



Bevezetés a programozásba 2

10. Előadás: Esettanulmány

A “siető diák módszer”

- A kódba “láttam már hasonlót” elv alapján részletek bevágása
- A hibaüzenetek eltüntetésének vágya annak megértése helyett
- “előbb bepötyögök mindent, aztán kijavítom a hibákat”
- “ha elég sokáig változtatom véletlenszerűen a programot, előbb-utóbb működni fog”

Módszer – munkabeosztás

- Inkrementális programozás:
 - kijelölünk egy részmegoldás-sorozatot, aminek az eleje valamilyen stabil megoldás (“hello world”, példaprogramok), a vége a feladat megoldása, és a közbeeső lépések egyesével
 - nem nehezek
 - külön tesztelhetőek
 - néhány sor módosításával megoldhatóak
 - így lépésről lépésre haladhatunk, és mindig van biztos pont, amihez visszatérhetünk, ha gond van
 - Gond általában van. Nem gondolunk mindig mindenre előre.

Módszer – kód szerkezet

- Struktúrált programozás
 - szervezzük függvényekbe a teendőket, felső szinten absztrakt függvényekbe, onnan az egyre konkrétabbak felé alfüggvényhívásokba, hierarchikus szerkezetbe
 - top-down, felülről lefelé szervezés
 - redukcionista felfogás
 - a módosítási kényszer miatti változtatási kör alfüggvény szinten marad, kis változás a feladatban kis változást jelent a kódban

Módszer – kód szerkezet

- Objektum orientált programozás
 - a funkciót a feldolgozandó/tárolandó adatok szerint csoportosítsuk, az osztályok általában egyenrangúak
 - bottom-up, alulról felfelé építkezés
 - moduláris felfogás
 - módosítási kényszernél az ábrázolás határáig terjed a változtatási kör, tehát a láthatósági szabályozás miatt osztályszinten marad, bővíteni pedig szinte fájdalommentesen lehet

Esettanulmány: Widgetek

- A következő példa egy leegyszerűsített Widgetkészlet kezdete, leszámaztatással működő gomb megoldással
- **Közvetlenül a leadott beadandóba másolni veszélyes**

Esettanulmány: Widgetek

- Tervezési fázis:
 - milyen szolgáltatásokat szeretnénk?
 - soreditort, gombot
 - eseményvezérelt működést
 - OOP tervezés: mik közösek a gombokban és a soreditorban (és a jövőben hozzáadandó widgetekben)?
 - eseménykezelő és rajzoló műveletek felülete
 - hely, méret, fókuszmanipuláció

Widget

```
class Widget {
public:
    Widget(int x, int y, int sx, int sy)
        : _x(x), _y(y), _sx(sx), _sy(sy) {}
    bool inside(int x, int y) {
        return x > _x && x < _x+_sx && y > _y && y < _y+_sy;
    }
    virtual void show() {}
    virtual void handleEvent(event ev) {}
    void setfocus(bool x) { _isfocused=x;}
protected:
    int _x, _y, _sx, _sy;
    bool _isfocused;
};
```


Widget

```
class Widget {  
public:  
    Widget(int x, int y, int sx, int sy)  
        : _x(x), _y(y), _sx(sx), _sy(sy) {}  
    bool inside(int x, int y) {  
        return x > _x && x < _x + _sx && y > _y && y < _y + _sy;  
    }  
    virtual void show() {}  
    virtual void handleEvent(event ev) {}  
    void setfocus(bool x) { _isfocused=x;}  
protected:  
    int _x, _y, _sx, _sy;  
    bool _isfocused;  
};
```

Megjelenítés és
eseménykezelés
felület. Ezeket a
függvényeket konkrétan
sosem fogjuk meghívni,
csak a leszármazottakét

