Eseménykezelés

Gregorics Tibor

gt@inf.elte.hu

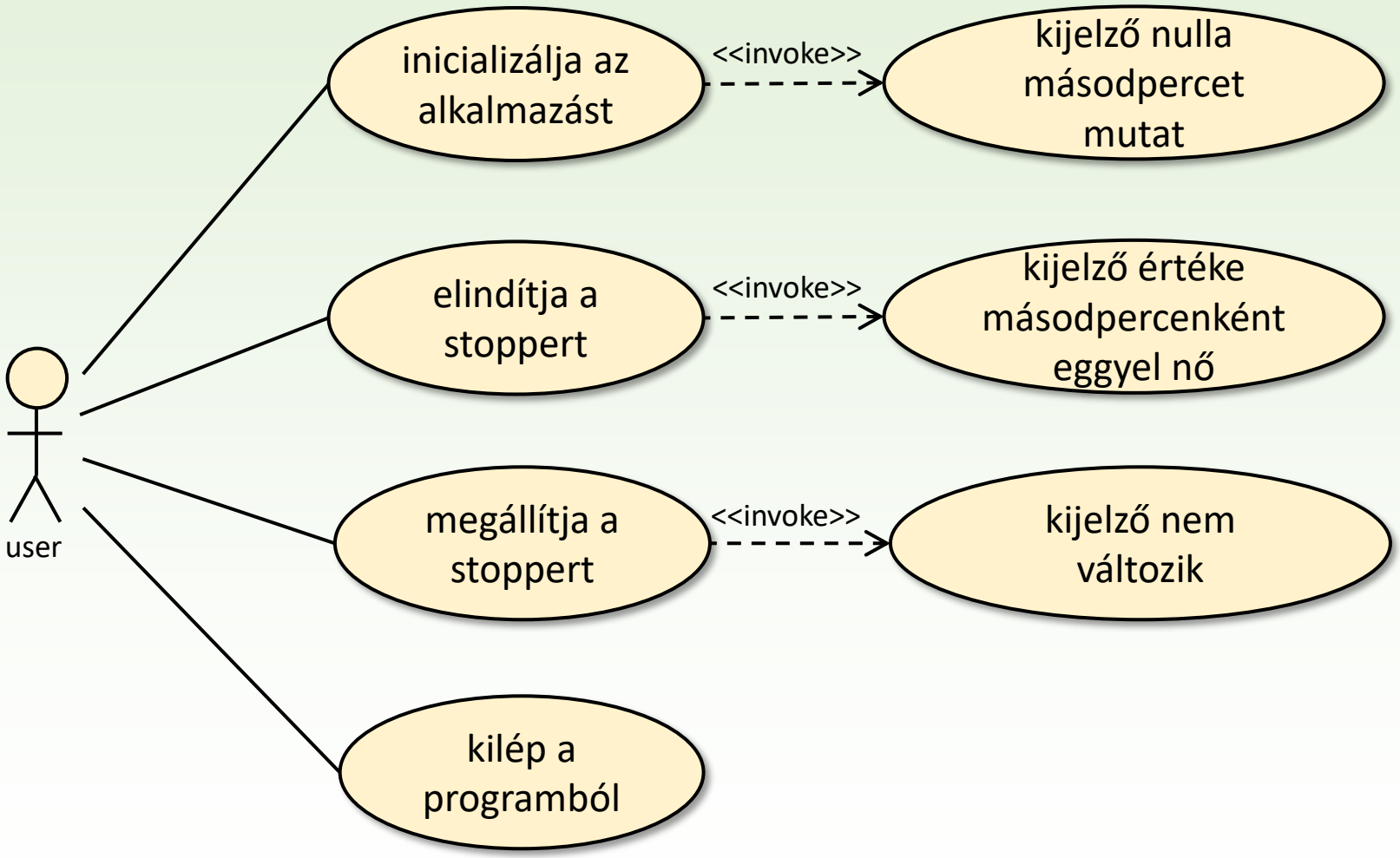
http://people.inf.elte.hu/gt/oep

Feladat

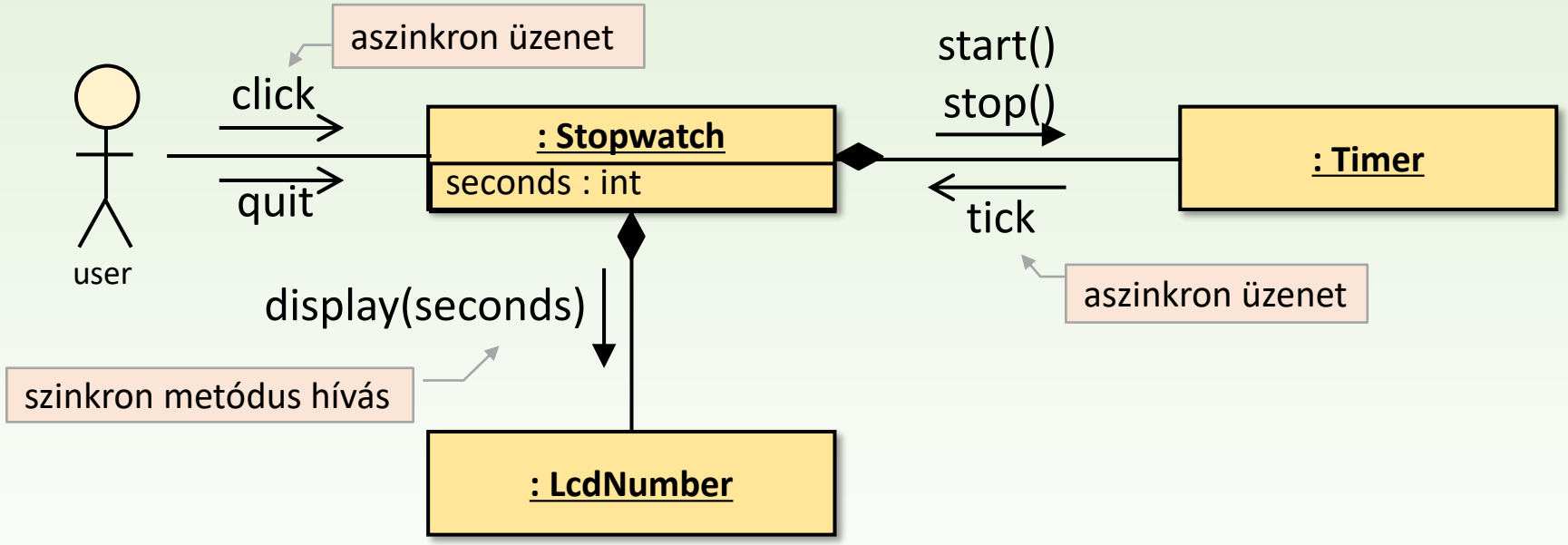
Készítsünk egy stoppert, amely másodpercenként jelzi a múló időt. Ez a folyamat

- egy jelzés hatására induljon el, és ugyanezen jelzés hatására álljon le;
- illetve ezen jelzés ismétléseinek hatására váltakozva folytatódjon tovább vagy szüneteljen;
- az alkalmazás befejezésére külön jelzést használjunk.

