Eseménykezelés

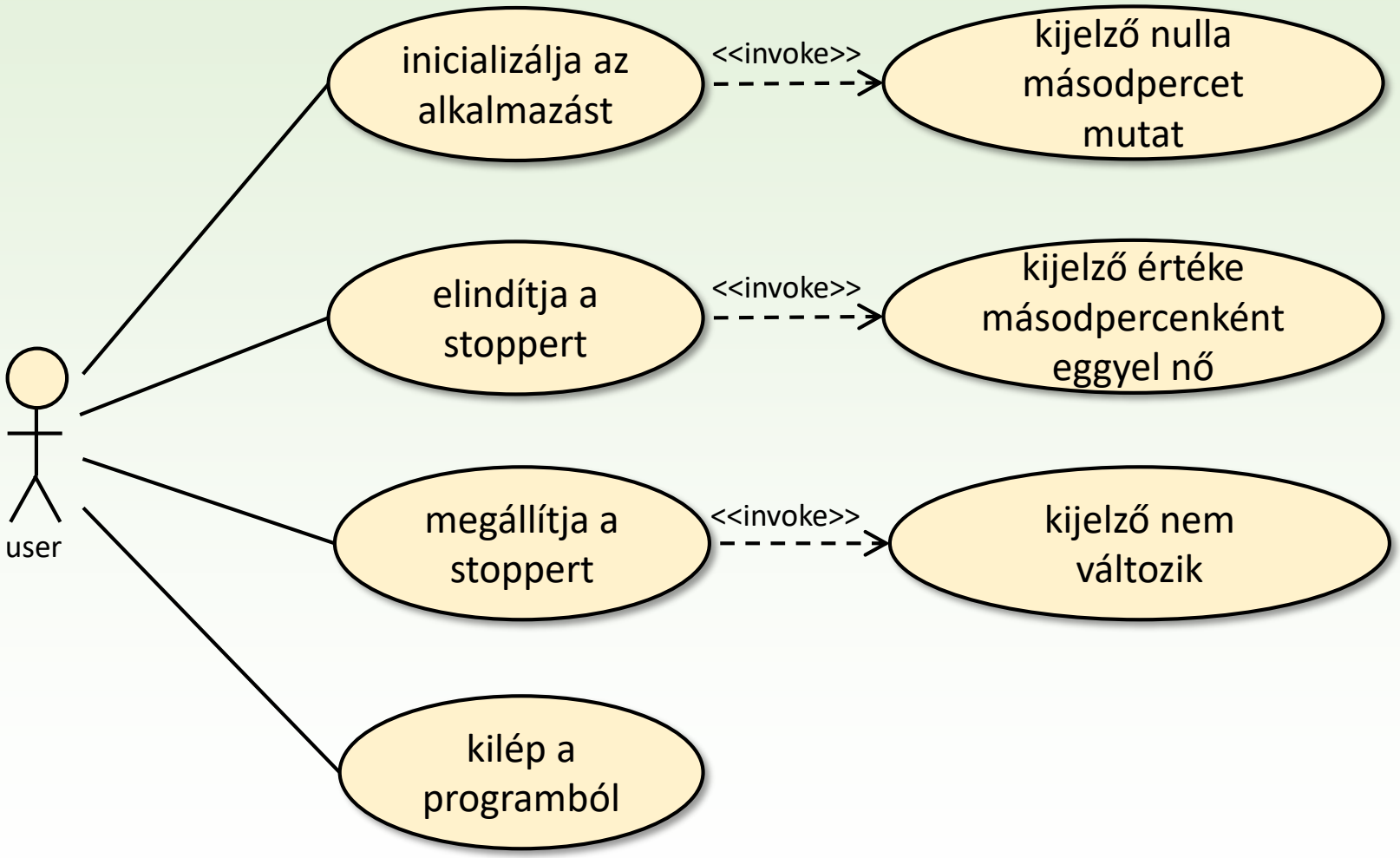
Aszinkron kommunikáció

Feladat

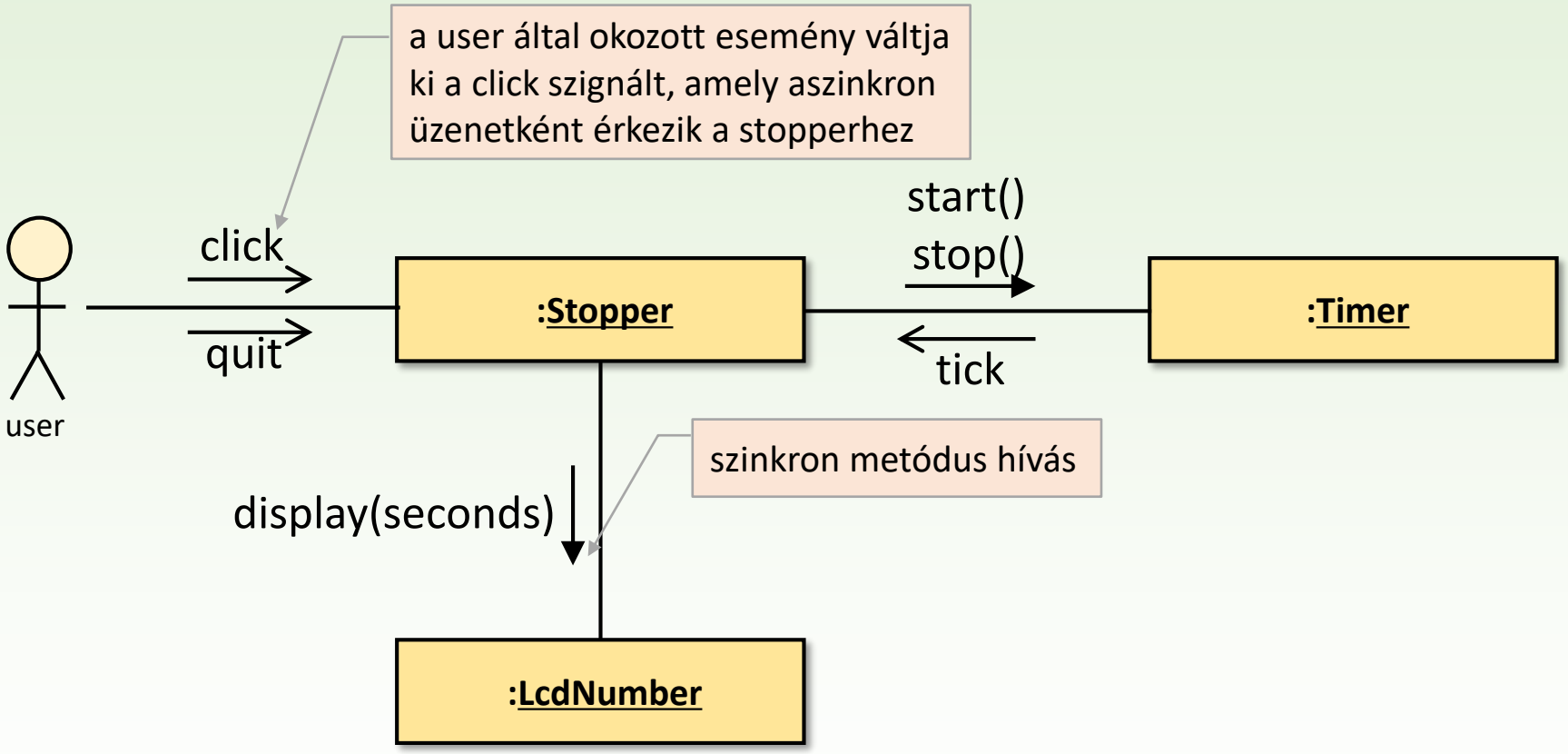
Készítsünk egy stoppert, amely másodpercenként jelzi a múló időt. Ez a folyamat

- egy adott jelzés hatására induljon el;
- ugyanezen jelzés ismétléseinek hatására váltakozva szüneteljen illetve folytatódjon tovább;
- egy adott másik jelzés hatására álljon le.