Widget

```
class Widget {
public:
    Widget(int x, int y, int sx, int sy)
        : _x(x), _y(y), _sx(sx), _sy(sy) {}
    bool inside(int x, int y) {
        return x > _x && x < _x+_sx && y > _y && y < _y+_sy;
    }
    virtual void show() = 0;
    virtual void handleEvent(event ev) = 0;
    void setfocus(bool x) { _isfocused=x; }
protected:
    int _x, _y, _sx, _sy;
    bool _isfocused;
};
```

Megjelenítés és
eseménykezelés
felület. Ezeket a
függvényeket konkrétan
sosem fogjuk meghívni,
csak a leszármazottakét

Widget

```
class Widget {
public:
    Widget(int x, int y, int sx, int sy)
        : _x(x), _y(y), _sx(sx), _sy(sy) {}
    bool inside(int x, int y) {
        return x > _x && x < _x+_sx && y > _y && y < _y+_sy;
    }
    virtual void show() = 0;
    virtual void handleEvent(event ev) = 0;
    void setfocus(bool x) { _isfocused=x;}
protected:
    int _x, _y, _sx, _sy;
    bool _isfocused;
};
```

Fókuszkezelés:
nem kell virtuális
tagfüggvény hozzá,
mert Widget szinten
intézhető a részfeladat

Tanulságok

- Ha közös felület, akkor virtuális függvény
- Ha közös funkció, akkor mezők és tagfüggvények az őssosztályban, amit az örökösök majd megkapnak
- Ha sok hasonló dologra van szükség, akkor
 - ezek vagy különböznek **funkcióban**, és öröklődéssel érdemes megoldani a problémát
 - vagy ugyanazok a funkcióik, csak az adataik különböznek, és nincs szükség több típusra

Widgetek

- A Widget osztály egységes felülete miatt az eseményciklus teendője kiemelhető:
 - minden, ami a képernyőn megjelenik, az Widget leszármazott, és egységes felületen kommunikál
 - ezért közös adatszerkezetbe foghatóak
 - és a kontroll kezelése is egységessé válik
- A kontroll fogalma: “ahol épp fut a program” avagy “ki hív kit”. Az eseményvezérelt program és a hagyományos közötti különbség a kontroll kezelése.

Application

```
class Application {
public:
    Application(int x, int y): _x(x), _y(y) {
        _exit=false; _w.clear(); _focus=-1;}
    void add(Widget *w) { _w.push_back(w); }
    void run() {
        gout.open(_x, _y);
        event ev;
        while (gin && !_exit) {
            for (unsigned int i=0; i<_w.size(); i++) _w[i]->show();
            gout << refresh;
            gin >> ev;
            if (ev.type==ev_mouse) {
// eseménykezelés: ha az egéresemény _w[_focus]->inside()
// akkor _w[_focus]->handleEvent, különben körülnézünk, hogy
// ki lesz a fókuszban, vagy épp senki sem.
                }
                if (_focus!=-1)
                    _w[_focus]->handleEvent(ev);
            }
        }
        void shutdown() {_exit=true;}
protected:
    vector<Widget *> _w;
    int _focus;
    bool _exit;
    int _x, _y;
};
```

Application

```
class Application {
public:
    Application(int x, int y): _x(x), _y(y) {
        _exit=false; _w.clear(); _focus=-1;}
    void add(Widget *w) { _w.push_back(w); }
    void run() {
        gout.open(_x,_y);
        event ev;
        while (gin && !_exit) {
            for (unsigned int i=0;i<_w.size();i++) _w[i]->show();
            gout << refresh;
            gin >> ev;
            if (ev.type==ev_mouse) {
// eseménykezelés: ha az egéresemény _w[_focus]->inside()
// akkor _w[_focus]->handleEvent, különben körülnézünk, hogy
// ki lesz a fókuszban, vagy épp senki sem.
                }
                if (_focus!=-1)
                    _w[_focus]->handleEvent(ev);
            }
        }
        void shutdown() {_exit=true;}
protected:
    vector<Widget *> _w;
    int _focus;
    bool _exit;
    int _x, _y;
};
```