Használati eset diagram

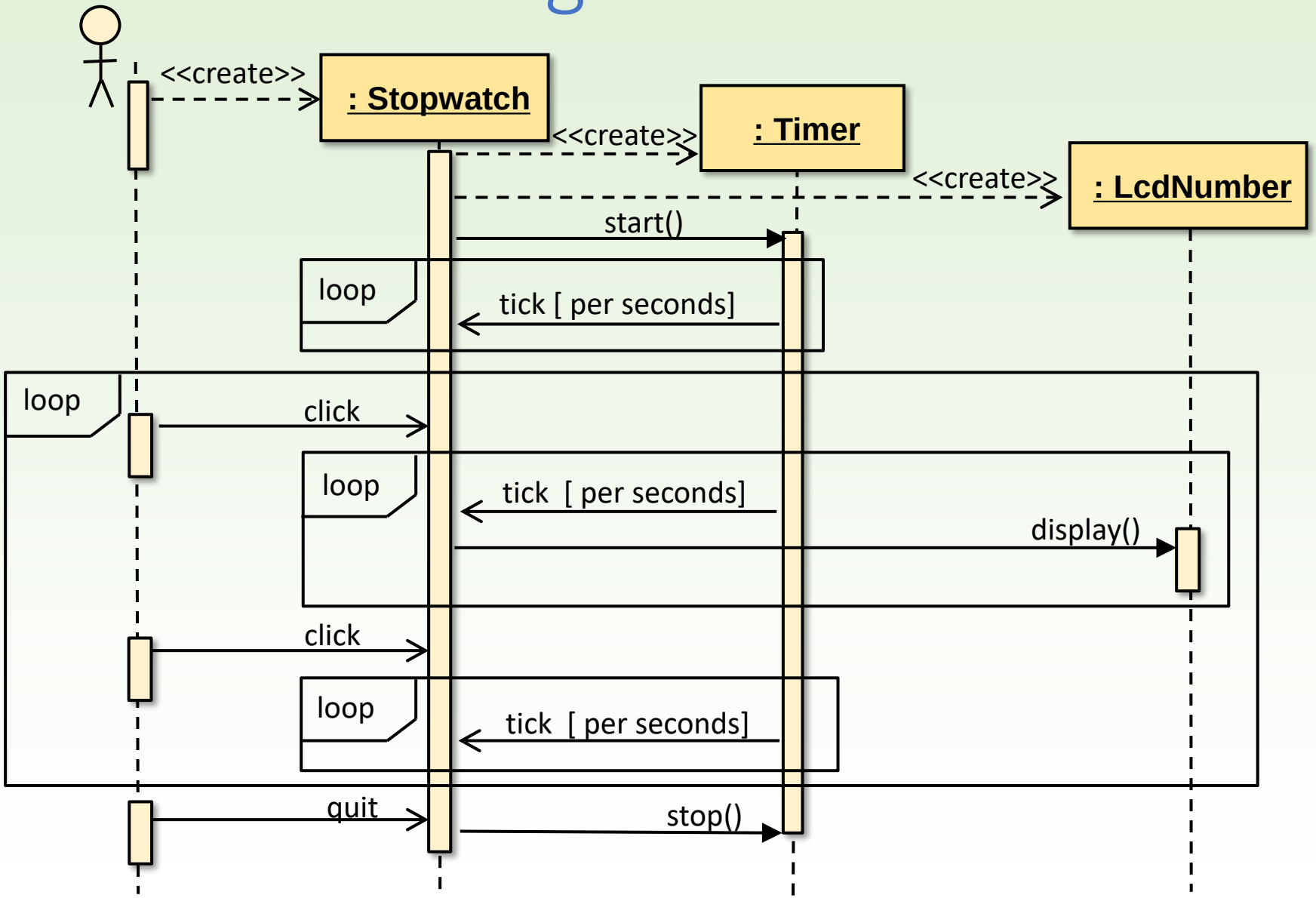


Kommunikációs diagram + Objektum diagram

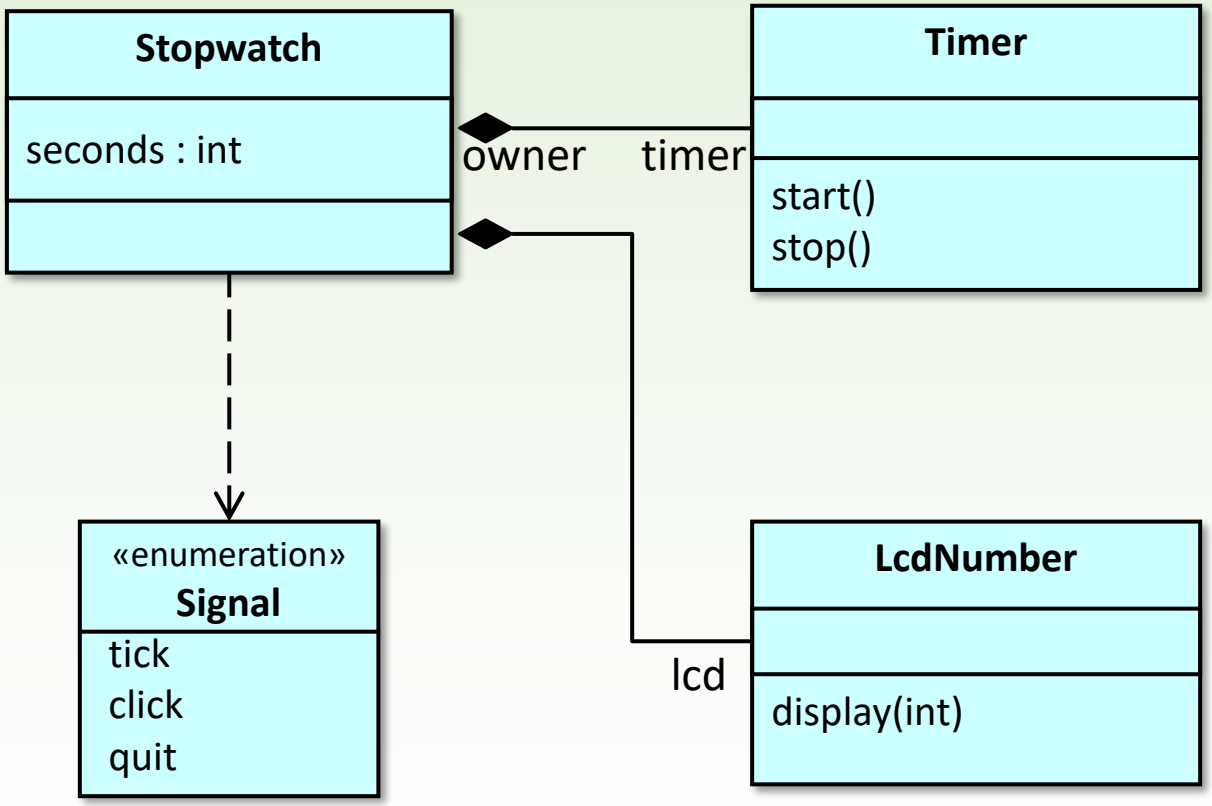


- a felhasználó és az időzítő is szignálokat küld a stoppernek
- a stopper a számára küldött szignálokat aszinkron módon dolgozza fel, azaz nem blokkolhatja egy új üzenet beérkezését azért, mert egykorábbi üzenetre reagál éppen.

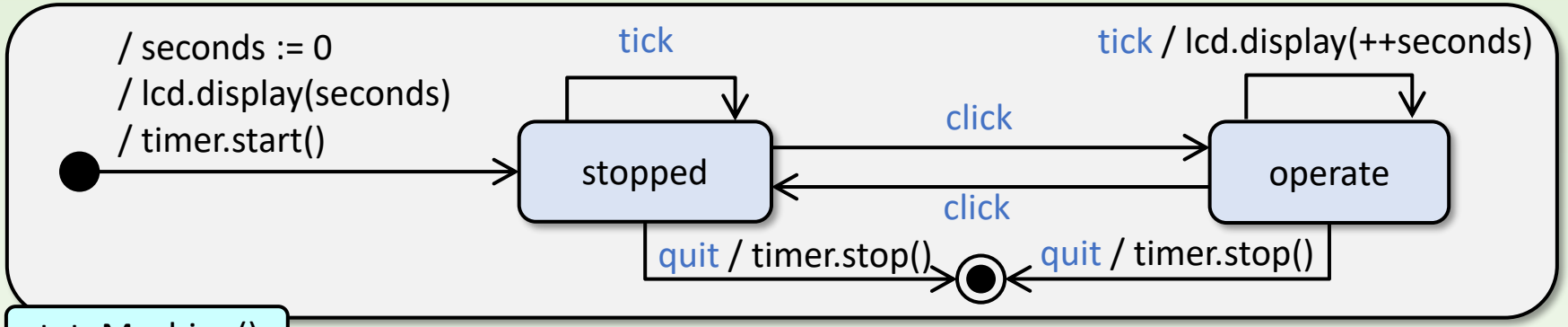
Szekvencia diagram