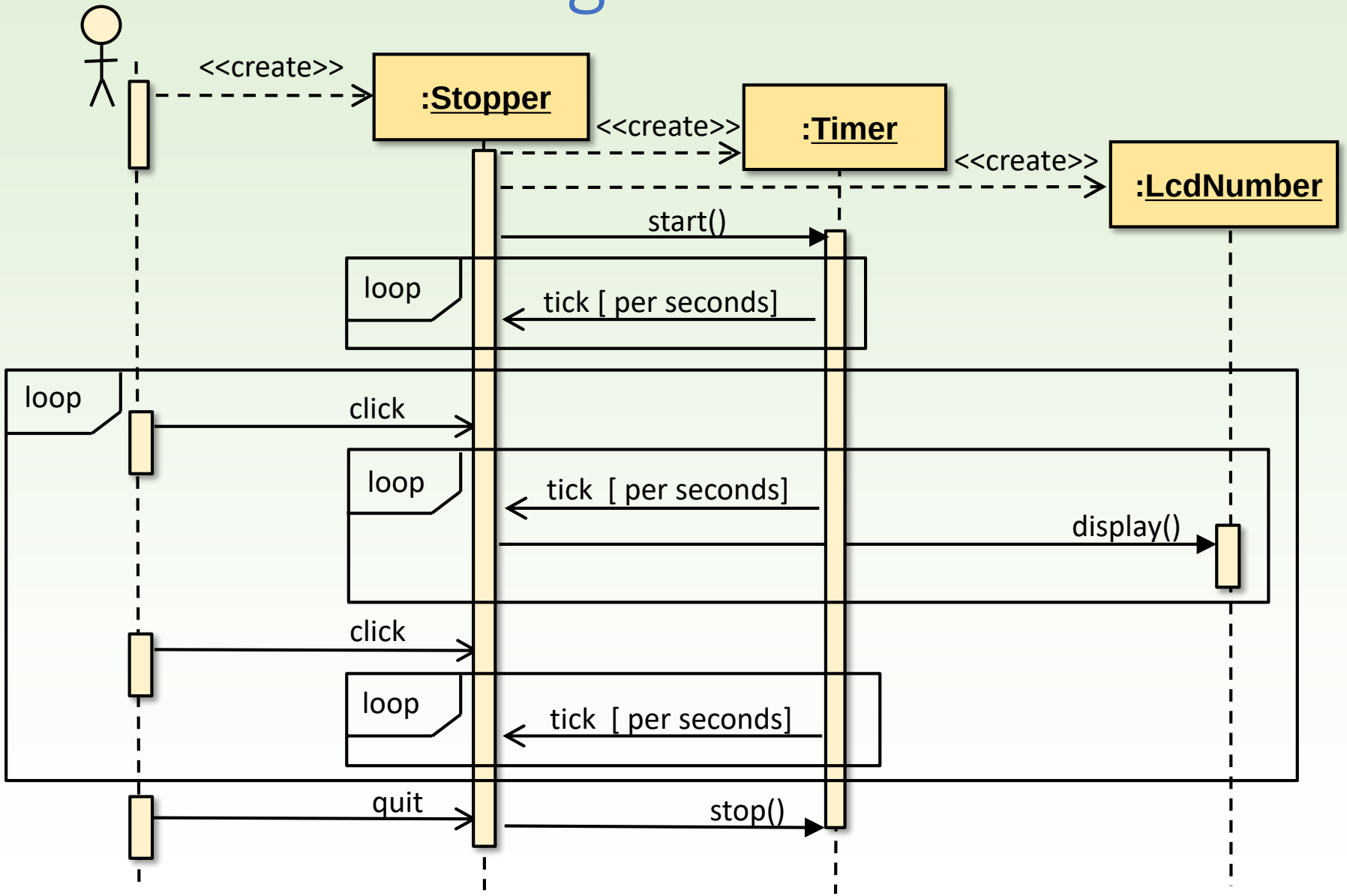
Használati eset diagram



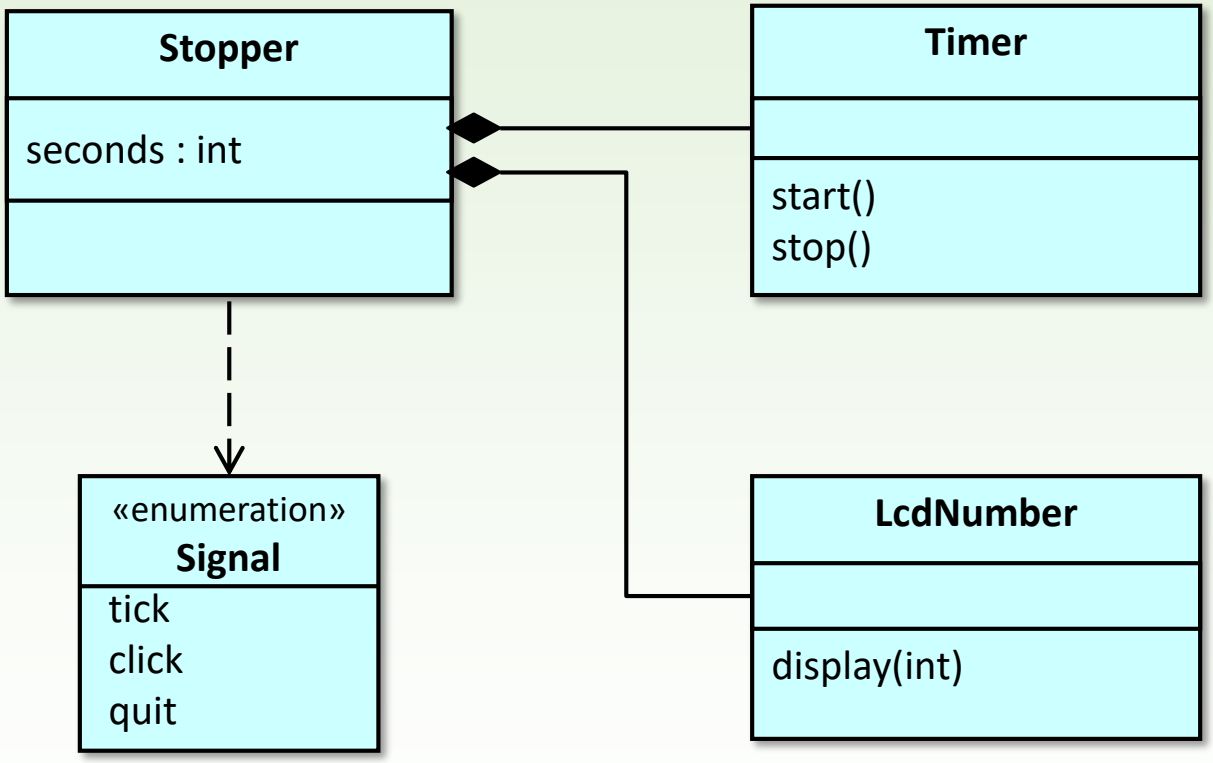
Kommunikációs diagram



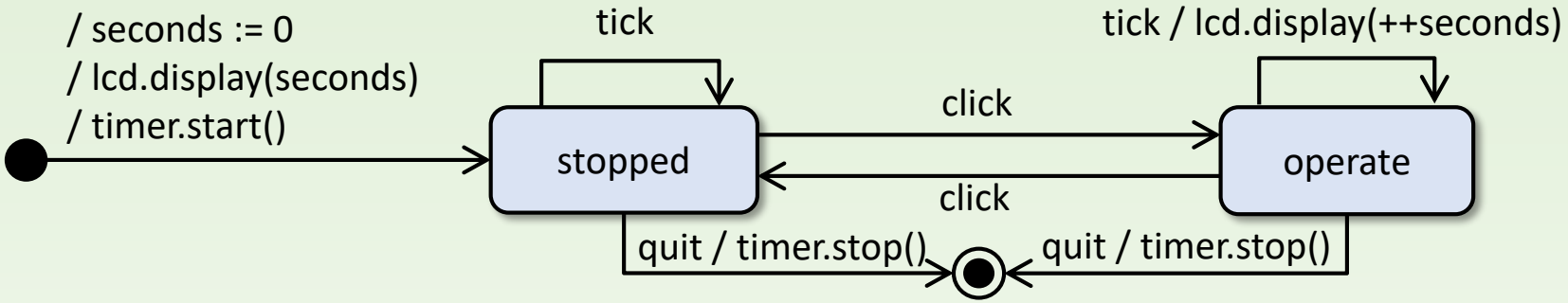
Szekvencia diagram



Osztálydiagram: elemzés



Stopper állapotgép diagramja



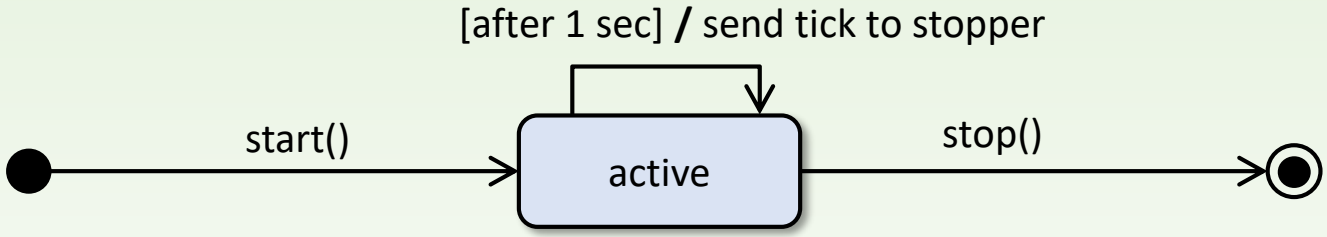
stateMachine()

```
seconds := 0
lcd.display(seconds) timer.start()
