

*Programozás alapjai II.*  
*Grafikus felületek és a C++*  
*(Ismeretbővítő, fakultatív előadás)*

---

Szeberényi Imre, Somogyi Péter

BME IIT

<szebi@iit.bme.hu>



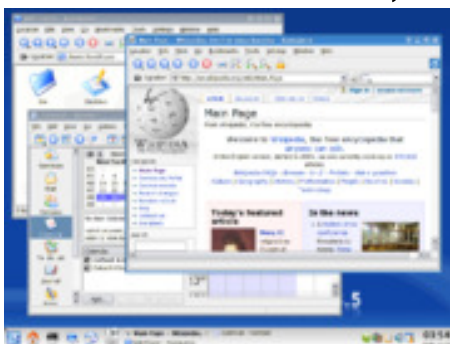
MŰEGYETEM 1782

# *Minden sokat változott...*

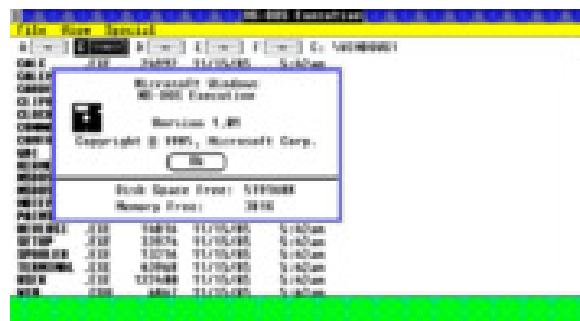


# Grafikus megjelenítő se volt mindig

- Annak ellenére, hogy a 60-as években már volt grafikus megjelenítő elterjedésükre még várni kellett. (egér: 1963)
- A mai értelemben vett grafikus felhasználói felületek (GUI) a 80-as években alakultak ki.
- X Window, MS Windows, Mac OS, ...



C++ programozási nyelv



© BME-IIT Sz.I.



2020.04.13.

# Hogyan működnek ?

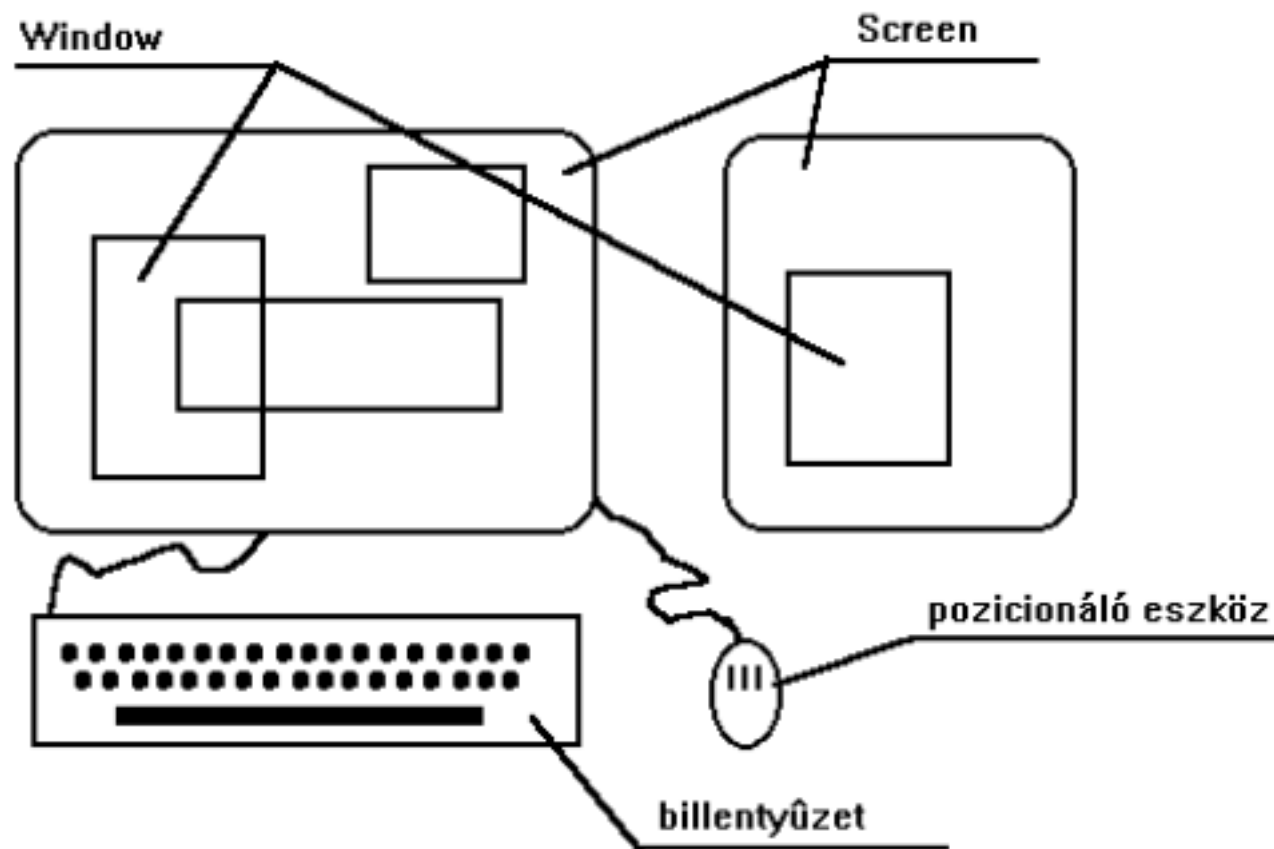
- Objektum szemlélet (widget, gadget)
- **Eseményvezérelt** (objektumok eseményekkel kommunikálnak)
- A felhasználói felület tervezése és a program logikája gyakran elválik (külön módosítható)
- Első elterjedt grafikus rendszer a UNIX szabványos grafikus felülete az X Window rendszer, amit objektum szemléletű, de nem OO nyelven írták (C-ben)