Osztálydiagram



Stopwatch állapotgépe



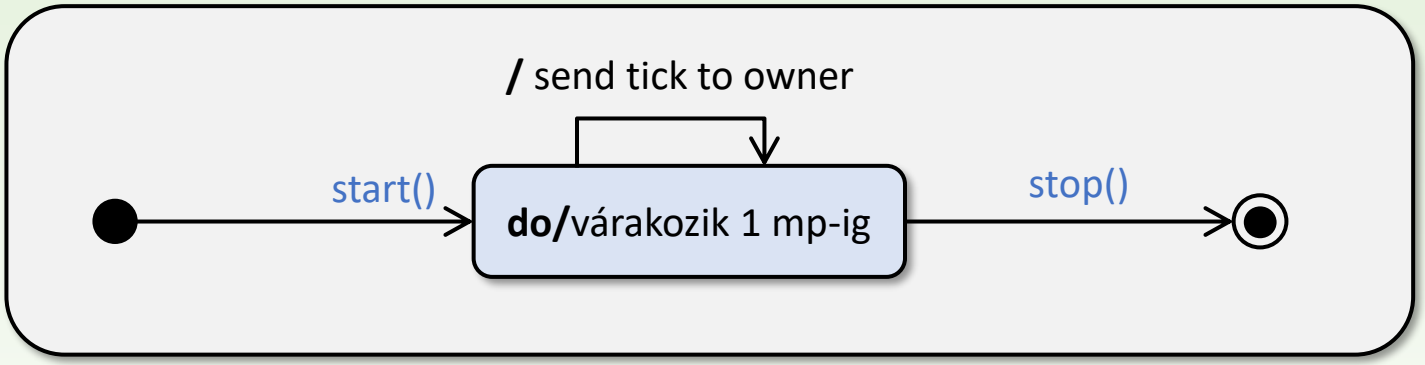
stateMachine()

```
seconds := 0
lcd.display(seconds)
timer.start()
currentState := stopped
while currentState ≠ final loop
   currentState := transition(getSignal(), currentState)
endloop
```

állapot-átmenet tábla

	stopped	**operate**
click	operate	stopped
tick	stopped	**do**/ lcd.display(++seconds) operate
quit	**do**/ timer.stop() final	**do**/ timer.stop() final