Widgetek kezelése,
számoltartása

Application

```
class Application {
public:
    Application(int x, int y): _x(x), _y(y) {
        _exit=false; _w.clear(); _focus=-1;}
    void add(Widget *w) { _w.push_back(w); }
    void run() {
        gout.open(_x,_y);
        event ev;
        while (gin && !_exit) {
            for (unsigned int i=0;i<_w.size();i++) _w[i]->show();
            gout << refresh;
            gin >> ev;
            if (ev.type==ev_mouse) {
// eseménykezelés: ha az egéresemény _w[_focus]->inside()
// akkor _w[_focus]->handleEvent, különben körülnézünk, hogy
// ki lesz a fókuszban, vagy épp senki sem.
                }
                if (_focus!=-1)
                    _w[_focus]->handleEvent(ev);
            }
        }
        void shutdown() {_exit=true;}
protected:
    vector<Widget *> _w;
    int _focus;
    bool _exit;
    int _x, _y;
};
```



Kontroll kezelése

LineEdit

```
class LineEdit : public Widget {
public:
    LineEdit(int x, int y, int sx, int sy, string s) :
        Widget(x,y,sx,sy), _s(s) {}
    virtual ~LineEdit() {}
    virtual void show() {
        gout << /*keret, háttértörlés*/ << text(_s);
        if (_isfocused) gout << text("|");
    }
    virtual void handleEvent(event ev) {
        if (/*betű jött és _s még nem túl hosszú*/)
            _s+=ev.keycode;
        if (ev.keycode == key_backspace)
            _s=_s.substr(0,_s.length()-1);
    }
    string value() {return _s;}
protected:
    string _s;
};
```

Widgetek

- Ezzel az alapok megvannak:
 - van sok Widget-utódunk, és az Application osztja a kontrollt, nincs más dolgunk, mint leszármaztatni
 - kivéve a gombokat, amikhez külön-külön funkció tartozik, és ezeket valahol meg kell írni
- Mégis, van közös minden gombban, ezeket érdemes összefogni egy ősgombban.

Button

```
class Button : public Widget {
public:
    Button(string s, int x, int y, int sx, int sy)
        : Widget (x,y,sx,sy), _s(s) {}
    virtual ~Button() {}
    virtual void action() {}
    void show() {
        if (_isfocused) {
            //aktív háttér
        } else {
            //passzív háttér
        }
        gout << /*pozíció, szín*/ << text(_s);
    }
    virtual void handleEvent(event e) {
        if (inside(e.pos_x, e.pos_y)) {
            if (e.button==btn_left) action();
        }
    }
protected:
    string _s;
};
```

ExitButton

```
class ExitButton : public Button{
public:
    ExitButton(string s, Application &app, int x, int y,
               int sx, int sy)
        : Button(s,x,y,sx,sy), _app(app) {}
    virtual ~ExitButton() {}
    void action() {
        _app.shutdown();
    }
protected:
    Application &_app;
};
```

Widgetek

- Eddig minden oké, de mi a helyzet akkor, ha a widgeteknek egymással kell kommunikálniuk?
 - Egyelőre a widgetek egymástól függetlenek
 - Az Application bővítése nem megoldás: minden alkalmazásban más és más kommunikációt igénylő helyzet állhat elő
 - Sebaj: Készítsünk leszármazottat az Applicationból, és bővítsük a widgetekkel és a közös tevékenységekkel
 - Figyelem: elértük az általánosság határát, ami innentől jön, az alkalmazásfüggő