# *X window rendszer fogalmai*

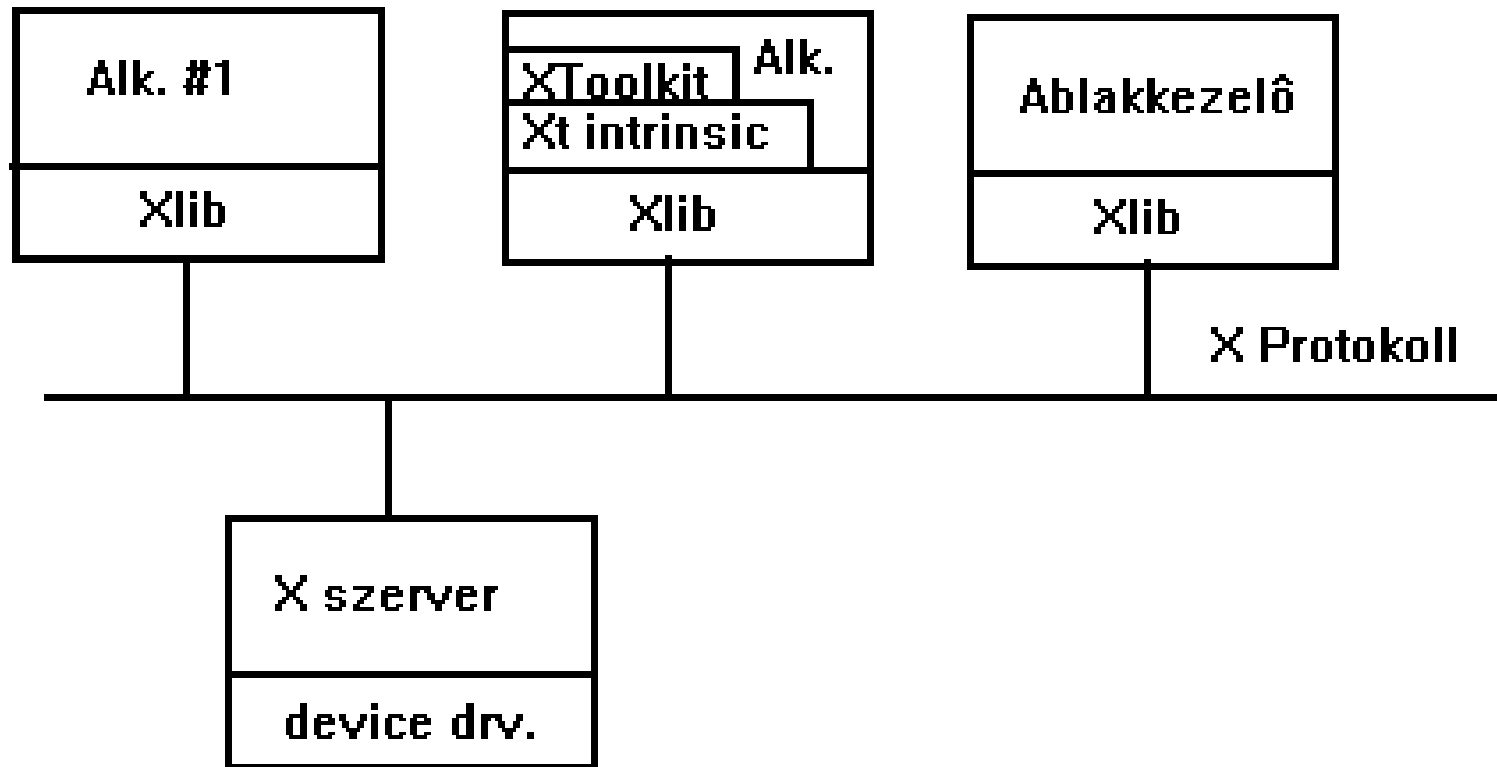
---

- Kliens = a szolgáltatást igénybe vevő.
- Egy adott berendezés (gép) egyszerre elláthat szerver funkciókat is, és kliens programokat is futtathat.
- X szerver = az X display-t működtető, a kliensek számára grafikus szolgáltatást nyújtó program.

# *X display = munkahely felépítése*



# *X rendszer szoftver architektúrája*



# *Eseményvezérelt alkalmazás*

---

- **Inicializálás:**  
Kapcsolódás a szerverhez, window-k létrehozása, szerver erőforrások lefoglalása és attribútumaik beállítása.
- **Eseményhurok:**  
a programhoz érkező események feldolgozása.
  - Nem a program vezérli a felhasználót, hanem a felhasználó a programot



# *Kapcsolódás a szerverhez*

## **HOST:SERVER.SCR**

alakú azonosító stringgel történik, amit vagy explicit kap a megfelelő Xlib rutin, vagy az explicit megadás hiányában a **DISPLAY** környezeti változóból veszi.

- **HOST:** A szerveret futtató számítógép hálózati azonosítója (név vagy cím).
- **SERVER:** Az adott hoston futó szerver azonosító száma (0. az első szerver).
- **SCR:** A kívánt screen sorszáma (0. az első).
  - Például: bubuka.iit.bme.hu:0.0

# „Egyszerű” X program

```
int main() {
    Display *display;
    Window wMain;
    XEvent event;
    if ((display = XOpenDisplay(NULL)) == NULL) {
        fprintf(stderr, "Can't connect\n"); exit(1);
    }
    wMain = XCreateSimpleWindow(display,
        DefaultRootWindow(display), 0, 0, width,
        height, borderWidth, border, bacgkground);
```

további inicializálások, gc, eseménymaszk, ablakok ...

```
XMapWindow(display, wMain);
while(1) {
    XNextEvent(display, &event); .....
}
XCloseDisplay(display);
}
```

események kezelése

# Események kezelése

```
XSelectInput (display, wMain, ExposureMask |  
             KeyPressMask | ButtonPressMask |  
             StructureNotifyMask);
```

események  
kiválasztása

```
XMapWindow (display, wMain);
```

```
while (1) {  
    XEvent          event;  
  
    XNextEvent (display, &event);  
  
    switch (event.type) {  
        case ConfigureNotify:  
            ...  
            break;
```

eseményhurok

események  
felismerése

# Események kezelése /2

```
case Expose:
    ...
    XDrawString(display, wMain, gc, .....
    ...
break;
case KeyPress:
case ButtonPress:
    ...
    if (...) {
        XUnloadFont(display, font_info->fid);
        XFreeGC(display, gc); XCloseDisplay(display);
        exit(1);
    }
}
```

rajzolás

erőforrások felszabadítása

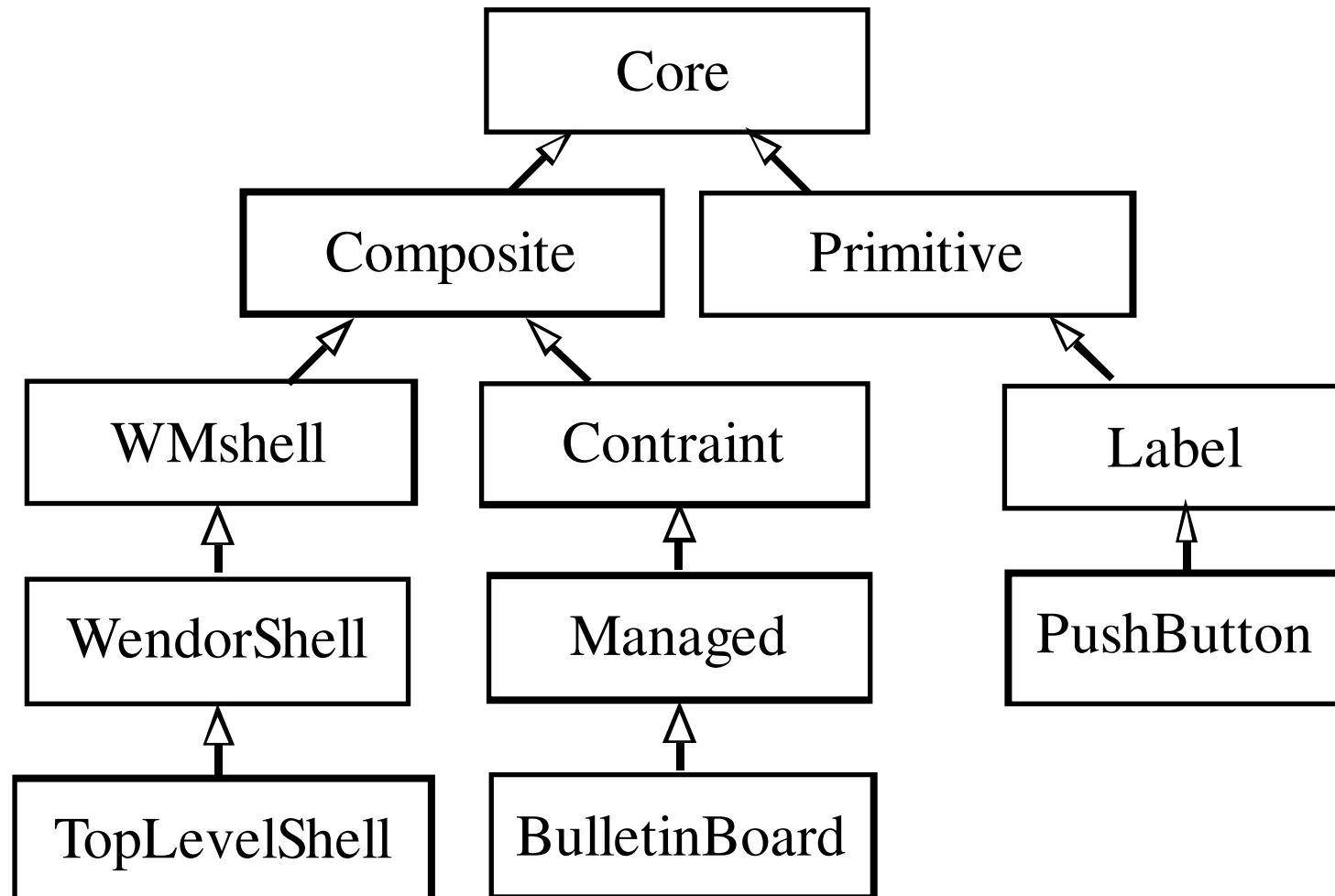
# *Lehet egyszerűbben ?*

- Az X toolkit intrinsic, mint "OO motor" szolgáltatásaira épülő toolkit segítségével.
  - Athena, OpenLook, Motif, CDE, KDE ...
- Objektum orientált (pl. C++) nyelvhez kapcsolódó könyvtárak / toolkitek alkalmazásával
  - Agar, CEGUI, CLX, dlib C++, FLTK, FOX, GLUI, GTK+, IUP, Juce Lgi, Qt, Quinta, Tk, TnFOX, Ultimate++, VCF, wxWidgets, YAAF, XForms, XVT, ...