Timer állapotgépe



start()

```
active := true
stateMachine()
```

stateMachine()

```
while  (active)
    wait 1 sec
    send tick to owner
endwhile
```

stop()

```
active := false
```

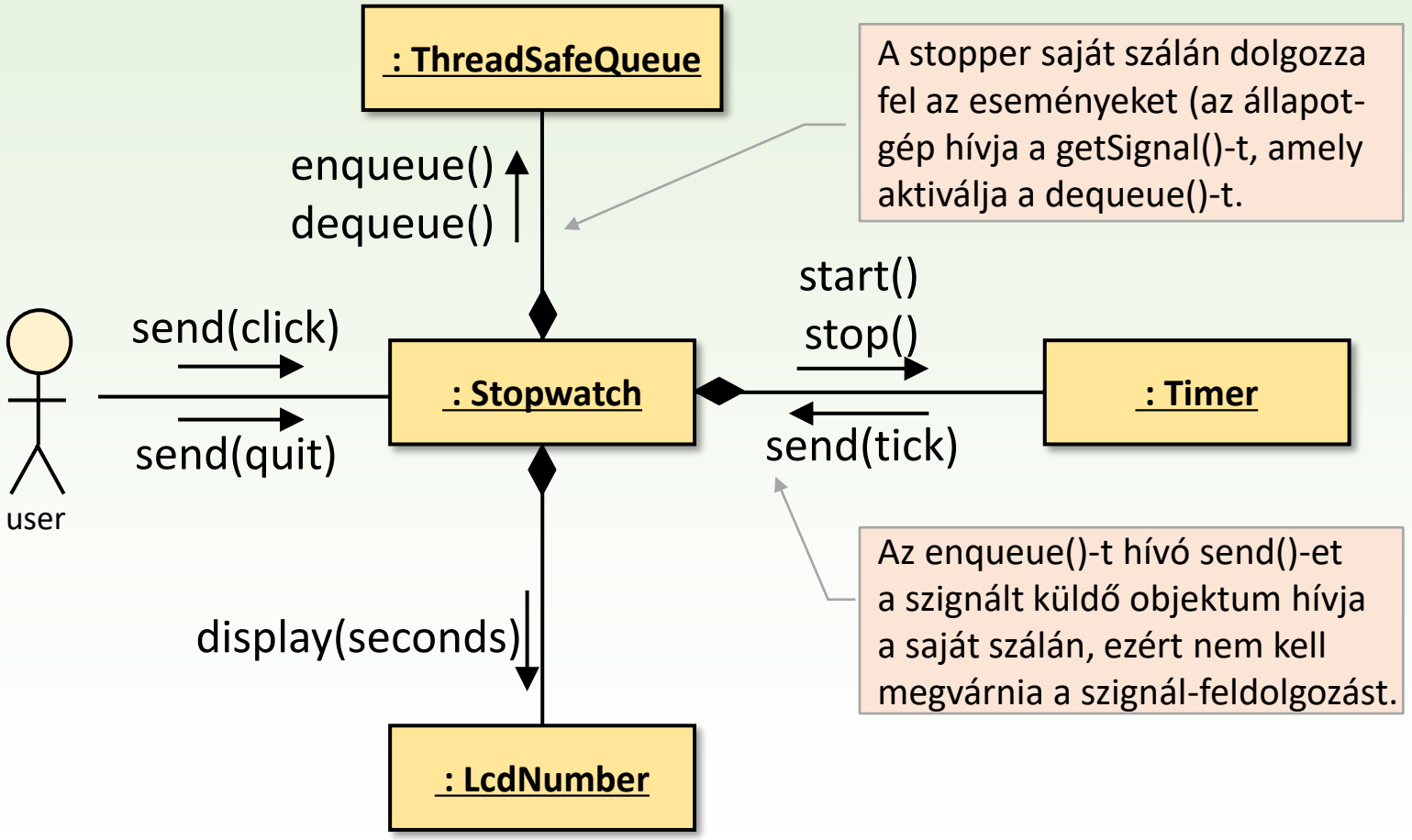
Aktív objektum

- Egy objektum lehet
  - passzív, ha ő csak a valaki mástól kapott üzenet hatására tevékenykedik: változtatja állapotát, küld üzenetet másnak.
  - aktív, ha önerőből is küldhet üzenetet, azaz anélkül, hogy ezt egy mástól kapott üzenet váltaná ki.
- Több aktív objektum együttműködése esetén gondoskodni kell ezek (állapotgépeinek) párhuzamos futtatásáról, azaz többszálú alkalmazásra van szükség. A mi konkrét feladatunk megoldásában
  - a felhasználó (user) egy aktív objektum, amely az alkalmazás főszálán tevékenykedik (elindítja és megállítja a stoppert)
  - az időzítő is aktív objektum (a stoppernek küld másodpercenként szignálokat), ezért külön szálon kell futtatnunk az állapotgépét