MyApplication és OkButton

```
class OkButton;
class MyApplication : public Application{
public:
    MyApplication(int x, int y);
    void okButton() {
        ofstream of("output.txt");
        of << le->value();
        of.close();
    }
private: ExitButton * xb; LineEdit * le; OkButton * ob;
};

class OkButton : public Button{
public:
    OkButton(string s, MyApplication &app, int x, int y, int sx, int sy)
        : Button(s,x,y,sx,sy), _app(app) {}
    virtual ~OkButton() {}
    void action() {_app.okButton();}
protected:
    MyApplication &_app;
};

MyApplication::MyApplication(int x, int y) : Application(x,y) {
    xb = new ExitButton("Exit",*this, 10, 60, 100, 40);
    le = new LineEdit(10,10,280,40,"line editor");
    ob = new OkButton("Save",*this, 150, 60, 100, 40);
    add(xb);    add(le);    add(ob);
}
```

MyApplication és OkButton

```
class OkButton;
class MyApplication : public Application{
public:
    MyApplication(int x, int y);
    void okButton() {
        ofstream of("output.txt");
        of << le->value();
        of.close();
    }
private: ExitButton * xb; LineEdit * le; OkButton * ob;
};

class OkButton : public Button{
public:
    OkButton(string s, MyApplication &app, int x, int y, int sx, int sy)
        : Button(s,x,y,sx,sy), _app(app) {}