# Toolkit

- Objektum típusokat definiál, melyekkel megvalósíthatók a szokásos GUI elemek
  - label, button, radiobutton, checkbox, editbox, bulletinboard, scrollbar, stb.
- Az objektumok közös őssel rendelkeznek (widget, v. gadget).
- Származtatással újabb objektumok hozhatók létre.
- Az objektumok kommunikálnak az alkalmazással és az X szerverrel.

# Motif toolkit hierarchia (részlet)



# Motif hello

```
#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Label.h>
main(int argc, char *argv[]) {
    Widget topLevel, hello;
    topLevel = XtInitialize(argv[0],
        "Motifhello", NULL, 0, &argc, argv);
    hello = XtCreateManagedWidget("hello",
        xmLabelWidgetClass, topLevel, NULL, 0);
    XtRealizeWidget(topLevel);
    XtMainLoop();
}
```

publikus header

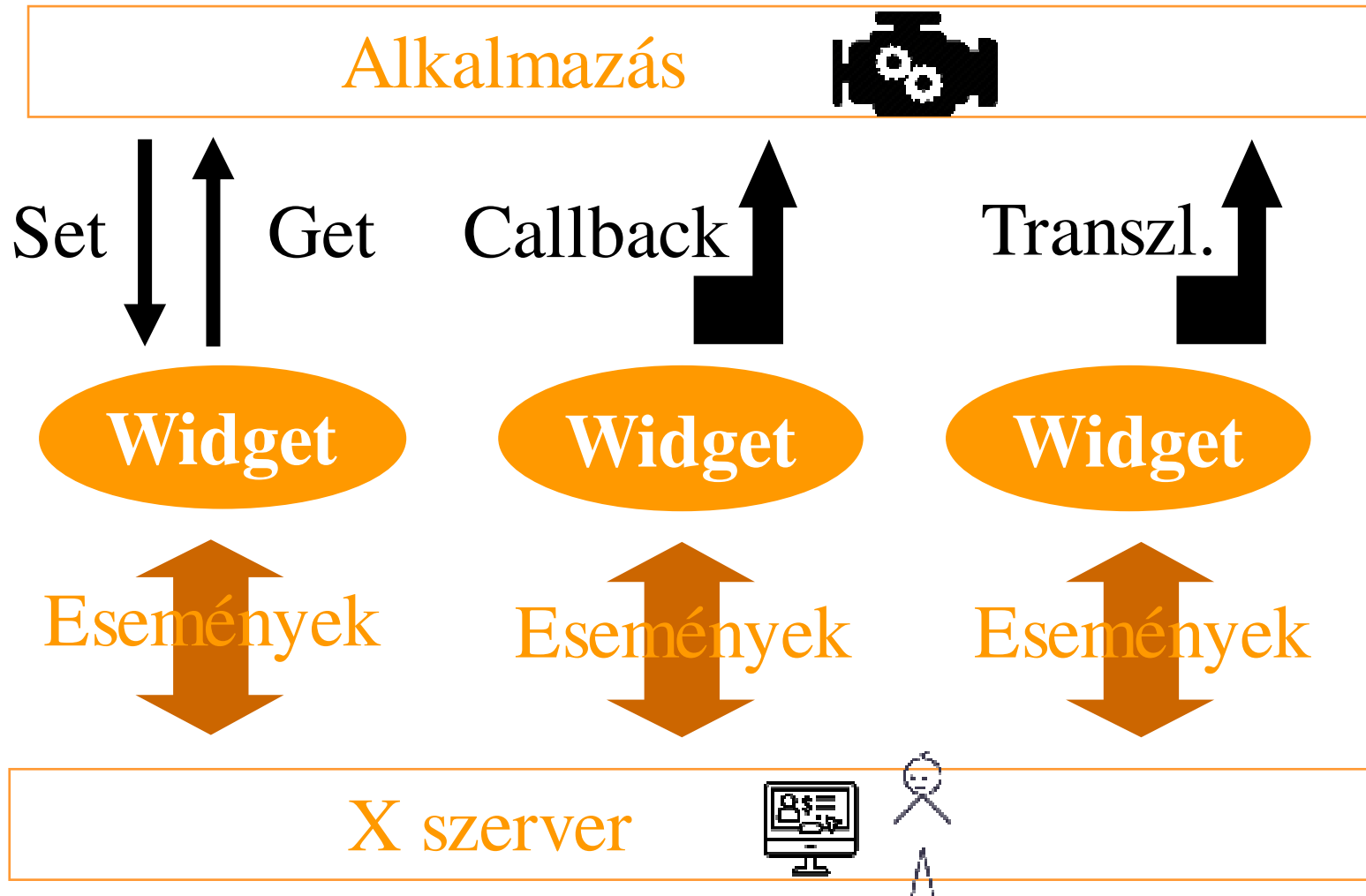
new

osztály

eseményhurok



# Komunikációs sémák



# *MS Windows*

---

- Szintén a 80-as évek elején indult
- Hasonló alapelvek:
  - eseményvezérlés
  - raszter orientált grafika
  - objektum orientált szemlélet
- Fő különbségek:
  - az X nem része az OS-nek
  - az X hálózatorientált
  - az X szerver/kliens megközelítésű

# *MS windows program szerkezete*

---

```
WinMain(.....) { // inicializálások
// ablak mint "objektum" regisztrálása
// menü, kurzor, icon, méret, szín, eseménykezelő, ...
    RegisterClass(.....);
// létrehozás
    InitInstance(.....) ;
// üzenetek feldolgozása:
    while (GetMessage(&msg, NULL, 0, 0)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
}
```

# *MS windows eseménykezelés*

```
WndProc(...) {  
    ....  
    switch (message) {  
        case WM_PAINT:  
            hdc = BeginPaint(hWnd, &ps);  
  
            ....  
            EndPaint(hWnd, &ps);  
            break;  
        case WM_DESTROY:  
            PostQuitMessage(0);  
            break;  
  
        .....  
    }  
}
```

V. Studio: new project → visual c++ → win32 project

## *new project → visual c++ → win32 proj*

---

- Skeleton alkalmazás keletkezik.
- Fordítható, üres ablak, menüje is van.
- GUI elemeket külön szöveges fájlban írjuk le (resource file, \*.rc)
- Külön editorral, vagy szövegesen szerkeszthető.
- Resource compiler lefordítás után (\*.res) belegyűrja az exe-be, így együtt hurcolható.