currentState := stopped
while  (currentState='stopped'
          or currentState='operate' )
   transition(getSignal())
endwhile
```

transition(signal : Signal)

```
switch (currentState) {
      case stopped:
         switch (signal)
            case click: currentState = operate
            case tick:
            case quit:  active := false; timer.stop()
         endswitch
       case operate:
         switch (signal)
            case click: currentState = stopped
            case tick:   lcd.display(++seconds)
            case quit:  active := false; timer.stop()
         endswitch
endswitch
```

Timer állapotgép diagramja



start()

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

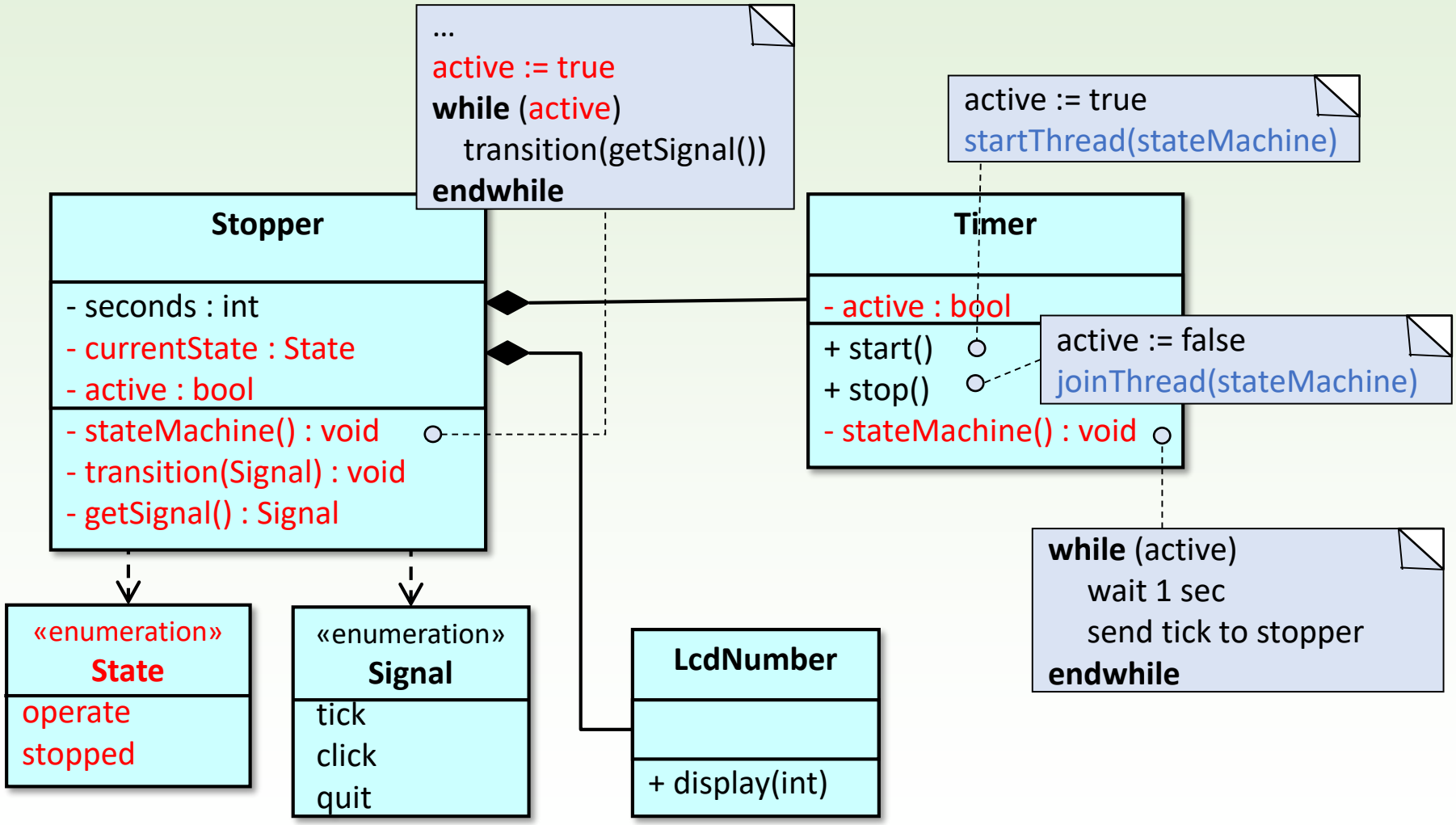
stateMachine()

