Event handling

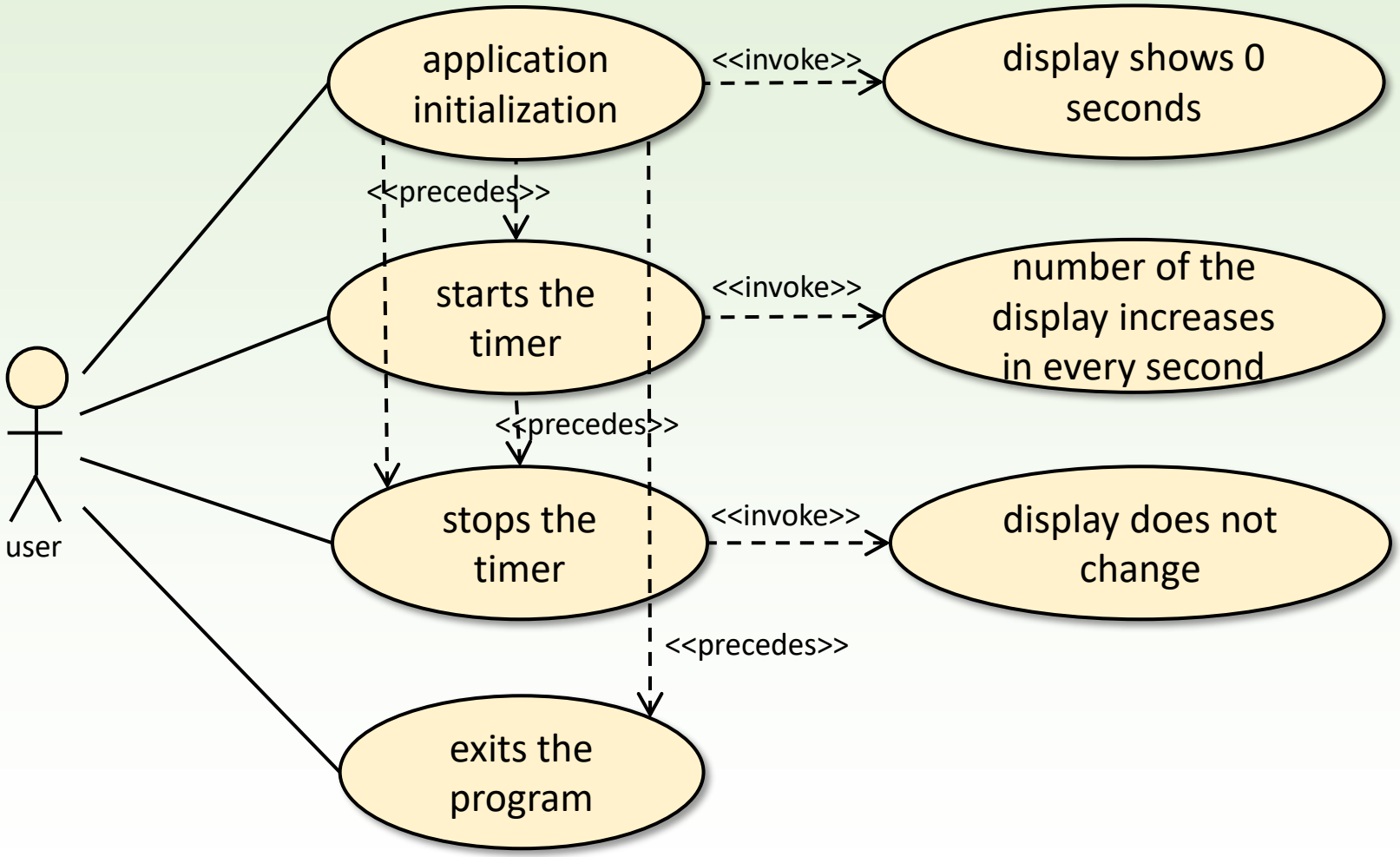
Asynchronous communication

Task

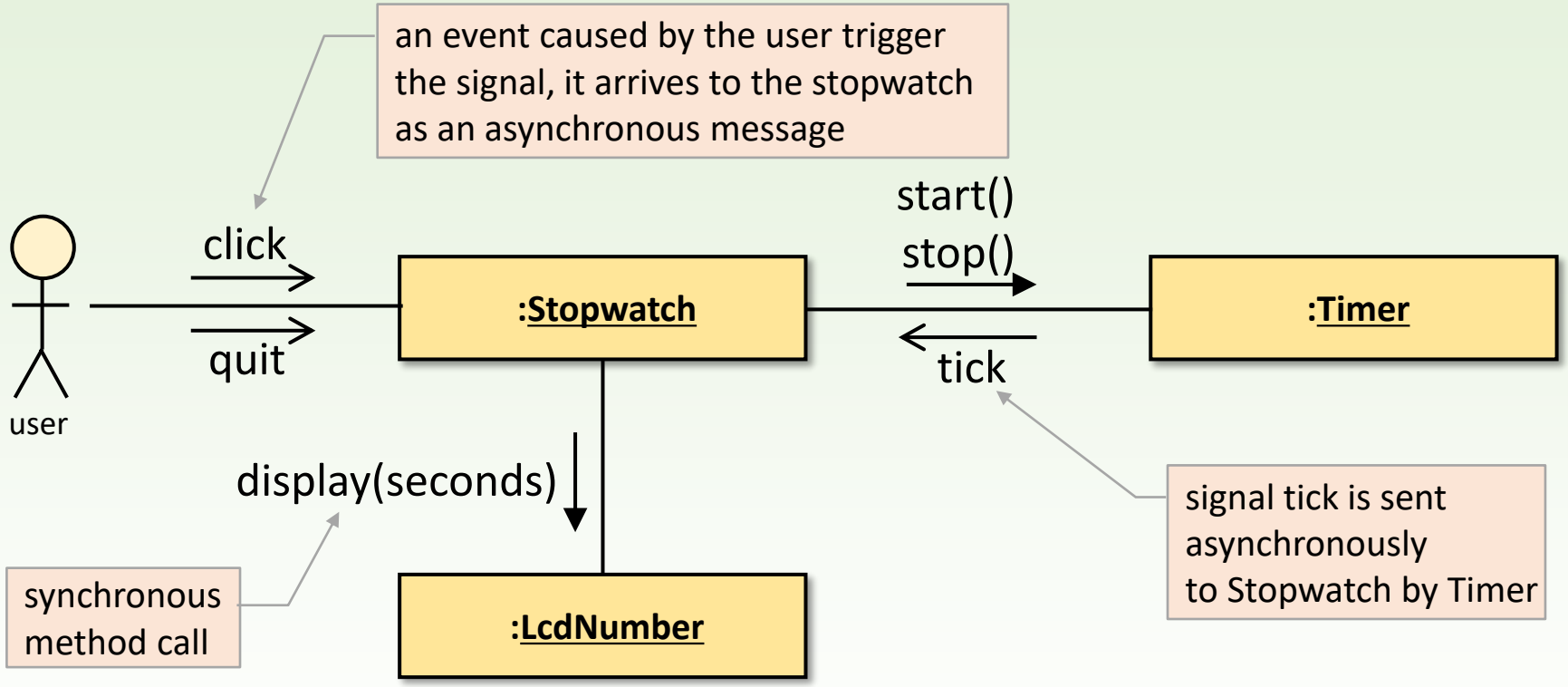
Create a timer which displays the time passed in every second. The process

- starts due to a signal,
- pauses and continues due to the same signal, and
- due to another signal, it stops.