# RC file példa

```
IDC_WIN32VS MENU
BEGIN
```

```
    POPUP "&File"
```

```
    BEGIN
```

```
        MENUITEM "E&xit",    IDM_EXIT
```

```
        MENUITEM "&Open",    IDM_OPEN
```

```
    END
```

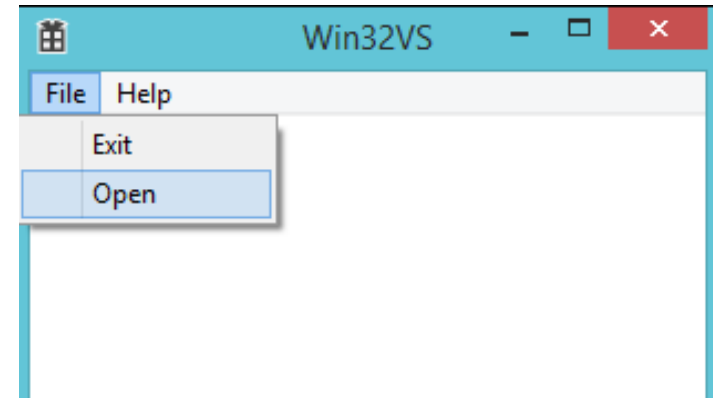
```
    POPUP "&Help"
```

```
    BEGIN
```

```
        MENUITEM "&About ...",    IDM_ABOUT
```

```
    END
```

```
END
```



# *FOX toolkit*

---

- C++ alapú
- átgondolt (...)
- kis méretű
- platform független (X, MS, Mac)
- számos ma szokásos megoldás
  - perzisztencia,
  - úszó dobozok,
  - buborék tippek

<http://www.fox-toolkit.org/>

# FOX hello

```
int main(int argc, char *argv[]){
    FXApp application("Hello", "FoxTest");
    application.init(argc, argv);
    FXMainWindow *main=new FXMainWindow(&application,
        "Hello", NULL, NULL, DECOR_ALL);
    new FXButton(main, "&Hello Fox!", NULL,
        &application, FXApp::ID_QUIT);
    application.create();
    main->show(PLACEMENT_SCREEN);
    return application.run();
}
```

üzenet

szerver oldalon

eseményhurok



# FOX eseménykezelés

- Események kezelését makrókkal felépített táblák segítik. -> callBack függvények

```
FXDEFMAP (myWindow) myWindowMap[]={  
    FXMAPFUNC (SEL_COMMAND, myWindow::ID_QUIT,  
    myWindow::cbFv),
```

esemény

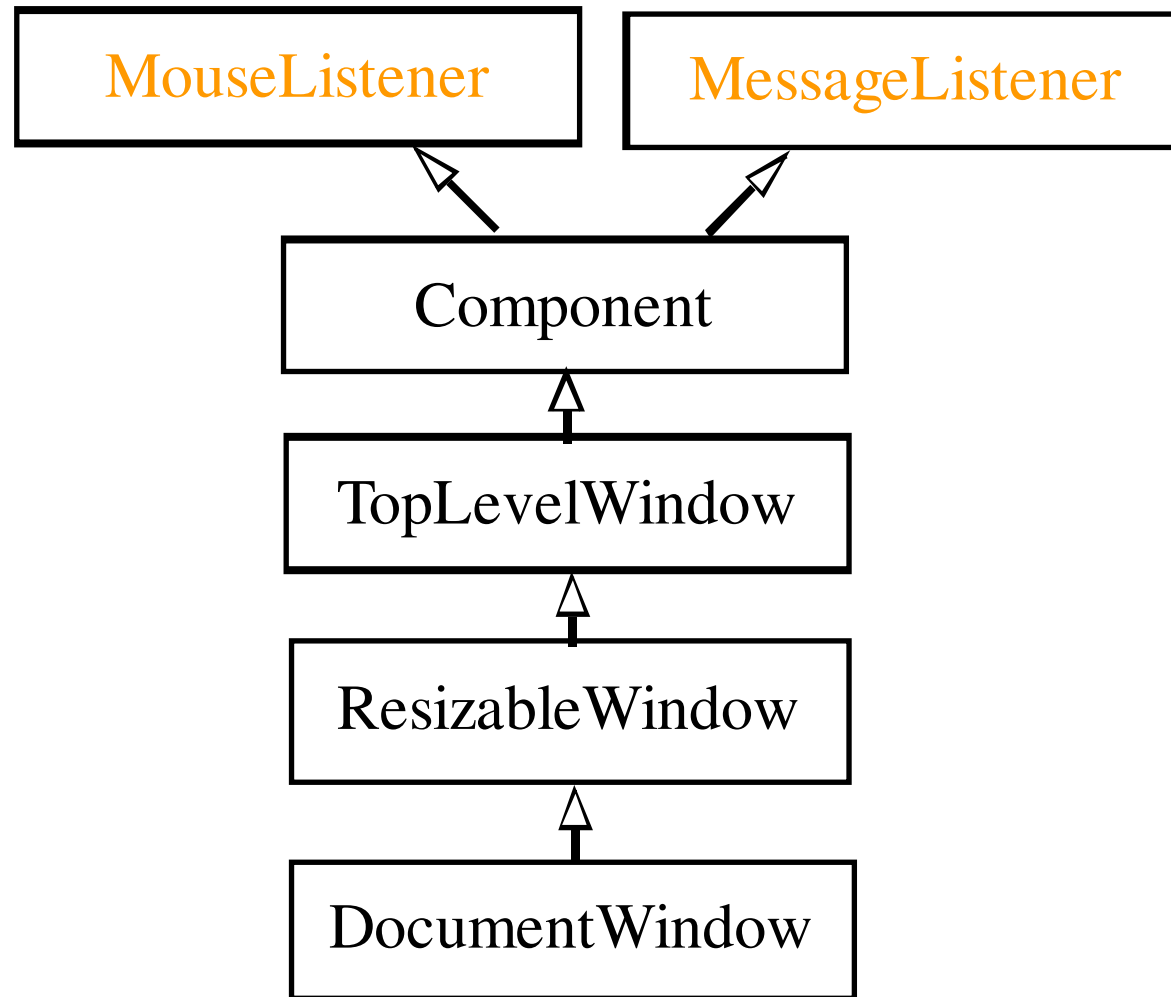
- Függvények egységes üzenetformátumot kapnak:
  - long cbFv(FXObject\* sender, FXSelector sel, void \*ptr);

# *JUCE toolkit*

- C++ alapú, jobban kihasználja a C++ lehetőségeit
- átgondolt
- kis méretű
- platform független (X, MS, MAC, Android)
- makróktól mentes
- OpenGL integráció
- GPL + Commercial
- cross platform audio

<https://www.juce.com/>

# Objektum hierarchia példa



# JUCE hello

---

```
class JUCEHelloApplication :public JUCEApplication {
    HelloWorldWindow* helloWorldWindow;
public:

    void initialise (const String& commandLine) {
        helloWorldWindow = new HelloWorldWindow();
    }

    void shutdown() {
        delete helloWorldWindow;
    }
};

START_JUCE_APPLICATION (JUCEHelloApplication)
```



megvalósítandó  
metódusok

# JUCE hello /2

```
class HelloWorldContentComponent :public Component {
public:
    void paint (Graphics& g) { // paint üzenet
        g.fillAll (Colours::white);
        g.setColour (Colours::black);
        g.setFont (20.0f, Font::bold);
        g.drawText (T("Hello World!"),
                    0, 0, getWidth(), getHeight(),
                    Justification::centred, false);
    }
};

class HelloWorldWindow :public DocumentWindow {
public:
    HelloWorldWindow() :DocumentWindow (
        T("Hello World"), Colours::yellow,
        DocumentWindow::allButtons, true ) {
        setContentComponent (
            new HelloWorldContentComponent());
        setVisible (true);
    }
};
```

interfész  
jelleg

# *FLTK toolkit*

- C++ alapú, kihasználja a C++ lehetőségeit
- kis méretű
- FLUID (Fast Light User-Interface Designer)
- platform független (X, MS, MAC)
- makróktól mentes
- OpenGL integráció, GLUT kompatibilis
- GNU
- uCFLTK mikrokontrollerekhez