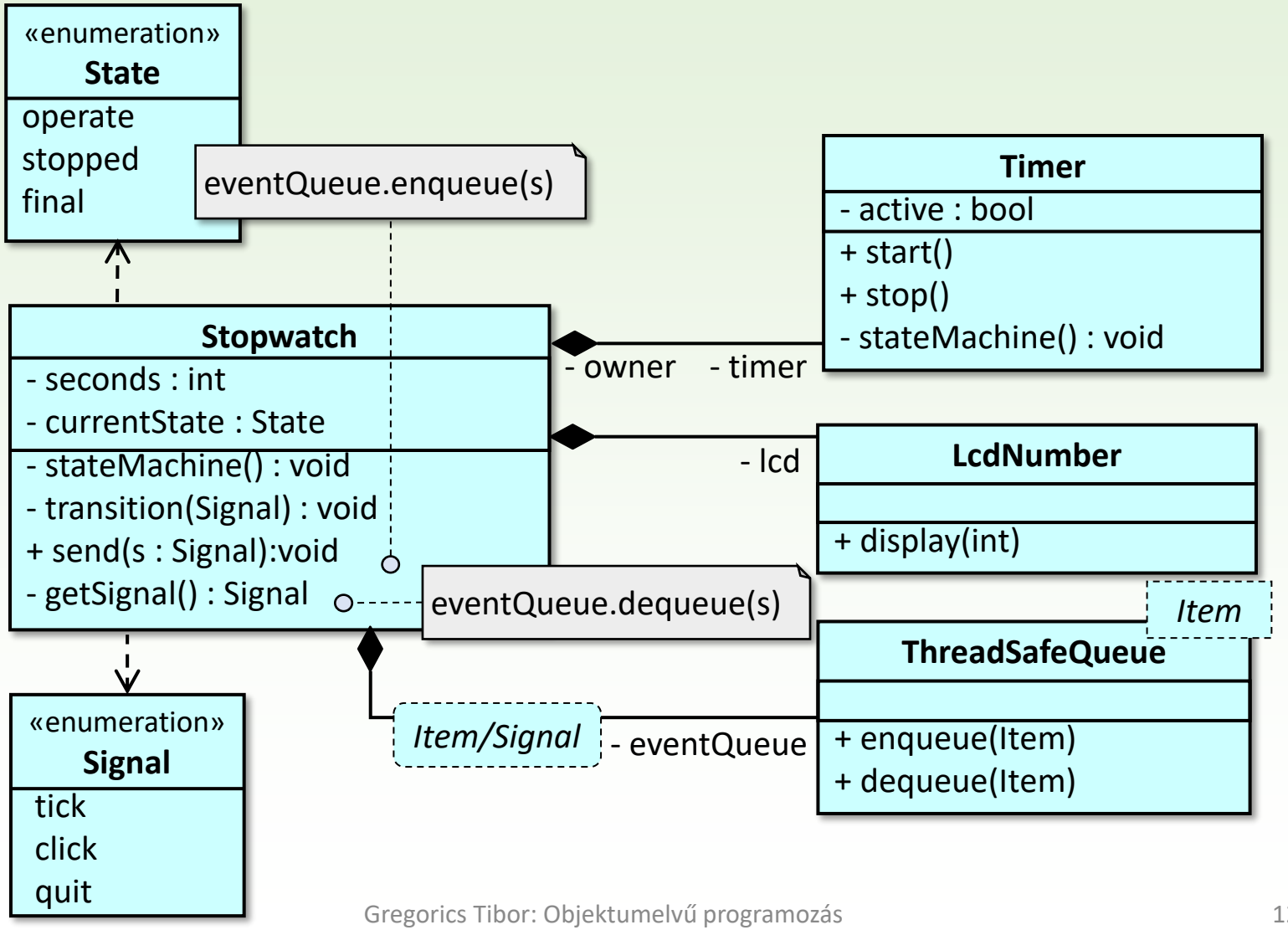
Üzenetek aszinkron feldolgozása

- A stopper egy passzív objektum (reagál a neki küldött szignálokra), de a szignálokat azok beérkezésével aszinkron módon dolgozza fel, ezért az állapotgépe külön szálat igényel.
- A stopper a hozzá több irányból érkező szignálokat egy eseménysorba gyűjti.
  - A küldő objektum (ami most fő program vagy az időzítő) az eseménysor send() metódusának hívásával tehet bele egy szignált a stopper eseménysorába, ennél fogva ez a metódus a küldő objektum szálán fog végrehajtódni.
  - A stopper (saját szálán futó állapotgépe) az eseménysor getSignal() metódusával vehet ki egy szignált az eseménysorból.
  - Az eseménysor betesz és kivesz műveleteit szinkronizálni kell: kölcsönösen kizárásos módon kell működniük. Ráadásul üres sor esetén a kivesz műveletet várakozó utasítással blokkolni kell.

Kommunikációs diagram (megvalósítási szint)



Osztálydiagram (megvalósítási szint)



Stopwatch osztály

```
class Stopwatch {
public:
        Stopwatch();
        ~Stopwatch();
        void send(Signal event) {_eventQueue.enqueue(event);}
private:
      enum State {operate, stopped, final};

        void stateMachine();
        void transition(Signal event);
        Signal getSignal() ;

        Timer _timer;
        LcdNumber _lcd;
        ThreadSafeQueue<Signal> _eventQueue;

        State _currentState;
        int _seconds;

        std::thread _thread;
};
```

enum Signal {tick, click, quit};

külön szál a Stopwatch::stateMachine()-nek
#include <thread>

stopwatch.h

Stopwatch eseménykezelője

stopwatch.cpp

```
void Stopwatch::transition(Signal signal)
{
    switch (_currentState) { // mi az állapot
        case stopped:
            switch (signal) { // mi a szignál
                case click: _currentState = operate; break;
                case tick : break;
                case quit : _currentState = final; break;
            }
            break;
        case operate:
            switch (signal) {// mi a szignál
                case click: _currentState = stopped; break;
                case tick : _lcd.display(++_seconds); break;
                case quit : _currentState = final; break;
            }
            break;
    }
}
```

	stopped	**operate**
click	operate	stopped
tick	stopped	**do**/ lcd.display(++seconds) operate
quit	**do**/ timer.stop() final	**do**/ timer.stop() final

Stopwatch osztály

```
Stopwatch::Stopwatch():_timer(this)
{
	_eventQueue.startQueue();
	_thread = std::thread(&Stopwatch::stateMachine, this);
}

Stopwatch::~Stopwatch()
{
	_thread.join();
	_eventQueue.stopQueue();
}
```

elindul a Stopwatch állapotgépe a külön szálon

bevárja amíg az állapotgép le nem áll

```
void Stopwatch::stateMachine()
{
	_seconds = 0;
	_lcd.display(_seconds);
	_timer.start();
	_currentState = stopped;
	while(_currentState != final ) {
		transition(getSignal());
	}
	_timer.stop();
}

Signal getSignal()
{
	Signal s;
	_eventQueue.dequeue(s);
	return s;
}
```

stopwatch.cpp

Timer osztály

körkörös inkúdolás helyett
(a "timer.h"-t már inklúdolja a "stopwatch.h")

```
class Stopwatch;

class Timer{
        typedef std::chrono::milliseconds milliseconds;
public:
        Timer(Stopwatch *s) : _owner(s), _active(false) {}

        void start() ;
        void stop() ;
private:
        void stateMachine();

        Stopwatch *_owner;
        bool _active;

        std::thread _thread;
};
```

külön szál a Timer::stateMachine()-nek
#include <thread>

timer.h

Timer osztály

```
void Timer::start() {
	_active = true;
	_thread = std::thread(&Timer::stateMachine, this);
}

void Timer::stop() {
	_active = false;
	_thread.join();
}

void Timer::stateMachine(){
	std::condition_variable cond;
	std::mutex mu;
	while (_active){
		std::unique_lock<std::mutex> lock(mu);
		cond.wait_for(lock, milliseconds(1000));
		_owner->send(tick);
	}
}
```

külön szálon indul el a Timer állapotgépe

feltételes változó (cond) és szemafor (mu) a várakozás megvalósításához
#include <condition_variable>
#include <mutex>

egy másodpercig blokkolja a szálat

timer.cpp

LcdNumber osztály

```
class LcdNumber
{
public:
	void display(int seconds)
	{
		std::cout << extend((seconds % 3600) / 60) + ":"
			+ extend((seconds % 3600) % 60) << std::endl;
	}
private:
	std::string extend(int n) const
	{
	   std::ostringstream os;
	   os << n;
	   return (n < 10 ? "0" : "") + os.str();
	}
};
```

