test lekérdezésére szolgál az `adat` függvény. A testet a `test` attribútum tárolja.
- A `felveszgomb` megnyomásakor végrehajtandó esemény a `felveszakció`. Ekkor meg kell próbálni létrehozni egy testet az adatokból (`létrehoz`), és ha ez sikerül, akkor a gomb kódot kell állítani, és az ablakot kell elrejteni.
- A `végegomb` lenyomásakor végrehajtandó esemény a `végeakció`. Ekkor a gomb kódot kell állítani, és az ablakot megszüntetni.

- Egy test létrehozásakor meg kell határozni a méreteket, a típust, és megfelelő adatok esetén létrehozni a megfelelő objektumot. Numerikus értékeknél ügyelni kell arra, hogy a szövegdoboz tartalma konvertálható legyen. Ezt kivételkezeléssel tudjuk ellenőrizni.

Vegyük észre, hogy minden egyes alap típusból (Kör, Négyzet, Háromszög) egyetlen példány elegendő, hiszen adatot nem tárolunk, csak a terület kiszámításának módját adjuk meg. Éppen ezért felesleges minden hasáb jellegű testben egy külön objektumot létrehoznunk, elegendő a megfelelő példányra hivatkoznunk.

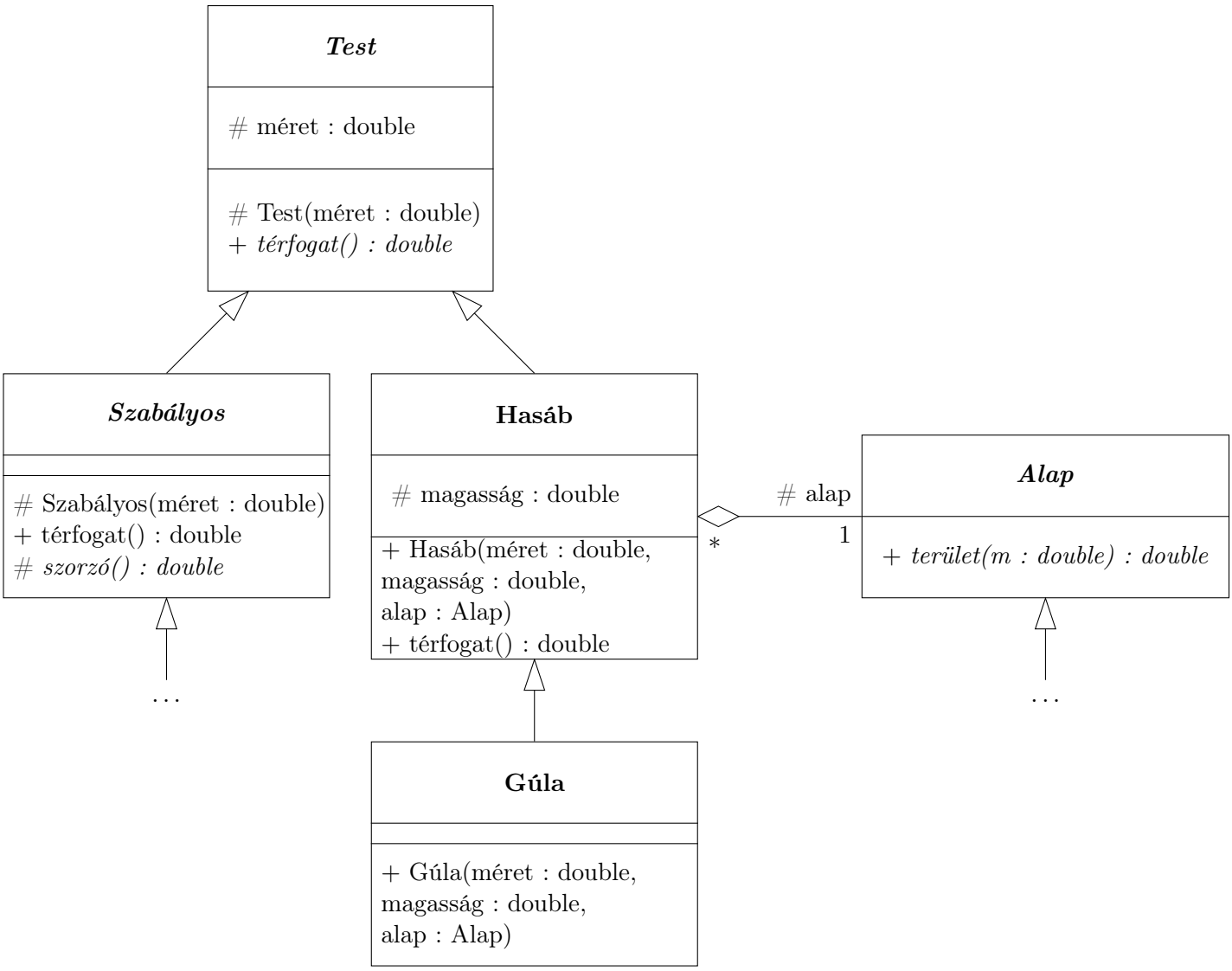
Ennek megfelelően a konkrét alap osztályokban vezessünk be egy rejtett attribútumot az egyetlen példány tárolására, egy műveletet annak elérésére, és a konstruktort rejtsük el, hogy elkerüljük a felesleges példányok létrehozását.