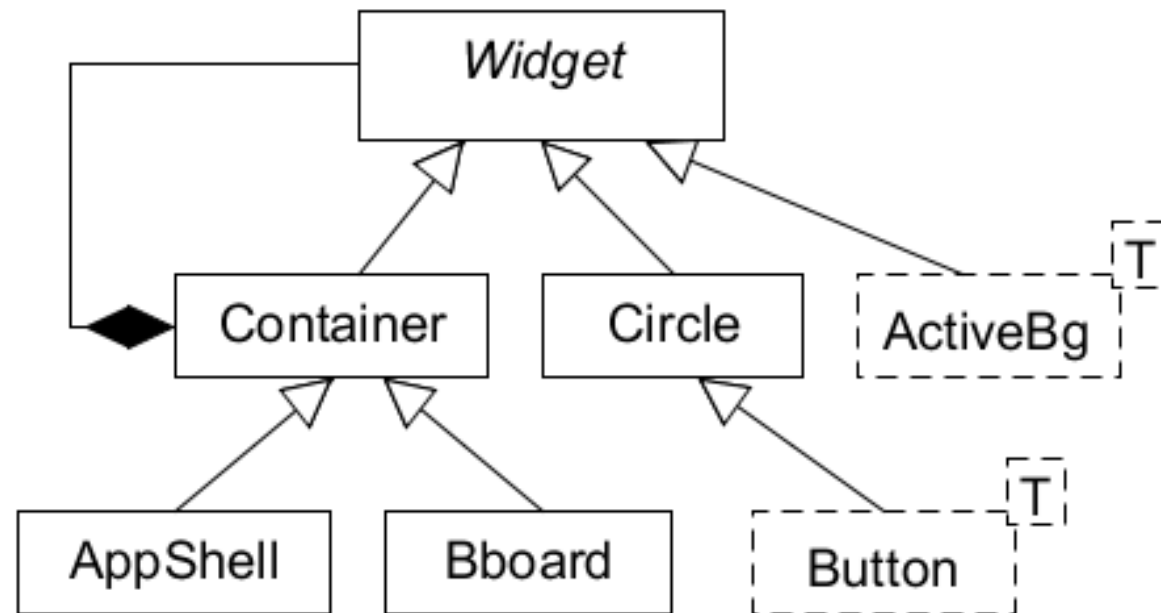
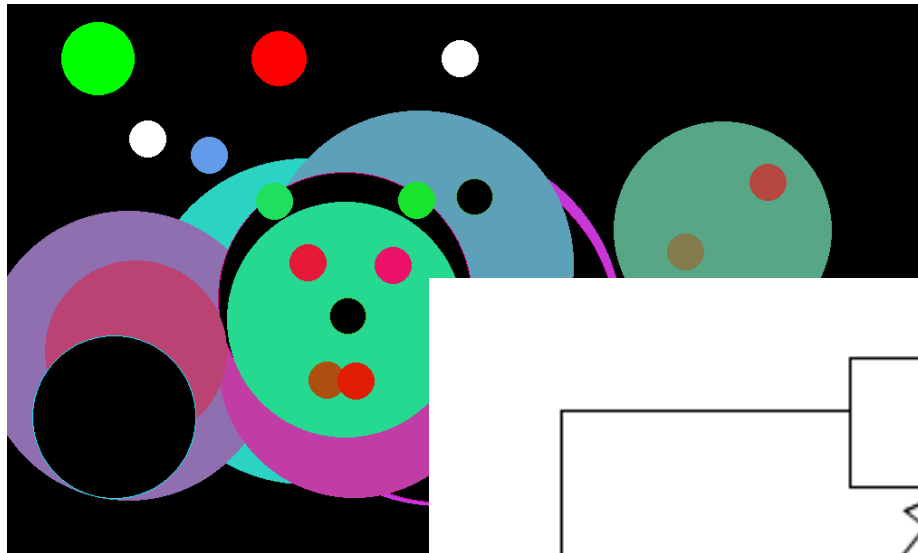
<http://www.fltk.org>

# FLTK hello

```
class Hello :public Fl_Window {
    static void quit (Fl_Widget*,void*) {// privát m.
        exit(0);
    }
public:
    Hello(int w, int h, const char *n=0)
        :Fl_Window(w, h, n) {
        Fl_Button *bt =
            new Fl_Button(10, 10, 100, 25, "Exit");
        bt->callback(quit); // cb. fv. összerendelés
        callback(quit);
        show(); // megjelenítés
    }
};

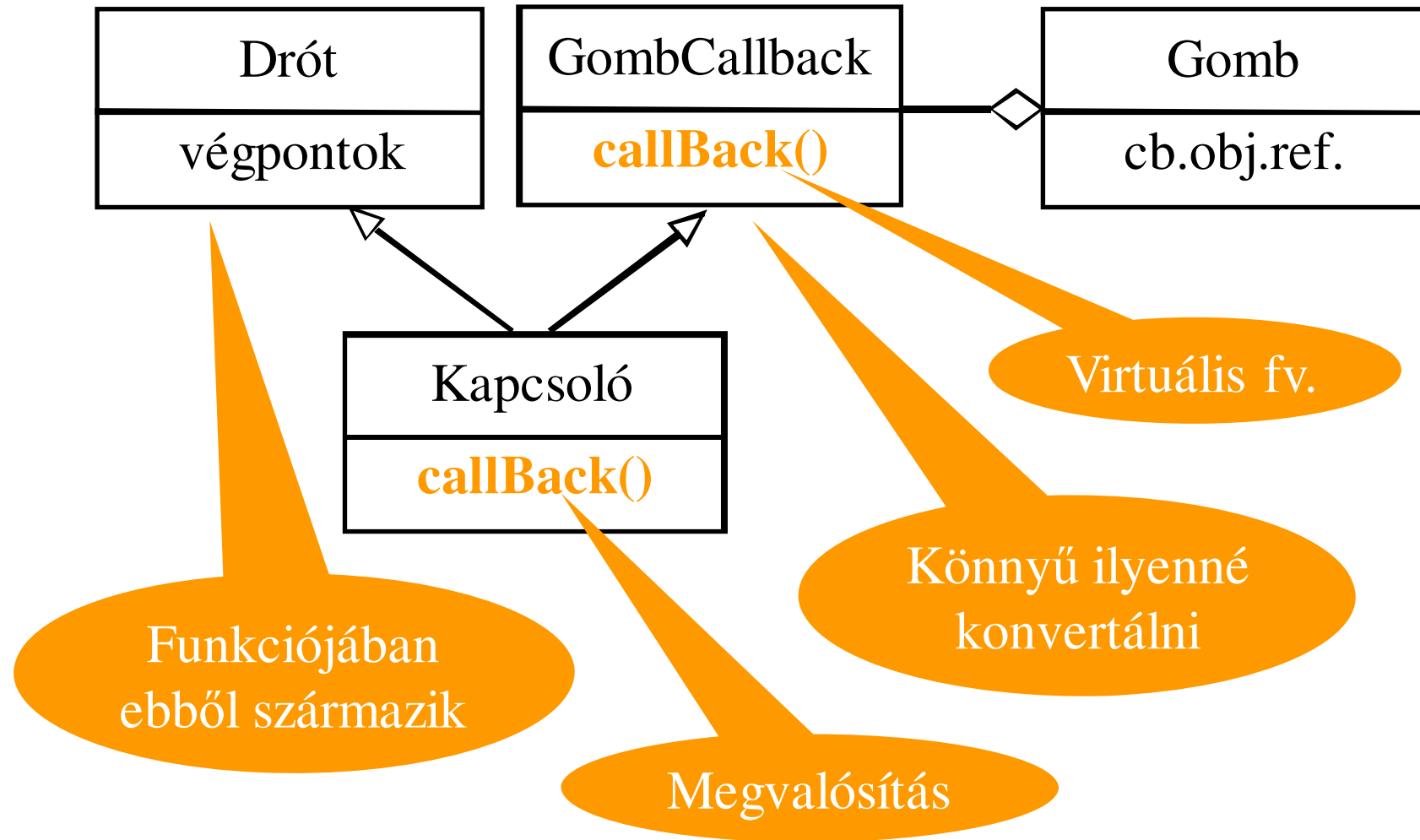
int main() {
    Hello hel(400, 200, "Hello");
    return Fl::run(); // eseménykezelés indul
}
```

# SDL\_bboard demo





# callback mechanizmus



# Kapcsoló megvalósítása

```
class GombCallback {           // callback funkcióhoz
public:
    virtual void callBack() = 0; // virtuális cb. függvény
};

class Gomb { // felhasználói felület objektuma
    GombCallback &cb; // objektum referencia
public:
    Gomb (GombCallback &t) :cb(t) {} // referencia inic.
    void Nyom() { cb.callBack(); } // megnyomták
    ....
};
```


## *Kapcsoló megvalósítása/2*

```
class Kapcsoló :public Drot, public GombCallback {  
    int be;                // állapot  
public:  
    void ki();  
    void be();  
    void callBack() { if (be) ki(); else be(); } // callback  
};  
...  
Kapcsoló k1;  
Gomb g1(k1); // kapcsoló és a callBack fv. összerendelése
```

# *signal/slot mechanizmus*

- Az örökléssel megvalósított callback mechanizmus nagyon szoros csatolást jelent a két objektum között, ráadásul nem típusbiztos.
- signal/slot lényegesen lazább csatolást jelent.