lcdnumber.h

ThreadSafeQueue osztálysablon

```
template <typename Item>
class ThreadSafeQueue
{
public:
    ThreadSafeQueue() { _active = false; }

    void enqueue(const Item& e);
    void dequeue(Item& e) ;

    void startQueue()  { _active = true; }
    void stopQueue()   { _active = false; _cond.notify_all(); }
    bool empty() const { return _queue.empty(); }
private:
    std::queue<Item> _queue;

    bool _active;
    std::mutex _mu;
    std::condition_variable _cond;
};
```

a _cond objektummal blokkolt összes szálnak (azaz a dequeue()-t használó Stopwatch állapotgépének) engedélyezi a folytatást

szemafor az enqueue() és dequeue() kölcsönös kizárásos módon való használatához
#include <mutex>

feltételes változó a dequeue() várakoztatásához
#include <condition_variable>

threadsafequeue.hpp

ThreadSafeQueue osztálysablon

```
template <typename Item >
void ThreadSafeQueue::enqueue(const Item& e)
{
	std::unique_lock<std::mutex> lock(_mu);
	_queue.push(e);
	_cond.notify_one();
}

template <typename Item >
void ThreadSafeQueue::dequeue(Item& e)
{
	std::unique_lock<std::mutex> lock(_mu);
	while(empty() && _active){
		_cond.wait(lock);
	}
	if(_active){
		e = _queue.front();
		_queue.pop();
	}
}
```

az enqueue() a dequeue()-val kölcsönösen kizárásos módon hívható

a _cond objektummal blokkolt szálnak (csak egy van) engedélyezi a folytatást

a dequeue() a enqueue()-val kölcsönösen kizárásos módon hívható

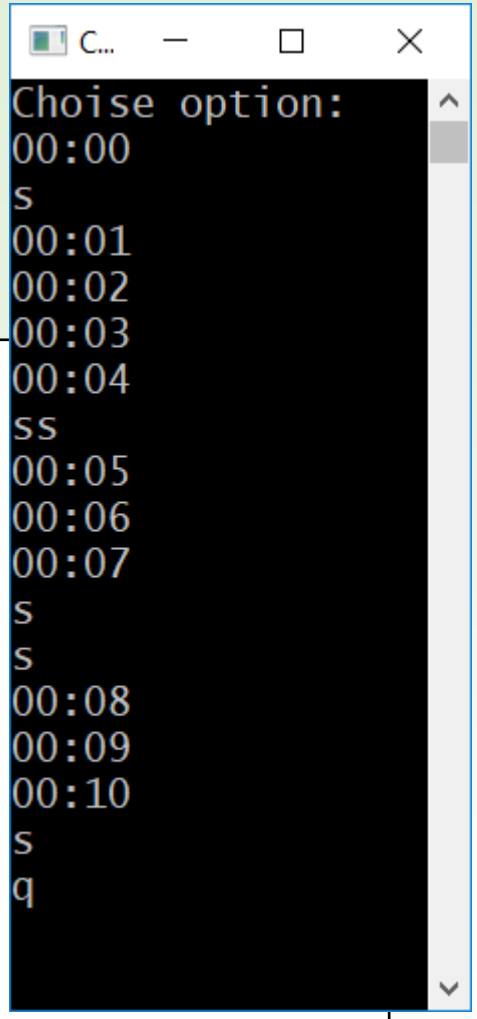
várakozik, amíg a sor üres és aktív

threadsafequeue.hpp

main() függvény

```
int main()
{
	Stopwatch stopwatch;
	std::cout << "Choise option:" << std::endl;
	char o;

	do{
		std::cin >> o;
		if(o == 's'){
			stopwatch.send(click);
		}
	}while(o != 'q');
	stopwatch.send(quit);

	return 0;
}
```

main.cpp



Továbbfejlesztés

- Mennyivel szebb lenne az alkalmazás, ha grafikusan is megjeleníthetnénk a stopper órát egy ablakban egy „nyomógombbal” együtt, amelyik a stopper órát elindítja, illetve megállítja.

- És menyivel könnyebb lenne az alkalmazás elkészítése, ha nem kellene mindent a nulláról felépíteni, hanem az ilyen alkalmazásokban egyformán jelen lévő általános elemeket készen kapva csak be kellene építeni őket.

Eseményvezérelt alkalmazások készítését támogató fejlesztő eszközök elemei

általános

- tipikus megjelenéssel és szignálküldési szokással rendelkező objektumok
  (ablak, lcd kijelző, időzítő)
- eseménykezelő mechanizmus
  - szignálok aszinkron küldése,
    (küldő és fogadó külön szálon)
  - szignálok eseménysorának biztonságos kezelése, szignálok célba juttatása

egyedi

- alkalmazás objektumainak létrehozása, ehhez
  - egyedi osztályok (származtatása)
  - a felhasználói felületen való megjelenésük terve
- eseménykezelő függvények elkészítése
- eseménykezelő függvények hozzárendelése szignálokhoz

Stopwatch felhasználói felülete

- A Stopwatch osztályt úgy tervezzük meg, hogy a példányosításakor ablakként jelenjen meg a képernyőn.
- A Stopwatch osztály bizonyos adattagjainak osztályai (LCD kijelző, nyomógomb, időzítő) egy meglevő készlethez tartoznak. Ezek közül néhány meg is jelenik majd a Stopwatch ablakában.

A stopper egy ablakszerű objektum, amely bezárása kiváltja a quit szignált. Tartalmaz egy LCD kijelzőt, egy nem látható, tick szignálokat küldő időzítőt, továbbá egy nyomógombot, amellyel megnyomásával click szignálok küldhetők.

Stopwatch eseménykezelői

Meghatározzuk, hogy az alkalmazás során bekövetkező események által kiváltott szignálokhoz milyen eseménykezelést rendeljünk.

```
switch (currentState) {
    case stopped:
        switch (signal)
            case click: currentState = oper
            case tick:
            case quit:  timer.stop()
        endswitch
    case operate:
        switch (signal)
            case click: currentState = stop
            case tick:   lcd.display(++secon
            case quit: timer.stop()
        endswitch
endswitch
```

```
switch (signal) {
    case click:
        switch (curr
            case stop
            case ope
        endswitch
    case tick:
        switch (curr
            case stop
            case oper
        endswitch
    case quit:
        timer.stop()
endswitch
```

click:

```
switch (currentState) {
    case stopped: currentState = operate
    case operate: currentState = stopped
endswitch
```

tick:

```
switch (currentState) {
    case stopped:
    case operate: lcd.display(++seconds)
endswitch
```

quit:

```
timer.stop()
```

Igények

- Olyan fejlesztő környezetet keresünk, ahol
  - a grafikus megjelenéssel rendelkező objektumok már meglévő osztályok példányai lehetnének biztosítva annak a lehetőségét, hogy egyedi tulajdonságaikat is megadhassuk (OOP technikák).
  - nem kellene olyan mechanizmusokat újra és újra megvalósítani, mint az aszinkron szignálkezelés, vagy az időzítő.
  - elegendő lenne a feladat egyedi elemeit leprogramozni, mint például a konkrét szignálokra kiváltott esemény-kezelés.
  - a grafikus elrendezést egy vizuális tervezővel alakíthatnánk ki.