Java program: `Testek2` projekt

2. A testekre vonatkozó feladat módosítása

Az előző megoldásban a beágyazott alap objektum tulajdonképpen feleslegessé teszi a speciális hasáb és gúla (Henger, Négyzetes, Háromszöges, Kúp, NGúla) osztályokat. Ezekre egyedül a darabszám miatt van szükség. Ha nem követeljük meg az egyes fajtákból létrehozott objektumok tárolását, akkor ezek az osztályok elhagyhatóak.

A szabályos testeket ez a változás nem érinti, így azokkal most nem kell külön foglalkoznunk.


Test
méret : double
Test(méret : double)
+ térfogat() : double
Szabályos
Szabályos(méret : double)
+ térfogat() : double
# szorzó() : double
...
Hasáb
magasság : double
+ Hasáb(méret : double,
magasság : double,
alap : Alap)
+ térfogat() : double
# alap
*
1
Alap
+ terület(m : double) : double
...
Gúla
+ Gúla(méret : double,
magasság : double,
alap : Alap)

Java program: `Testek3` projekt

(A programban szerepel a felszín kiszámítása is, és az ehhez szükséges műveletek. Szabályos testek esetén két szorzó tényező kell, ezért az eddig a térfogathoz használt `szorzó` `tszorzó` lett, a felszínre vonatkozó tényező pedig `fszorzó`.)

3. Párbeszédablak elkészítése interaktív tervezővel

Java program: `Testek4` projekt

Grafikus felhasználói felület

Két csomag elemeiből lehet a felületet elkészíteni: `awt`, `swing`.

Mi a `swing` csomag elemeiből alakítjuk ki a felület elemeit:

- `JFrame` keret,
- `JLabel` címke,
- `JButton` gomb,
- `JPanel` panel,
- `JComboBox` kombobox,
- `JList` lista,
- `JSlider` csúszka,
- `JTable` táblázat.

Eseménykezeléshez az `awt` csomag is szükséges.

1. Üres keret létrehozása

A főprogramot külön osztályba helyezzük el: `Keret`. Itt csak egy keret objektumot hozunk létre.

```
public class Keret
{
    public static void main(String[] args)
    {
        new ÜresKeret();
    }
}
```

A keretet megadó osztály: `ÜresKeret`.

```
import javax.swing.JFrame;

public class ÜresKeret extends JFrame
{
    public ÜresKeret()
    {
        setTitle("Üres keret");
        setSize(200, 60);
        setVisible(true);
    }
}
```

A konstruktorban megadjuk az ablak címét, méretét, végül láthatóvá tesszük (megjelenítjük).

A program futtatása során a következő ablak jelenik meg. (Parancssor ablakban a megfelelő könyvtárból a `java ÜresMain` parancs megadása után.)

A lezáró gombra kattintva a program ablaka bezárul, de a program futása nem fejeződik be. (Parancssorban nem kapjuk vissza a promptot, fejlesztőkörnyezetben a futtatási konzol ablak mutatja, hogy a program fut.)

Ennek oka, hogy a program nem fejeződött be, csak a keret tűnt el a képernyőről.

Program leállítása:

- parancssorból a `Ctrl+C` lenyomásával
- NetBeans környezetben a `Run` menü `Stop Build/Run` pontjával

1.1. Program leállítása kilépéskor

Az alapértelmezett megvalósítás csak eltűnteti a keretet.

1.1.1. A kerethez rendelet automatikus bezáró művelet előírása

```
import javax.swing.*;

public class ÜresKeret extends JFrame
{
    public ÜresKeret()
    {
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        setTitle("Üres keret");
        setSize(200, 60);
        setVisible(true);
    }
}
```

1.1.2. A keret kiegészítése a kilépés gombot kezelő eseménykezelővel

A kezelőt a `WindowAdapter` osztályból kell származtatni, és az `addWindowListener` művelettel a kerethez rendelni. Egyetlen feladata az ablak bezárásának kezelése, amit a `windowClosing` eljárás átdefiniálásával tehetünk meg. Az eljárás paramétere a bezárást okozó `WindowEvent` típusú esemény, amit nem használunk. Az eljárásban ki kell lépni a programból.