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

stop()

```
active := false
```

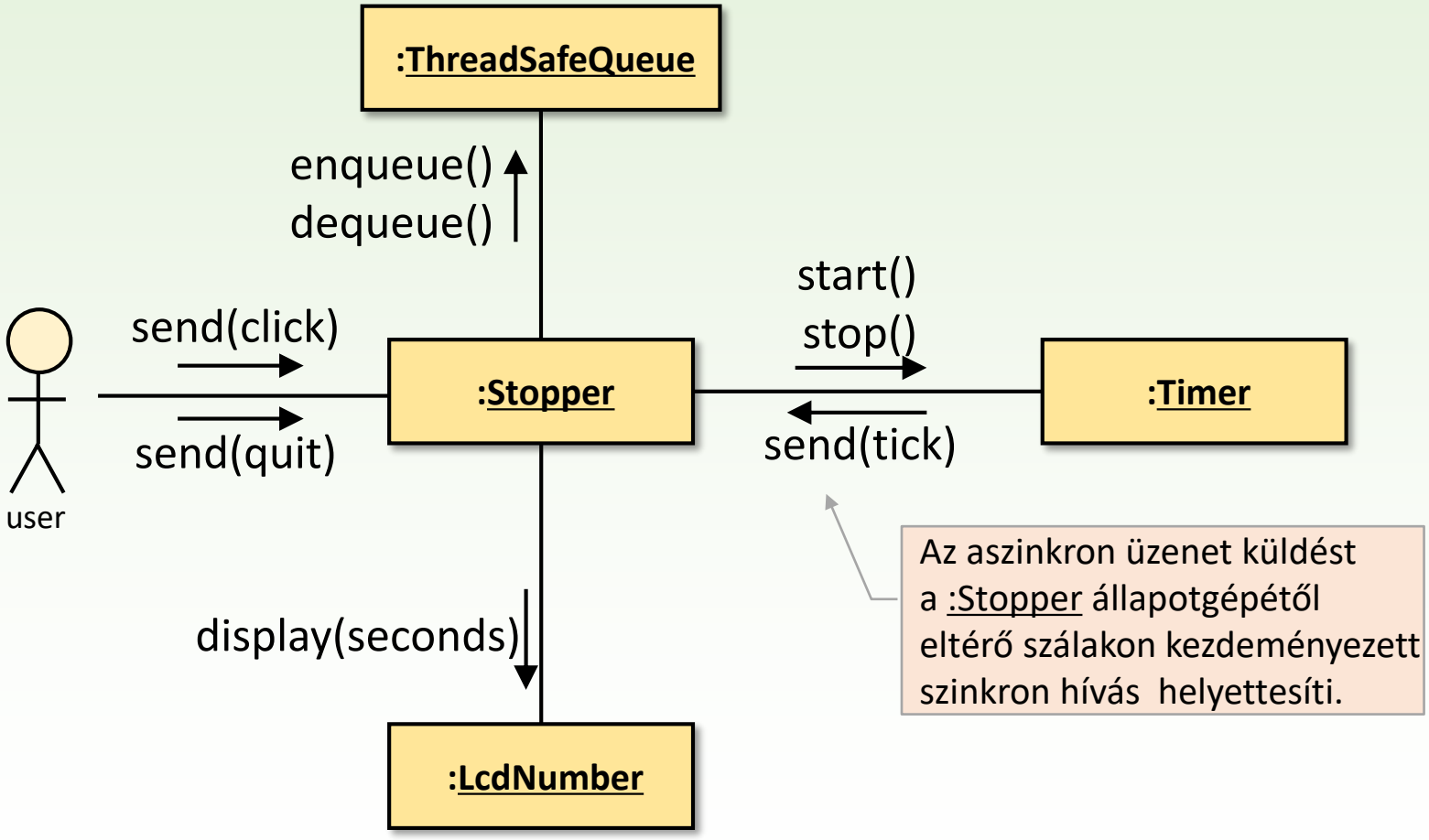
Osztálydiagram: tervezés



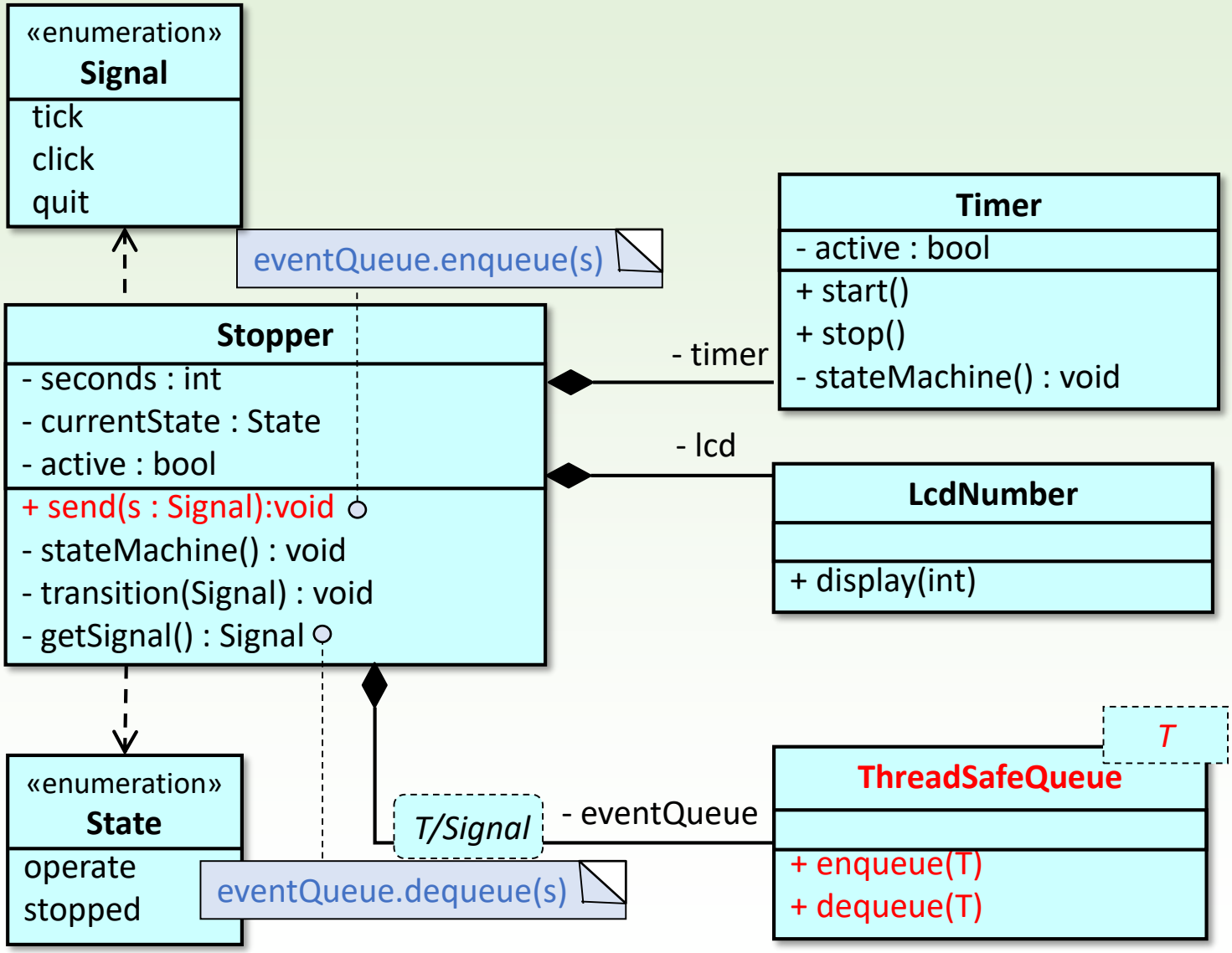
Megvalósítás

- Többszálú alkalmazásra van szükség, mert több aktív objektum van.
  - Külön szálon kell futnia a stopper és a timer állapotgépének.
- A stopper objektum a több irányból érkező aszinkron üzeneteket (a szignálokat) egy eseménysorba gyűjti.
  - Az eseménysorba a stopper objektum által biztosított send() metódus tesz majd be egy szignált. Ezt a metódust a küldő objektum a saját szálán hívja meg.
  - Az eseménysorból a szignálokat a stopper objektum saját szálán futó állapotgépe (getSignal()) veszi ki.
  - 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: megvalósítás



Osztálydiagram: megvalósítás



Stopper osztály

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

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

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

     State _currentState;
     int _seconds;
     bool _active;
     std::thread _processorThread;
};
```

```
enum Signal {tick, click, quit};
```

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

stopper.h

Stopper osztály

külön szálon indul el a Stopper állapotgépe

bevárja amíg az állapotgép leáll

```
Stopper::Stopper():_timer(this),
     _processorThread(&Stopper::stateMachine, this)
{
     _eventQueue.startQueue();
}
Stopper::~Stopper()
{
     _processorThread.join();
     _eventQueue.stopQueue();
}
```

```
void Stopper::stateMachine()
{
     _seconds = 0;
     _lcd.display(_seconds);
     _timer.start();
     _currentState = stopped;
     _active = true;
     while(_active) { //amíg nincs terminálás
          if(_active) transition(getSignal());
     }
     _timer.stop();
}
Signal getSignal()
{
     Signal s;
     _eventQueue.dequeue(s);
     return s;
}
```

stopper.cpp

Stopper eseménykezelője

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

stopper.cpp

Timer osztály

A „timer.h”-t már inklúdolja a „stopper.h”.
Itt viszont szükség lenne a Stopper-re.