Stopwatch Qt-val fejlesztve

Stopwatch : public QWidget
Ablakszerű vezérlő objektum, amely más vezérlőket (időzítő, kijelző, nyomógomb) tartalmaz.
Lezárása váltja ki a quit szignált.

QLCDNumber display metódussal

QPushButton típusú objektum váltja ki a click szignált

```
#include <QApplication>
#include "stopwatch.h"
int main(int argc, char *argv[])
{
    QApplication app(argc,argv);
    Stopwatch *stopwatch = new Stopwatch;
    stopwatch ->show();
    return app.exec();
}
```

main.cpp

QApplication
Többek között gondoskodik arról, hogy az események a megfelelő vezérlőkhöz jussanak el, hogy ott szignálokat váltsanak ki.

Stopwatch osztály, mint QWidget

```
#include <QWidget>
class QTimer;
class QLCDNumber;
class QPushButton;
enum State {stopped, operate};
class Stopwatch: public Qwidget
{
        Q_OBJECT
    public:
        Stopwatch(QWidget *parent=0);
    protected:
        void closeEvent(QCloseEvent * event) { _timer->stop(); }
    private:
        QTimer      *_timer;
        QLCDNumber  *_lcd;
        QPushButton *_button;
        State _currentState;
        int   _seconds;
        QString Stopwatch::format(int n) const;
    private slots:
        void oneSecondPass(); // tick
        void buttonPressed();  // click
};
```

a kilépéskor generált (quit) szignál eseménykezelője

QTimer típusú objektum váltja ki a tick szignált

most nem az LcdNumber metódusa

szignálok eseménykezelői

stopwatch.h

Qt-s Stopwatch eseménykezelői

tick szignál eseménykezelője
a korábbi natív C++ kód
transition() metódusának része

```
Stopwatch::oneSecondPass() {
        switch (_currentState){
                case operate: _lcd->display(format(++_seconds)); break;
                case stopped: break;
        }
}
```

click szignál eseménykezelője
a korábbi natív C++ kód
transition() metódusának része

```
Stopwatch::buttonPressed() {
        switch (_currentState){
                case operate: _currentState = stopped; break;
                case stopped: _currentState = operate; break;
        }
}
```

stopwatch.cpp

Qt-s Stopwatch konstruktora

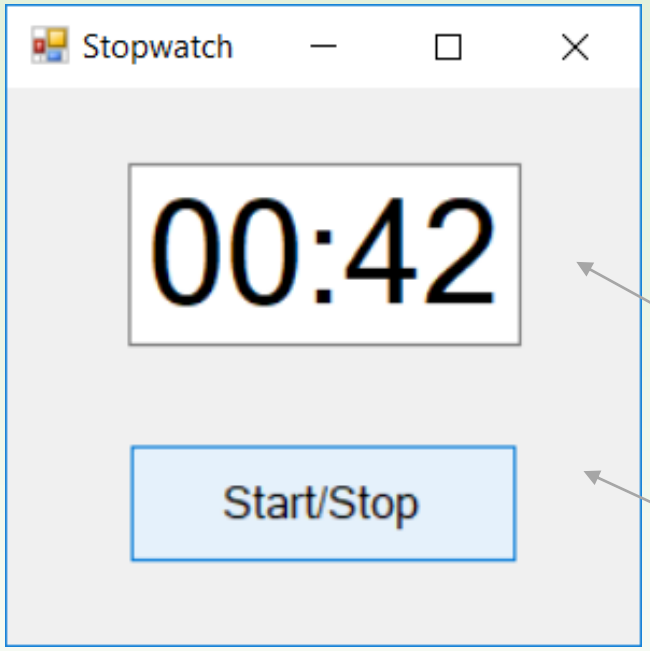
```
Stopwatch::Stopwatch(QWidget *parent) : Qwidget(parent)
{
        setWindowTitle(tr("Stopwatch"));
        resize(150, 60);

        _timer  = new QTimer;
        _lcd    = new QLCDNumber;
        _button = new QPushButton("Start/Stop");

        …

        connect(_timer,  SIGNAL(timeout()), this, SLOT(oneSecondPass()));
        connect(_button, SIGNAL(clicked()), this, SLOT(buttonPressed()));

        _currentState = stopped;
        _seconds = 0;
        _lcd->display(_seconds);
        _timer->start(1000);
}
```

itt kerül sor a grafikus vezérlőknek
az ablakban való elrendezésére
és egyéb tulajdonságainak megadására.
Ez automatikusan is generálható,
ha vizuális tervezőt (QtDesigner) használunk

szignálok és kezelőik egymáshoz rendelése:
tick (timeout()) ~ oneSecondPass()
click (clicked()) ~ buttonPressed()

stopwatch.cpp

Stopwatch .net alatt fejlesztve



Stopwatch : Form
Ablakszerű vezérlő objektum, amely más vezérlőket (időzítő, kijelző, nyomógomb) tartalmaz.
Lezárása váltja ki a quit szignált.