(Ezt az utat kell követni, ha a kilépéskor egyéb tevékenysége(ke)t is el kell végeznünk.)

```
import javax.swing.*;
import java.awt.event.*;

public class ÜresKeret extends JFrame
{
    public ÜresKeret()
    {
        addWindowListener(new WindowAdapter()
        {
            @Override
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });
        setTitle("Üres keret");
        setSize(200, 60);
        setVisible(true);
    }
}
```

Az előző megoldásban szereplő „implicit” objektum helyett az objektumot felvehetjük az osztályon belül. (`Keret1` projekt)

```
import javax.swing.*;
import java.awt.event.*;

public class ÜresKeret extends JFrame
{
    public ÜresKeret()
    {
        addWindowListener(kilépéskezelő);
        setTitle("Üres keret");
        setSize(200, 60);  setVisible(true);
    }

    private WindowAdapter   kilépéskezelő = new WindowAdapter()
    {
        @Override
        public void windowClosing(WindowEvent e)
        {
            System.exit(0);
        }
    };
}
```

Végül ugyanez elérhető egy külön osztály létrehozásával is, amelynek objektumát adjuk meg paraméterként.

```
import java.awt.event.*;
public class KilépésKezelő extends WindowAdapter
{
    @Override
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
}

import javax.swing.*;
public class ÜresKeret extends JFrame
{
    public ÜresKeret()
    {
        addWindowListener(new KilépésKezelő());
        setTitle("Üres keret");
        setSize(200, 60);
        setVisible(true);
    }
}
```

2. Kattintásokat számláló gomb

A következő programban elhelyezünk egy gombot a keretben, amely számlálja és megjeleníti a kattintásokat. Ezen kívül a keret ikonját is megváltoztatjuk.

A keret ikonját a `setIconImage` művelettel adhatjuk meg. Ennek paramétere az ikon, de ezt csak a megfelelő eszközön (`Toolkit`) keresztül rendelhetjük a kerethez.

```
setIconImage(Toolkit.getDefaultToolkit().getImage("gomb.png"));
```

A `Toolkit` az `awt` csomag része.

Így azonban ez nem töltődik be a `jar` állományból. Ehhez URL-ként kell megadni a fájl nevét:

```
java.net.URL    url = Gomb.class.getResource("gomb.png");
setIconImage(Toolkit.getDefaultToolkit().getImage(url));
```

A számláló gombot a keret „adatterületére” helyezzük, amit a `getContentPane()` művelet ad meg. Ehhez az `add` művelettel vehetünk hozzá egy elemet.

A gombot egy osztály segítségével valósítjuk meg: `SzámlálóGomb`.

Ezt a `JButton` osztályból kell származtatni, és ki kell egészíteni eseménykezeléssel is, ezért az osztály az `ActionListener` interfészt is megvalósítja.

Java program: `Gomb1` projekt

Készítsük el ugyanezt az alkalmazást a NetBeans tervezőjével!

3. Kilépés gomb

Egészítsük ki az előző programot úgy, hogy a keretben helyezzünk el egy kilépésre lehetőséget adó gombot!

El kell készítenünk az előzőekhez hasonlóan a kilépésre szolgáló gombot, ahol az esemény kezelése a program befejezését jelenti.

A keret területére nem egy hanem két gombot kell elhelyeznünk. Ehhez meg kell adnunk a keret területén az elhelyezési módot (`setLayout`). A legegyszerűbb lehetőség a `FlowLayout`, amelybe folytonosan helyezhetünk el elemeket.

Ez a feladat ugyan egyszerű, de általában fontos lehet, hogy ugyanaz következzen be, amikor kilépünk a programból. Ezért célszerű az ablak lezárásához és a kilépő gomb lenyomásához ugyanazt az eljárást rendelnünk (`kilép`). Ezért az ablak bezárásához eseménykezelőt rendelünk, amelyben ezt hívjuk meg, és a gomb megnyomásakor is ezt hívjuk meg.

A kilépő gombot most másként hozzuk létre. Egy gomb konstruktorának paramétere lehet egy `Action` típusú objektum. Ebben meg lehet adni az esemény kezelést, illetve a gomb további tulajdonságait. Az `Action` egy interfész, amelyben szereplő műveletekre (az esemény kezelésének kivételével) alapértelmezett megvalósítást ad az `AbstractAction` osztály. Ennek egy objektumával paraméterezzük a gombot.

Java program: `Gomb2` projekt

4. Elhelyezési módok

`FlowLayout` az elemek folyamatos elhelyezése sorban.

`BorderLayout` az elemek elhelyezése 5 helyre: középre (`CENTER`), felülre (`NORTH`), alulra (`SOUTH`), balra (`WEST`), jobbra (`EAST`).

`GridLayout` az elemek folyamatos elhelyezése megadott méretű rácsban, ugyanolyan méretben.

`GridBagLayout` az elemek elhelyezése rácsban, az elemek kiterjedése több egység is lehet.

`BoxLayout` elemek elhelyezése adott méretben, adott távolságra vízszintes vagy függőleges irányban.

`CardLayout` kártyapakli elhelyezés (csak egy elem látszik, ami változtatható).

`OverlayLayout` elemek egymásra helyezése.

5. Kattintások számának megjelenítése külön

Ne a gomb jelenítse meg a kattintások számát, hanem az egy külön mezőben legyen látható!

Kétféleképpen járhatunk el:

1. A `SzámlálóGomb` osztályt megváltoztatjuk: ismernünk kell a kiírás helyét, ami egy címke. (Vagy a keretet, aminek szólni kell, és az kérdezi le az értéket, és állítja a címke feliratát.)

   Java program: `Gomb3` projekt

2. A keretben tartjuk nyilván az értéket, és a számlálásért felelős gomb eseménykezelőjét is itt helyezzük el.

   Java program: `Gomb4` projekt