    virtual ~OkButton() {}
    void action() {_app.okButton();}
protected:
    MyApplication &_app;
};

MyApplication::MyApplication(int x, int y) : Application(x,y) {
    xb = new ExitButton("Exit",*this, 10, 60, 100, 40);
    le = new LineEdit(10,10,280,40,"line editor");
    ob = new OkButton("Save",*this, 150, 60, 100, 40);
    add(xb);    add(le);    add(ob);
}
}
```

MyApplication és OkButton

```
class OkButton;
class MyApplication : public Application{
public:
    MyApplication(int x, int y);
    void okButton() {
        ofstream of("output.txt");
        of << le->value();
        of.close();
    }
private: ExitButton * xb; LineEdit * le; OkButton * ob;
};

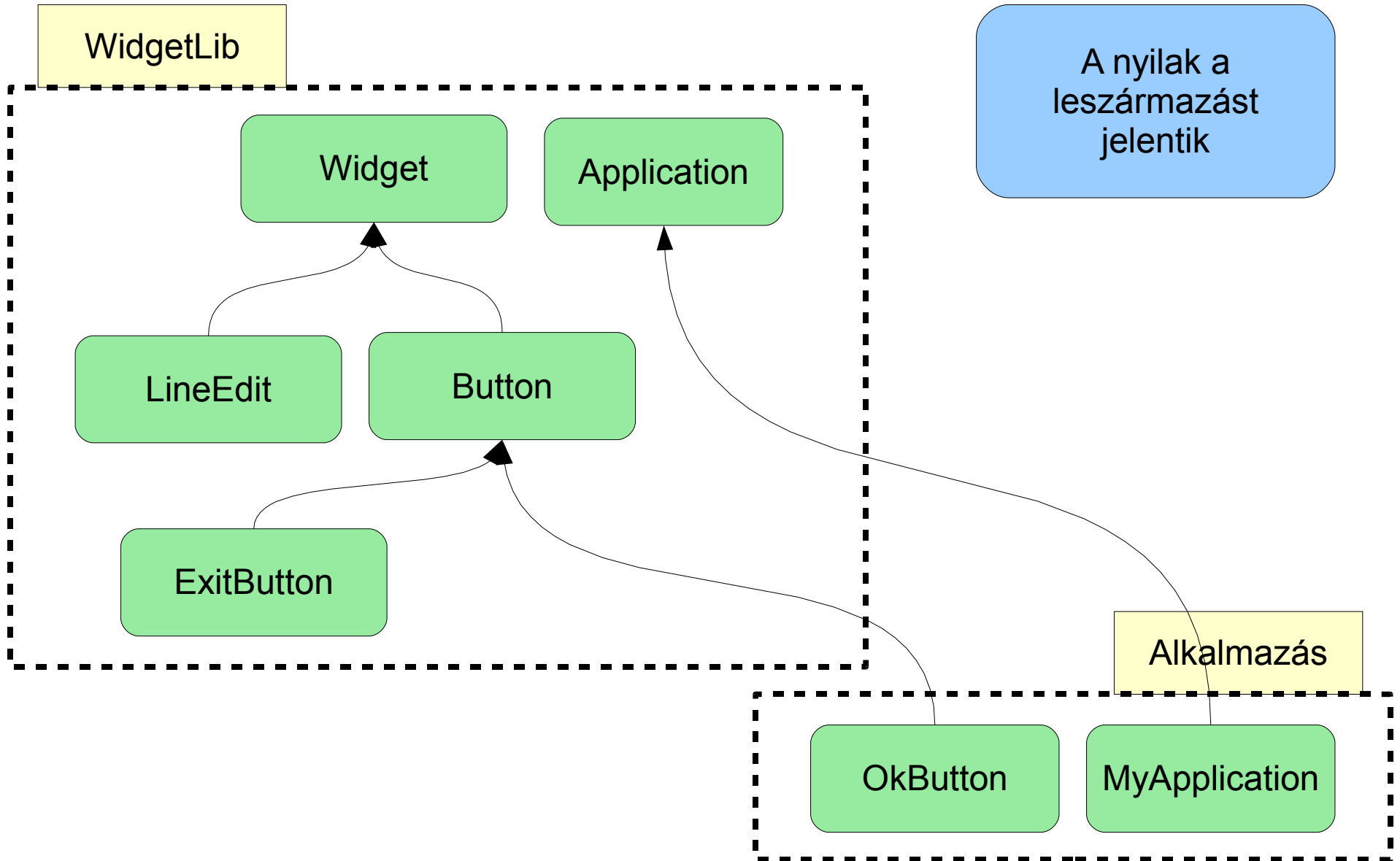
class OkButton : public Button{
public:
    OkButton(string s, MyApplication &app, int x, int y, int sx, int sy)
        : Button(s,x,y,sx,sy), _app(app) {}
    virtual ~OkButton() {}
    void action() {_app.okButton();}
protected:
    MyApplication &_app;
};

MyApplication::MyApplication(int x, int y) : Application(x,y) {
    xb = new ExitButton("Exit", *this, 10, 60, 100, 40);
    le = new LineEdit(10,10,280,40,"line editor");
    ob = new OkButton("Save", *this, 150, 60, 100, 40);
    add(xb);    add(le);    add(ob);
}
}
```