TextBox display metódussal

Button típusú objektum váltja ki a click szignált

Application
Többek között gondoskodik arról, hogy az események a megfelelő vezérlőkhöz jussanak el, hogy ott szignálokat váltsanak ki.

```
static class Program
{
    [STAThread]
    static void Main(){
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        Application.Run(new Stopwatch());
    }
}
```

program.cs

Stopwatch osztály, mint .net Form

```
public partial class Stopwatch : Form
{
    enum State { stopped, operate };
    State _currentState;
    DateTime _seconds = new DateTime(0);

    private System.Windows.Forms.Timer   _timer;
    private System.Windows.Forms.Button  _button;
    private System.Windows.Forms.TextBox _lcd;

    public Stopwatch(){ … }

    private void timer_Tick(object sender, EventArgs e) { … }
    private void button_Click(object sender, EventArgs e) { … }
    private void MainForm_FormClosed(object sender,FormClosedEventArgs e) { … }

    private void display()
    {
        _lcd.Text = string.Format("{0}:{1}",
            seconds.Minute.ToString().PadLeft(2, '0'),
            seconds.Second.ToString().PadLeft(2, '0'));
    }
}
```

a kód egy része itt is generálható automatikusan vizuális tervezéssel

A display() itt sem az lcd kijelző metódusa, hanem a stopperé

Stopwatch.cs

.net-es Stopwatch eseménykezelői

```
private void timer_Tick(object sender, EventArgs e)
{
    switch (currentState){
        case State.operate:
            seconds = seconds.AddSeconds(1);
            display();
            break;
        case State.stopped: break;
    }
}
private void button_Click(object sender, EventArgs e)
{
    switch (currentState){
        case State.operate:
            currentState = State.stopped;
            break;
        case State.stopped:
            currentState = State.operate;
            break;
    }
}
private void MainForm_FormClosed(object sender,FormClosedEventArgs e)
{
    _timer.Stop();
}
```

tick szignál eseménykezelője
a transition() metódus része

click szignál eseménykezelője
a transition() metódus része

quit szignál eseménykezelője
a transition() metódus része

stopwatch.cs

.net-es Stopwatch konstruktora

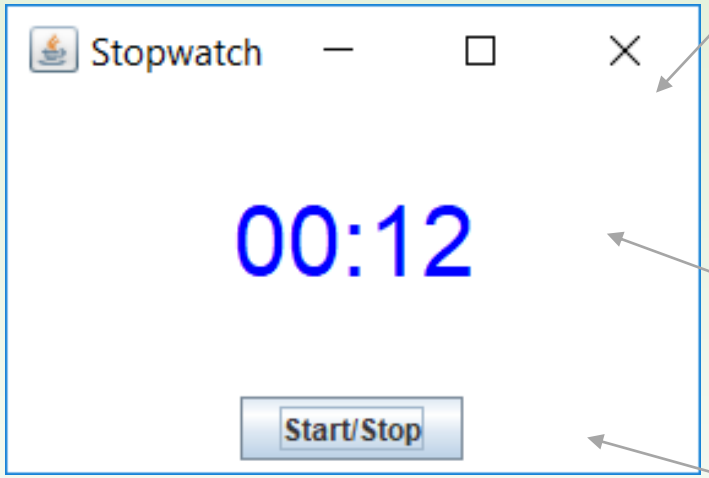
```
public Stopwatch(){
    this.components = new System.ComponentModel.Container();
    this.button = new System.Windows.Forms.Button();
    this.lcd = new System.Windows.Forms.TextBox();
    this.timer = new System.Windows.Forms.Timer(this.components);
    …
    this.Text = "Stopwatch";
    this.button.Text = "Start/Stop";
    this.lcd.Text = "00:00";
    this.timer.Interval = 1000;
    …
    this.Controls.Add(this.lcd);
    this.Controls.Add(this.button);
    …
    this._timer.Tick    += new System.EventHandler(this.timer_Tick);
    this._button.Click += new System.EventHandler(this.button_Click);
    this.FormClosed   += new System.Windows.Forms.
                              FormClosedEventHandler(MainForm_FormClosed);

    _currentState = State.stopped;
    display();
    _timer.Start();
}
```

vezérlők elrendezésének és egyéb tulajdonságainak megadása vizuális tervezéssel is végezhető, és a kód automatikusan generálható

szignálok és kezelőik egymáshoz rendelése

Stopwatch.cs

Stopwatch Java-ban



Stopwatch extends JFrame
Ablakszerű vezérlő objektum, amely más vezérlőket (időzítő, kijelző, nyomógomb) tartalmaz.
Lezárása váltja ki a quit szignált.

LCDNumber display metódussal

JButton típusú objektum váltja ki a click szignált

```
public class Stopwatch extends JFrame
{
      ...
   public static void main(String[] args) {
      new Stopwatch();
   }
}
```

Stopwatch.java

Stopwatch osztály, mint JFrame

```
public class Stopwatch extends JFrame
{
        enum State { operate, stopped }
        private State currentState;
        private int seconds = 0;

        private final static int SECOND = 1000 /* milliseconds */;
        private Timer timer = new Timer(SECOND, null);
        private LcdNumber lcd = new LcdNumber("00:00");
        private JButton button = new JButton("Start/Stop");
        private JPanel buttonPanel = new JPanel();

        public Stopwatch() { … }

        void click() { … }

        void tick() { … }

        protected void finalize() throws Throwable { … }

        public static void main(String[] args) {
                new Stopwatch();
        }
}
```

Stopwatch.java

Java-s LCD kijelző osztály

```
public class LcdNumber extends JLabel {
        public LcdNumber(String text) {
                super(text);
                setHorizontalAlignment(JLabel.CENTER);
                setOpaque(true);
                setBackground(Color.WHITE);
                setForeground(Color.BLUE);
                setFont(new Font(Font.DIALOG, Font.PLAIN, 40));
        }

        public void display(int seconds) {
                setText(String.format("%02d:%02d",
                                (seconds % 3600) / 60,      // minutes
                                (seconds % 3600) % 60));  // seconds
        }
}
```

a formázás az lcd kijelző része

LcdNumber.java

Java-s Stopwatch eseménykezelői

```
void click() {
    switch (currentState)  {
        case operate :
            currentState = State.stopped;
            break;
        case stopped :
            currentState = State.operate;
            break;
    }
}

void tick() {
    switch (currentState)  {
        case operate :
            ++seconds;
            lcd.display(seconds);
            break;
        case stopped :   break;
    }
}

protected void finalize() throws Throwable {
    if (timer.isRunning()) timer.stop();
    super.finalize();
}
```

click szignál eseménykezelője a transition() metódus része

tick szignál eseménykezelője a transition() metódus része

quick szignál maga után vonja az időzítő leállítását

Stopwatch.java

Java-s Stopwatch konstruktora

```
public Stopwatch() {
        super("Stopwatch");
        setBounds(250, 250, 300, 200);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        buttonPanel.setBackground(Color.WHITE);
        buttonPanel.add(button);
        add(lcd);
        add(buttonPanel, "South");

        button.addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e){ click(); }
        });

        timer.addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e){ tick(); }
        });
        currentState = State.stopped;
        lcd.display(seconds);
        timer.start();
        setVisible(true);
}
```

quit

események és kezelésük egymáshoz rendelése

Stopwatch.java