```
class Stopper;

class Timer{
    typedef std::chrono::milliseconds milliseconds;
public:
    Timer(Stopper *t) : _target(t), _active(false) {}

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

    Stopper *_target;

    bool _active;
    std::thread _processorThread;
};
```

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

timer.h

Timer osztály

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

void Timer::stop() {
     _active = false;
     _processorThread->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));
          _target->send(tick);
     }
}
```

külön szálon indul el a Timer állapotgépe

feltételes változó és szemafor
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 T>
class ThreadSafeQueue
{
public:
    ThreadSafeQueue() { _active = false; }

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

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

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

a _cond objektummal blokkolt összes szálnak (itt a dequeue()-t használó stopper állapotgépének) engedélyezi a folytatást

feltételes változó és szemafor kölcsönösen kizárás megvalósításához
#include <condition_variable>
#include <mutex>

threadsafequeue.hpp

ThreadSafeQueue osztálysablon

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

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

az enqueue() és a dequeue() kölcsönösen kizárásos módon hívható

a _cond objektummal blokkolt szálak közül egynek engedélyezi a folytatást

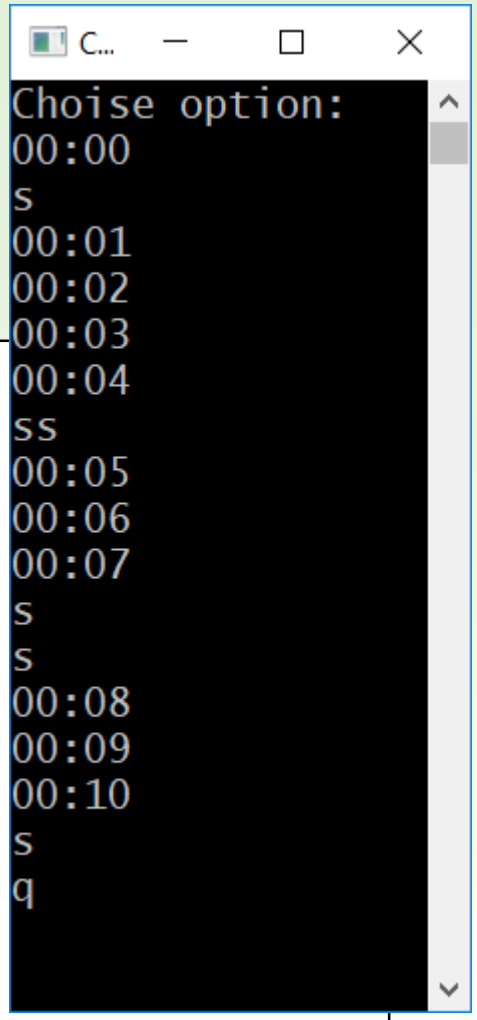
várakozik, amíg a sor üres és aktív

threadsafequeue.hpp

main() függvény

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

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

	return 0;
}
```

main.cpp



Eseményvezérelt alkalmazás egyedi és általános elemei

egyedi

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

általános

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

Stopper felhasználói felülete

A felhasználói felületet sokszor egy vizuális tervezővel rajzoljuk meg, amellyel

- létrehozhatjuk az egyedi ablakként megjelenő Stopper osztályt
- definiáljuk és példányosítjuk a stopper komponenseit (lcd kijelző, timer)
- elrendezzük az ablakon a felületen is megjelenő komponenseket

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 click és quit szignálok küldhetők.

Stopper eseménykezelése

Az alkalmazás során bekövetkező események által kiváltott szignálokhoz

- hozzá kell rendelni eseményeket kezelő metódusokat
- amelyeket implementálni kell

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

```
switch (signal) {
    case
        sv
        e
    case
        sv
        e
    case
        ti
endswitch
```

click:

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

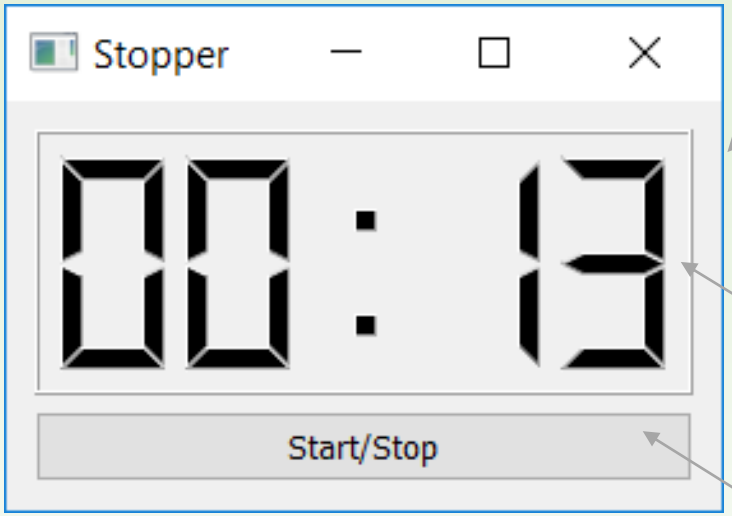
tick:

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

quit:

```
timer.stop()
```

Stopper Qt-val fejlesztve



Stopper : 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 "stopper.h"
int main(int argc, char *argv[])
{
    QApplication app(argc,argv);
    Stopper *stopper = new Stopper;
    stopper->show();
    return app.exec();
}
```