 → signal

slot → 

```
Boost:  
struct Hello {  
    void operator() { ..... }  
};  
boost::signal<void ()> sig;  
Hello hello;  
sig.connect(hello);  
sig();
```

# *signal/slot mechanizmus/2*

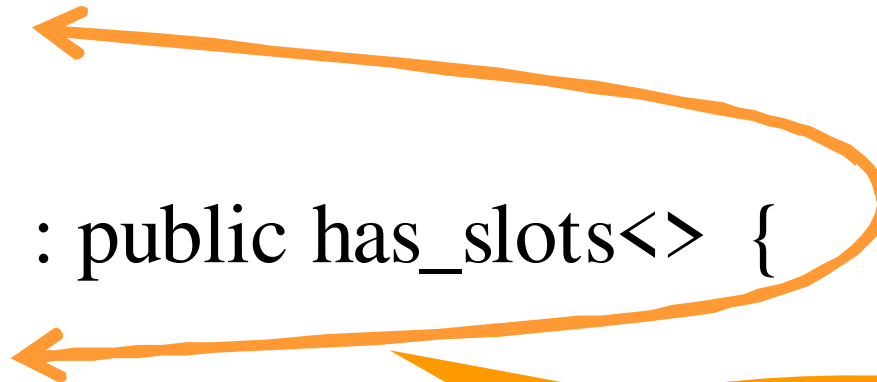
- Ez a mechanizmus világosabban lehetővé teszi a callback függvények összerendelését.
- Kevésbé szoros csatolás ad, ugyanakkor paraméterezzhető és típusos.

```
boost::signal<float (int, int)> sig;  
cout << sig(1, 2);
```

- Megvalósítás:
  - template-tel (boost, sigslot)
  - preprocesszorral (Qt)

## Példa: sigslot lib

```
struct Gomb {  
    signal0<> kapcsol;  
    ...kapcsol();  
};  
struct Kapcsoló : public has_slots<> {  
    void be();  
};  
Gomb g1; Kapcsoló k1;  
g1.Kapcsoló.connect(&k1, &kapcsoló::be);
```



A kapcsoló fv. hívás operátorát összeköti

## Példa: Qt

```
struct Gomb : public QObject{
    Q_OBJECT
    signals:
        void kapcsol(); };
```

moc kulcsszó  
(preproceszor)

```
struct Kapcsoló : public QObject{
    Q_OBJECT
    public slots:
        void be();
}; // moc (Meta-object compiler)
```

# Qt toolkit

- C++ alapú
- kiegészítő utasítások → előfeldolgozó
- platform független (X, MS, MAC)
- OpenGL integráció, GLUT kompatibilis
- 2008: Nokia
- 2009-től LGPL, QPL, és üzleti licenc
- 2012: Digia, 2014-től Qt Company
- számos további nyelv:
  - Python, C#, Ruby, Ada, Perl, PHP, Haskell
- Migrációs lehetőségek (MFC, Motif)

<http://qt-project.org>, <http://qt.digia.com/>



# Qt hello

---

```
#include <QApplication>
#include <QPushButton>

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);

    QPushButton hello("Hello world!");

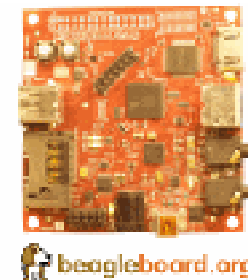
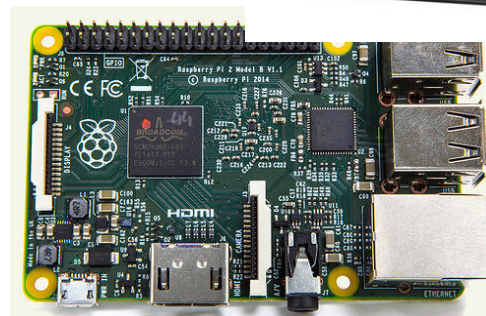
    connect(&hello, SIGNAL(clicked()), &app,
           SLOT(quit()));

    hello.resize(100, 30);

    hello.show();
    return app.exec();
}
```

# Qt platformok

- Android
- Blackberry
- iOS
- Linux/X11
- Mac OS X
- ~~Simbian~~
- Windows
- Windows CE
- Raspberry Pi



# *Wt toolkit*

---

- Egy érdekes példa a Wt (Web Toolkit)
- Segítségével teljesen C++-ban írhatunk meg egy web alkalmazást, ami minden szokásos dolgot tartalmazhat pl. sessionkezelést is.
- Nem kell ismerni egyéb technológiát mint pl: HTML, CSS, Java, php, stb.

<http://www.webtoolkit.eu>

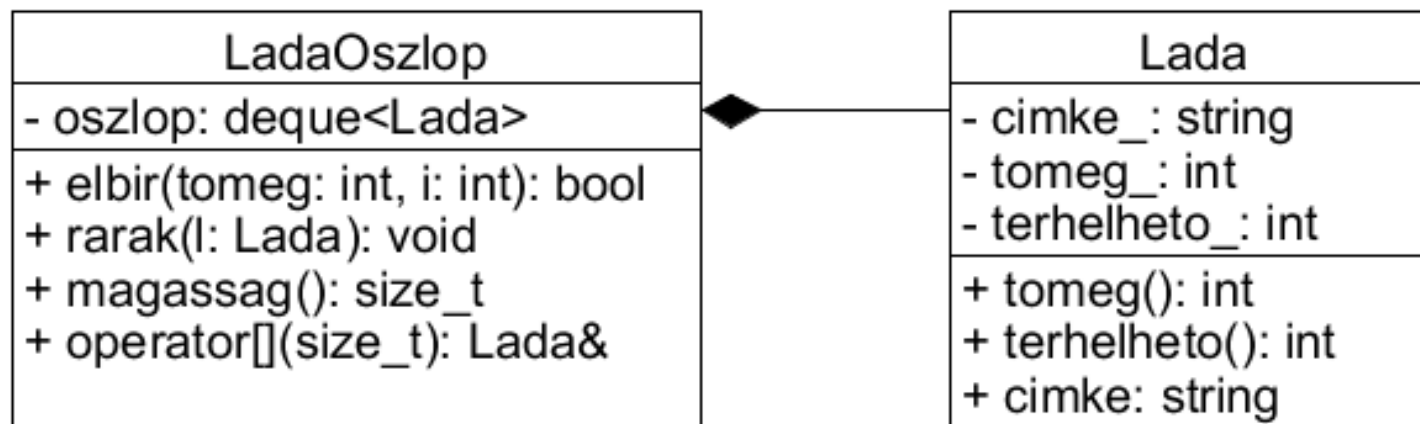
# Wt hello

```
WApplication *createAppl(const WEnvironment& env) {
    WApplication *appl = new WApplication(env);
    appl->setTitle("Hello world!");
    appl->root()->addWidget(
        new WText(L"<h1>Hello, World!</h1>"));
    WPushButton *Button = new
        WPushButton(L"Quit", appl->root());
    Button->clicked.connect(SLOT(appl,
        WApplication::quit));
    return appl;
}

int main(int argc, char **argv) {
    return WRun(argc, argv, &createApplication);
}
```

# 11. heti labor példa

- Ládákat modellezünk. Minden ládának van felirata, teherbírása és tömege.
- A ládákból ládaoszlopokat építünk. A teherbírás túllépésekor a láda összetörik.