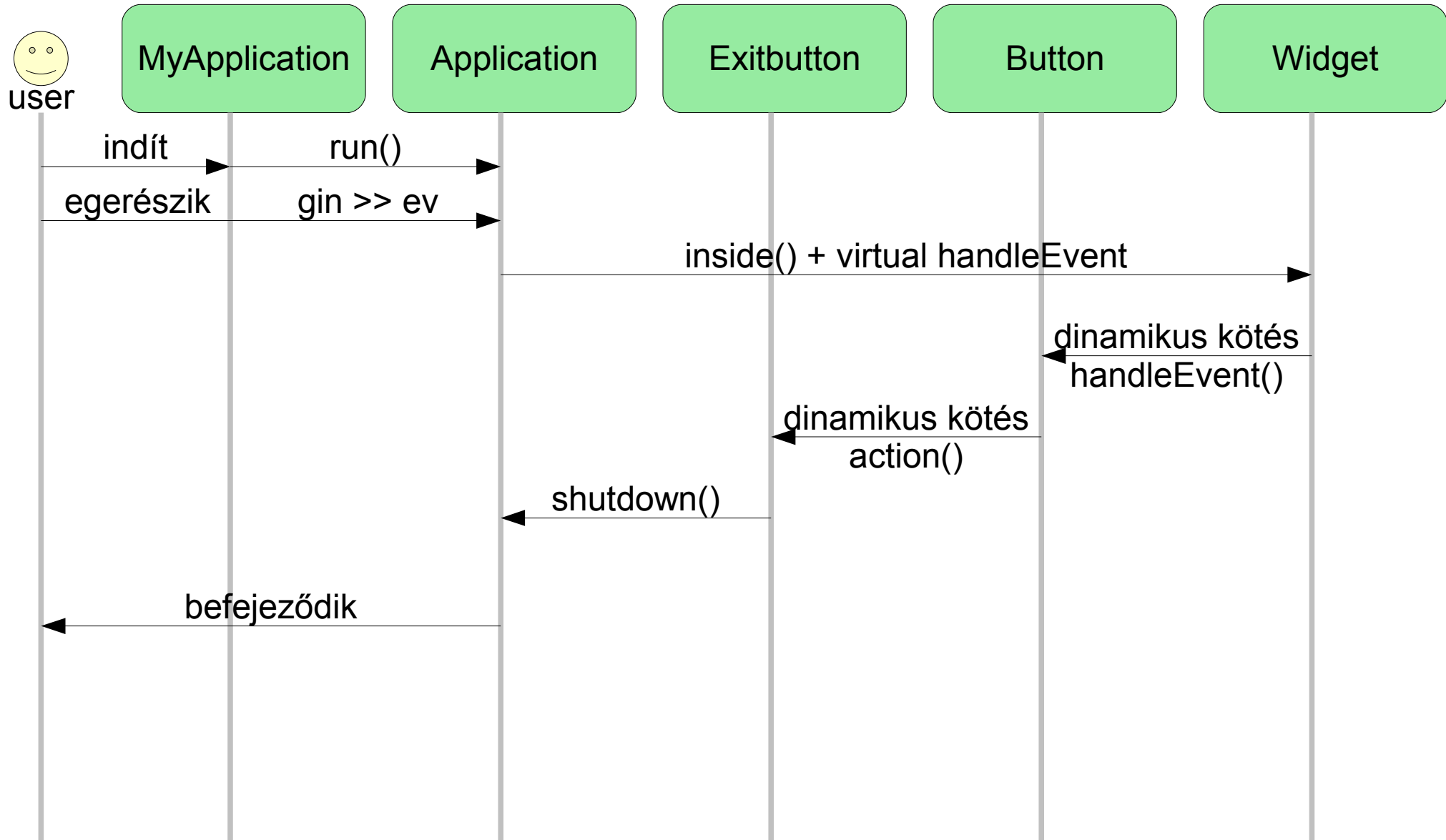

main()

```
int main()  
{  
    MyApplication app(300,200);  
    app.run();  
}
```

Rendszer



A gombnyomás folyamata



Tapasztalatok

- Az öröklődés lehetősége új és erőteljes eszköz
 - Újrafelhasználhatóság a leszámaztatáson keresztül
 - Dinamikus kötés révén már akkor egységesen kezelhetünk objektumokat, mikor azok még nem is léteznek
 - Nem látunk a jövőbe
 - Nem kellett a még nem létező elemek beépíthetőségéért extra munkát végezni

Alternatív lehetőségek

- Lehetséges lett volna
 - már Widget szinten kezelni az egyes alapvető eseményeket, és virtuális “onMouseButtonPressed” metódusokat használni
 - olyan Widget örököst használni, ami más Widgetek tartalmazására képes. Ez lehet például a Window, és a programot a MyApplication és a MyWindow megírásával oldjuk meg.
 - gomboknál függvénymutatókkal vagy funktorokkal dolgozni.
 - Widgetnek Application mezője, mint kötelező elem
 - stb..

Jótanácsok

- Érdeemes azzal kezdeni a beadandót, hogy ezt a példát megérted, kipróbálad, kiegészíted.
- És aztán az előző alternatív lehetőségek választéka szerint továbbgondolod, hogy neked mi a legszimpatikusabb.
 - Ennek pedig az a módja, hogy tesztalkalmazásokat készítesz. Sokat, még a géptermi ZH előtt.