main.cpp

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

Stopper osztály, mint QWidget

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

a kód egy része automatikusan is generálható

a kilépéskor generált (quit) szignál eseménykezelője

QTimer típusú objektum váltja ki a tick szignált

többi szignál eseménykezelője

stopper.h

Stopper osztály Qt eseménykezelése

```
Stopper::Stopper(QWidget *parent) : Qwidget(parent)
{
    setWindowTitle(tr("Stopper"));
    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 történik a vezérlők elrendezésének és egyéb tulajdonságainak megadása

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

stopper.cpp

Stopper osztály Qt eseménykezelői

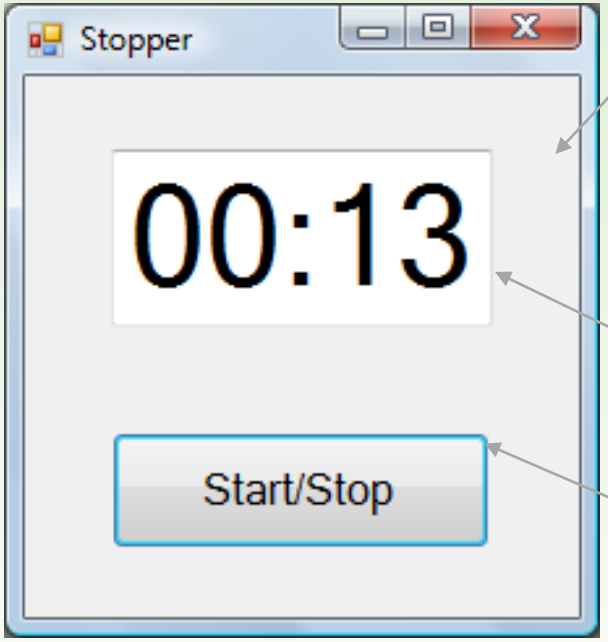
```
Stopper::oneSecondPass() {
    switch (_currentState){
        case operate: _lcd->display(format(++_seconds)); break;
        case stopped: break;
    }
}

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

QString Stopper::format(int n) const
{
    return extend((n % 3600) / 60) + ":" + extend((n % 3600) % 60);
}

QString Stopper::extend(int n) const
{
    return (n < 10 ? "0" : "") + QString::number(n);
}
```

tick szignál eseménykezelője

click szignál eseménykezelője

stopper.cpp

Stopper .net alatt fejlesztve



Stopper : 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

QApplication
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 Stopper());
    }
}
```

program.cs

Stopper osztály, mint .net Form

```
public partial class Stopper : 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 Stopper(){
        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 = "Stopper";
        this.button.Text = "Start/Stop";
        this.lcd.Text = "00:00";
        this.timer.Interval = 1000;
        …
        this.Controls.Add(this.lcd);
        this.Controls.Add(this.button);
        …
```

a kód egy része automatikusan is generálható

vezérlők elrendezésének és egyéb tulajdonságainak megadása

Stopper.cs

Stopper osztály .net eseménykezelése

```
        …
        this._timer.Tick   += new System.EventHandler(oneSecondPass);
        this._button.Click += new System.EventHandler(buttonPressed);
        this.FormClosed    += new System.Windows.Forms.
                           FormClosedEventHandler(stop);

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

    private void oneSecondPass(object sender, EventArgs e){ … }
    private void buttonPressed(object sender, EventArgs e){ … }
    private void stop(object sender,FormClosedEventArgs
    e){_timer.Stop();}

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

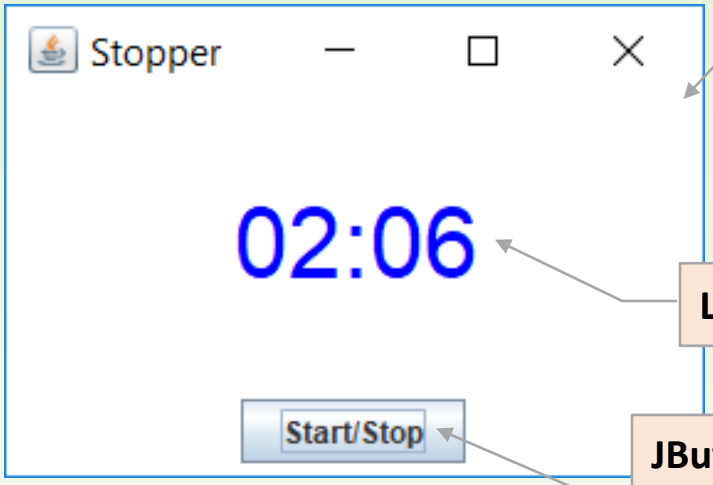
események és kezelésük egymáshoz rendelése

quit – FormedClosed

ez most nem az lcd kijelző metódusa, hanem a stopperé

Stopper.Designer.cs

Stopper Java-ban



Stopper 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 Stopper extends JFrame
{
    …
    public static void main(String[] args) {
        new Stopper();
    }
}
```

Stopper.java

Stopper Java-ban

```
public class Stopper extends JFrame
{
    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 Stopper() { … }

    void click() { … }

    void tick() { … }

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

Stopper.java

Stopper konstruktora Java-ban

```
public Stopper() {
    super("Stopper");
    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(); }
    });

    timer.start();
    setVisible(true);
}
```

button eseménykezelője

timer eseménykezelője

Stopper.java

Stopper eseménykezelői Java-ban

```
void click() {
    if (timer.isRunning()) {
        timer.stop();
    } else {
        timer.restart();
    }
}

void tick() {
    seconds++;
    lcd.display(String.format("%02d:%02d",
                              (seconds % 3600) / 60,    // minutes
                              (seconds % 3600) % 60)); // seconds
    if (seconds % 60 == 0) {      // round minutes
        Toolkit.getDefaultToolkit().beep();
    }
}
```

tick szignál eseménykezelője

click szignál eseménykezelője

Stopper.java

LCD kijelző Java-ban

```
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(String text) {
        setText(text);
    }
}
```

LcdNumber.java