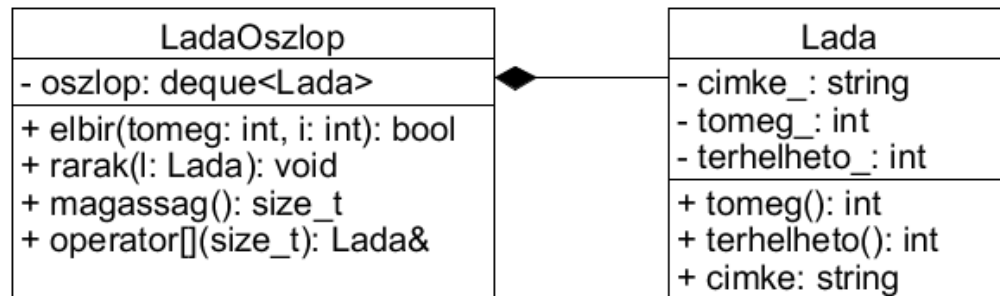


# Lada.h

```
class Lada {
    std::string cimke_;    ///< Láda felirata
    int tomeg_;           ///< Láda tömege
    int terhelheto_;      ///< Láda max. terhelhetősége
public:
    Lada(std::string cimke, int tomeg=10, int m=100)
        :cimke_(cimke), tomeg_(tomeg), terhelheto_(m) {}
    // Visszaadja a láda tömege.
    int tomeg() const { return tomeg_ ;}
    // Láda maximális terhelhetősége
    int terhelheto() const { return terhelheto_ ;}
    // Szöveges cimke.
    std::string cimke() const { return cimke_ ;}
};
```

# LadaOszlop.h

```
class LadaOszlop {
    std::deque<Lada> oszlop; ///< ládak (első felül)
public:
    bool elbir(int t, size_t i) const ;
    // Uj ládát helyez az oszlop tetejére.
    void rarak(Lada l);
    // Ládák száma az oszlopban.
    size_t magassag() const;
    // A ládák elérésére indexeléssel
    Lada& operator[](size_t i);
};
```



# LadaOszlop.cpp

```
bool LadaOszlop::elbir(int t, size_t i) const {
    for (size_t j = 0; j < i; j++)
        t += oszlop[j].tomeg();
    return oszlop[i].terhelheto()>=t;
}

void LadaOszlop::rarak(Lada l) {
    for (int i=oszlop.size()-1; i>=0; i--)
        if (!elbir(l.tomeg(), i))
            oszlop.erase(oszlop.begin()+i); //random iter
    oszlop.push_front(l);
}

Lada& LadaOszlop::operator[](size_t i) {
    return oszlop[i];
}
```



# LadaWidget.h

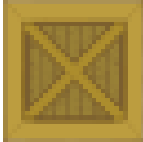
```
#include <Wt/WContainerWidget>
#include <Wt/WTable>
#include "ladak.h"

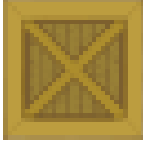
/// Az oszlopot reprezentáló webes kezelőelem
class LadaWidget : public Wt::WTable, public LadaOszlop{
public:
    LadaWidget(Wt::WContainerWidget* parent = 0);
    /// Új lada hozzáadása.
    hozzáad(Lada l);
    // Megjeleníti a ladakat.
    // @param t - felső láda tömege
    void rajzol(int t);
};
```

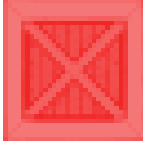
# Megjelenítés, hogyan?

Tömeg:

Hozzáad

 3. lada (3;10)

 2. lada (5;10)

 1. lada (2;10)

# LadaWidget.cpp

```
#include <Wt/WContainerWidget>
#include <Wt/WLabel>
#include <Wt/WTable>
#include <Wt/WImage>
#include "ladawidget.h"

using namespace Wt;

LadaWidget::LadaWidget (WContainerWidget* parent)
                                : WTable (parent) {}

void LadaWidget::hozzaad(Lada l) {
    rarak (l);
    rajzol (l.tomeg ());
}
```

# *LadaWidget.cpp*

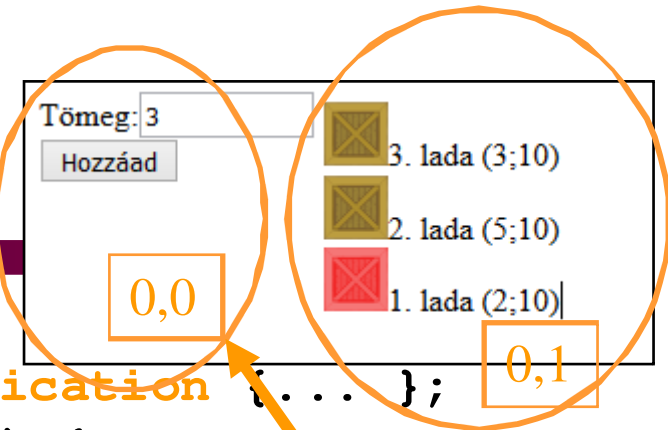
```
void LadaWidget::rajzol(int t) {
    WTable::clear();    // toroljuk a megj. ladakat

    for (size_t i = 0; i < magassag(); i++) {
        WLabel *l =
            new WLabel((*this)[i].cimke(), elementAt(i, 0));
        WImage *kep;
        if (elbir(t, i))
            kep = new WImage("img/chest.png");
        else
            kep = new WImage("img/chest-red.png");
        l->setImage(kep);
    }
}
```

# LadaApplication.h

```
class LadaApplication : public WApplication {
    int db;          ///< Ládák egyedi címkézéséhez
    LadaWidget *ladak;    ///< A ládák táblázata
    WLineEdit *tomegLine; ///< Tömeg beviteli mező
    // Új láda hozzáadása
    void hozzaad() {
        int tomeg = lineToint(tomegLine);
        int max = 10;    // most fix
        std::ostringstream ss;
        ss << ++db << ". lada (" << tomeg << ";"
            << max << ")";
        ladak->hozzaad(*new Lada(ss.str(), tomeg, max));
    }
public:
    LadaApplication(const WEnvironment& env);
};
```

# LadaApplication



```
class LadaApplication : public WApplication { ... };
LadaApplication::LadaApplication(...) {
    WTable* layout = new WTable();
    root()->addWidget(layout);
    WTable* panel = new WTable(layout->elementAt(0,0));
    ladak = new LadaWidget(layout->elementAt(0,1));
    layout->elementAt(0,1)->setPadding(5);
    WLabel* label = new WLabel(WString::tr("TÖMEG"),
                               panel->elementAt(0,0));
    tomegLine = new WLineEdit("3",
                              panel->elementAt(0,0));
    tomegLine->setValidator(new WIntValidator(0, 200));
    label->setBuddy(tomegLine); // a felirat melle
    WPushButton* button=new WPushButton(WString::tr("AD"),
                                         panel->elementAt(2,0));
```

# LadaWidget

```
// eseménykezelő függvények bekötése
button->clicked().connect(this,
                           &LadaApplication::hozzaad);
tomegLine->keyWentUp().connect(this,
                                &LadaApplication::tomegValtozott);
}

WApplication *createApplication(const
                                WEnvironment& env) {
    return new LadaApplication(env);
}

int main(int argc, char **argv) {
    return WRun(argc, argv, &createApplication);
}
```

# *Felhasználói input*

## **Egysoros editbox**

```
tomegLine = new WLineEdit("3",  
                           panel->elementAt(0,0));
```

## **Nyomógomb esemény bekötése:**

```
button->clicked().connect(this,  
                           &LadaApplication::hozzaad);
```

## **Billentyű felengedése az editboxban:**

```
tomegLine->keyWentUp().connect(this,  
                                &LadaApplication::tomegValtozott);
```



# Megjelenítés

Beviteli mezők

Tömeg:

Max:

hu

9. lada (40;100)

8. lada (10;60)

7. lada (10;100)

6. lada (10;100)

5. lada (10;100)

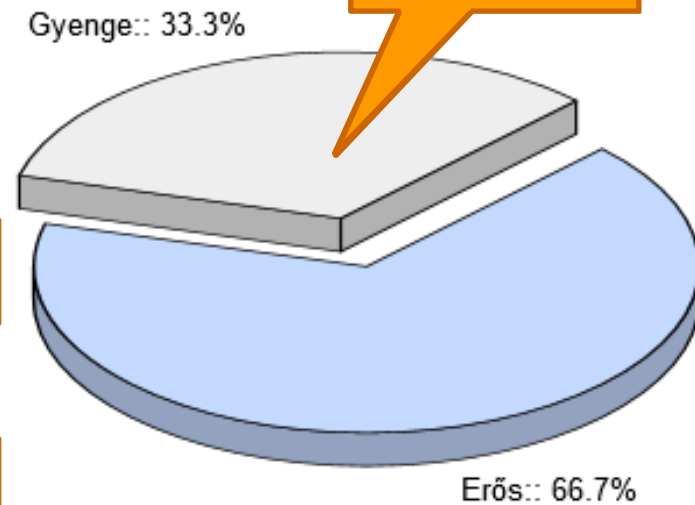
4. lada (10;100)

Nyomógomb

Választási lehetőség

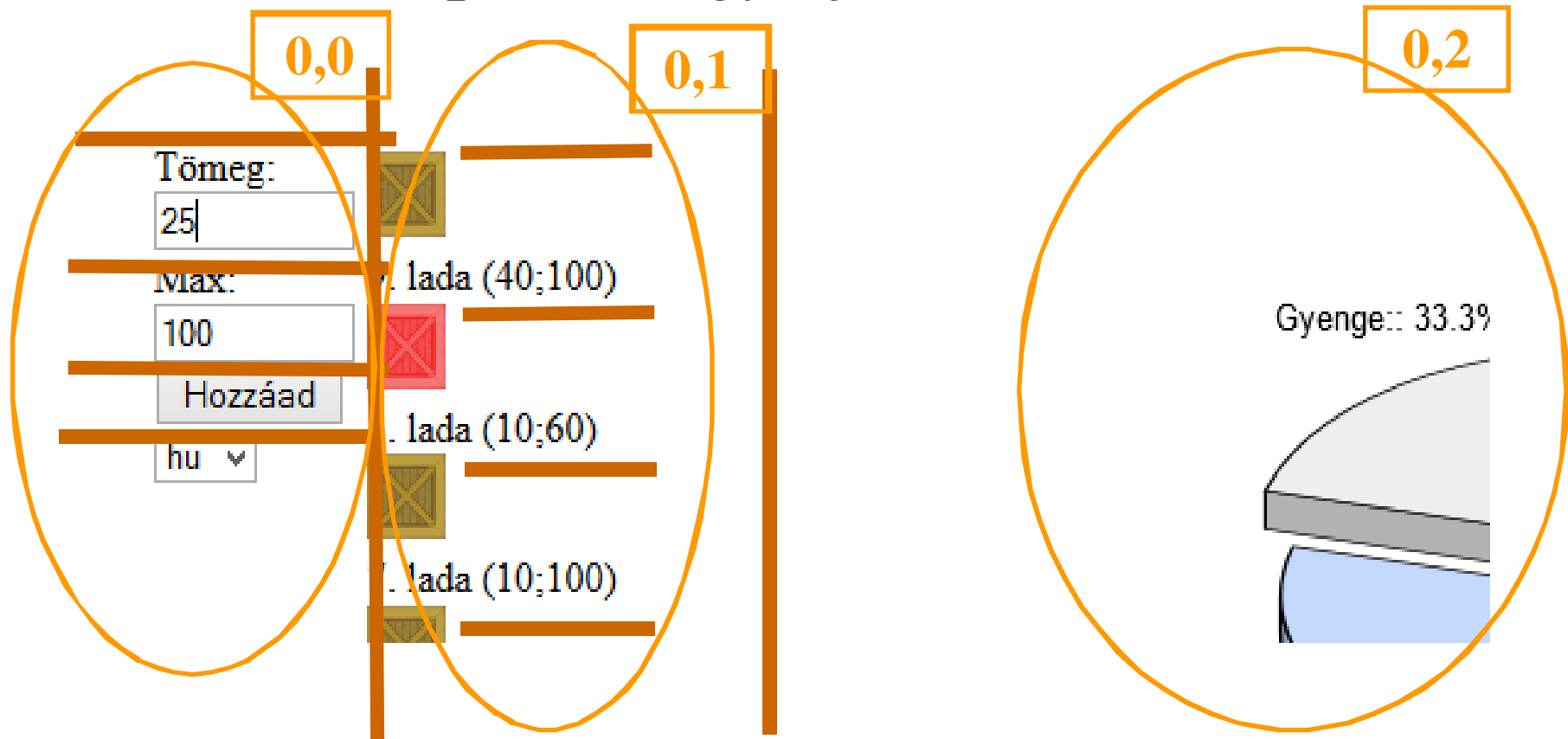
Gyenge ládák  
színezése

Diagram



# Fő grafikus elemek helye

Táblázatokkal helyezük el a widgeteket.  
A ládaoszlop belül egy újabb táblázat.



# *Internationalization & Localization*

---

- i18n = internatiloization
- l10n = localization (kulturális beágyazás)

Különböző jelek, formátumok

- dátum, idő,
- pénznem, jelek, mértékegységek

Különböző nyelveken

# *i18n és l10n támogatása*

---

- GNU gettext
- Wt::WMessageResourceBundle
- OASIS XLIFF
- POSIX catalogs
- Qt ts/tm
- Java properties
- Windows resources
- ...

# WMessageResourceBundle

- A `Wstring::tr` metódusa egy belső azonosító alapján a nyelvi környezetnek megfelelő xml formátumú fájlból olvassa ki szöveget. pl:

```
WPushButton (WString::tr ("HOZZAAD"), ...
```

```
<messages>
```

```
  <message id='TOMEG'>Tömeg:</message>
```

```
  <message id='HOZZAAD'>Hozzáad</message>
```

```
</messages>
```

```
<messages>
```

```
  <message id='TOMEG'>Mass:</message>
```

```
  <message id='HOZZAAD'>Add</message>
```

```
</messages>
```

app.xml

app\_en.xml

# *Fejlesztés támogatása*

---

- Sok állomány → egy alkalmazás
- Hogyan és melyiket kell lefordítani?
- Melyik változat az legutolsó?
- IDE (integrált fejlesztő eszköz)
  - fordítást belső eszközzel támogatja (nyelvet ismeri)
  - verziókövetést külső eszközzel
- Önálló univerzális eszközök
  - nem csak az adott nyelvhez

# make

- Egy szöveges leírás (*Makefile*), és az *állományok módosítási ideje* alapján végrehajtja cél (program, dokumentáció stb.) előállításához szükséges parancsokat.
- Makefile:
  - makró definíciók,
  - függőségi információk (szabályok és implicit szabályok)
  - végrehajtható parancsok
  - megjegyzések

# *Makefile szerkezete*

---

## **Makródefiníció:**

```
makró_név = string
```

## **Szabályok:**

```
cél1 [cél2] [::] [feltétel1...] [;parancsok] [#...]  
[<TAB>parancsok][#...]
```



# *make példa*

```
prog:      x.o y.o z.o
           cc x.o y.o z.o -o prog
x.o:      x.c x.h
           cc -c x.c
y.o:      y.c x.h
           cc -c y.c
z.o:      z.c
           cc -c z.c
```

feltételek

cél

parancs

!!!!!! A parancsok előtt <TAB> van !!!!!

## *make példa/2*

```
OBJECTS = x.o y.o z.o
```

```
HEADS = x.h
```

```
prog: $(OBJECTS)
```

```
$(CC) $(OBJECTS) -o prog
```

```
$(OBJECTS) : $(HEADS)
```

```
.c.o:
```

```
$(CC) -c $<
```

implicit szabály miatt a `.o`-k egyértelműen előállíthatók.

# *make változatok*

---

Számos változata és kiegészítése van:

- make (eredeti)
- BSD make
- GNU make
- nmake (Microsoft)
- CMake – cross platform make