Use case diagram

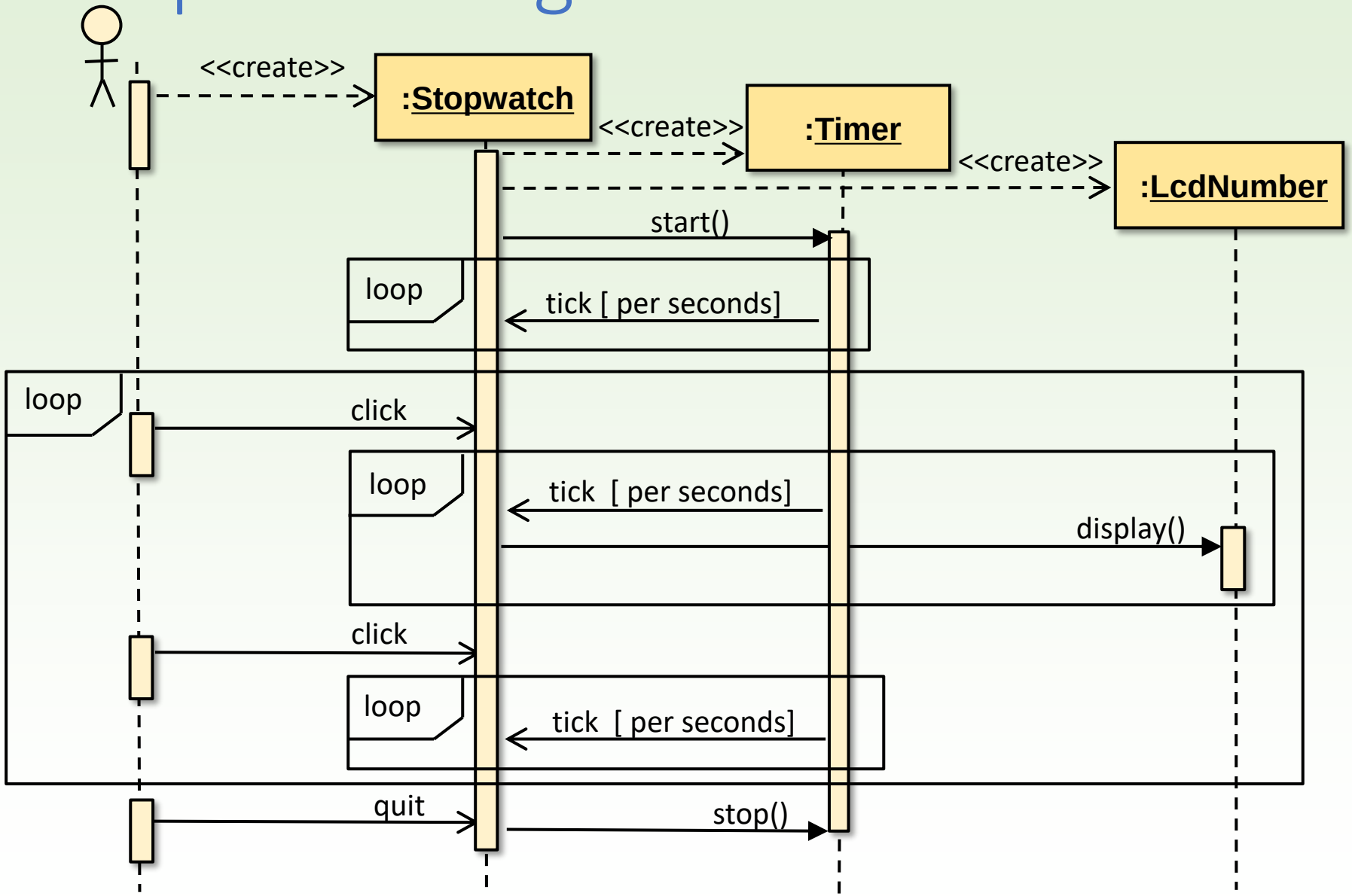


Communication diagram: planning

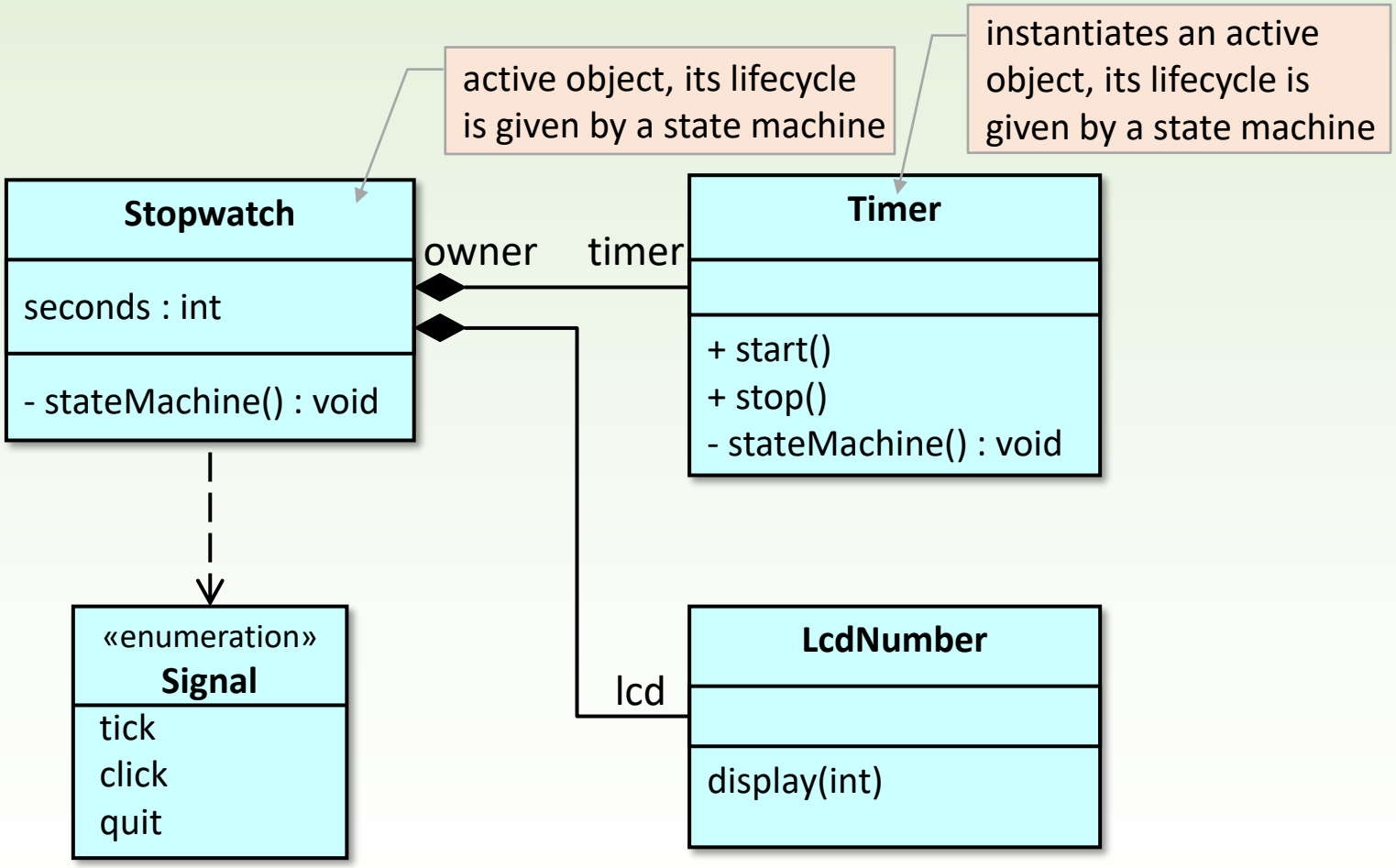


- Stopwatch and Timer are the so-called active objects.
- Stopwatch has to process the signals while the new ones arrive.
- Timer has to send a signal in every second no matter what.

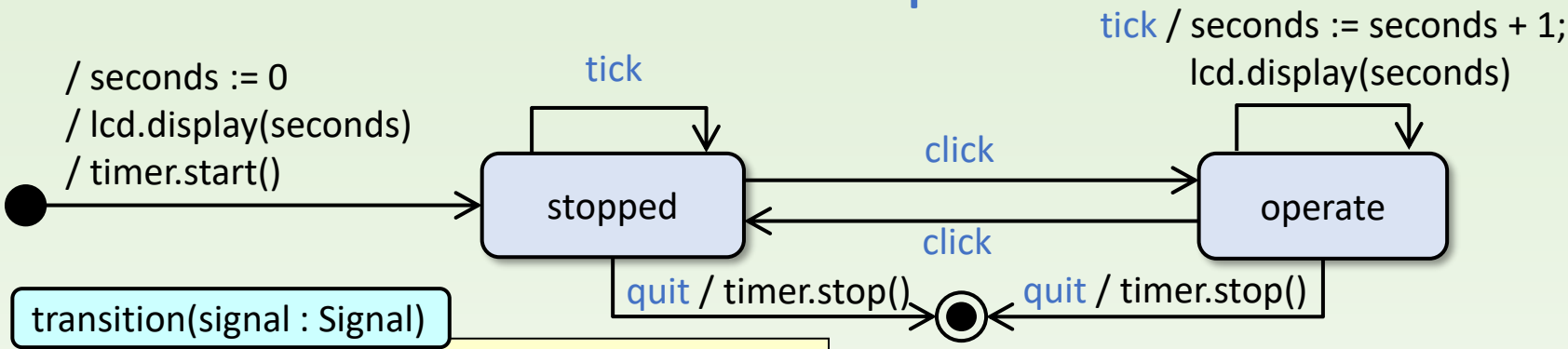
Sequence diagram



Class diagram: analysis



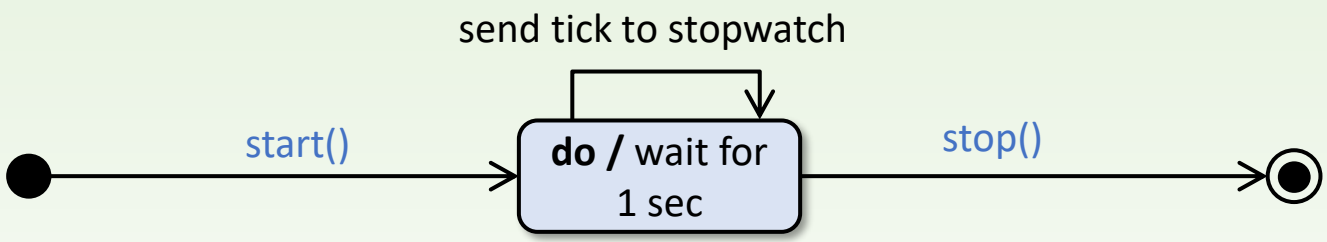
State machine of Stopwatch




```
transition(signal : Signal)
switch (currentState) {
      case stopped:
         switch (signal)
            case click: currentState := operate
            case tick:  skip
            case quit: active := false
         endswitch
      case operate:
         switch (signal)
            case click: currentState := stopped
            case tick:   seconds := seconds + 1
                         lcd.display(seconds)
            case quit:  active := false
         endswitch
endswitch
```

```
stateMachine()
seconds := 0
lcd.display(seconds)
timer.start()
currentState := stopped
active := true
while active loop
   transition(getSignal())
endloop
```

State machine of Timer

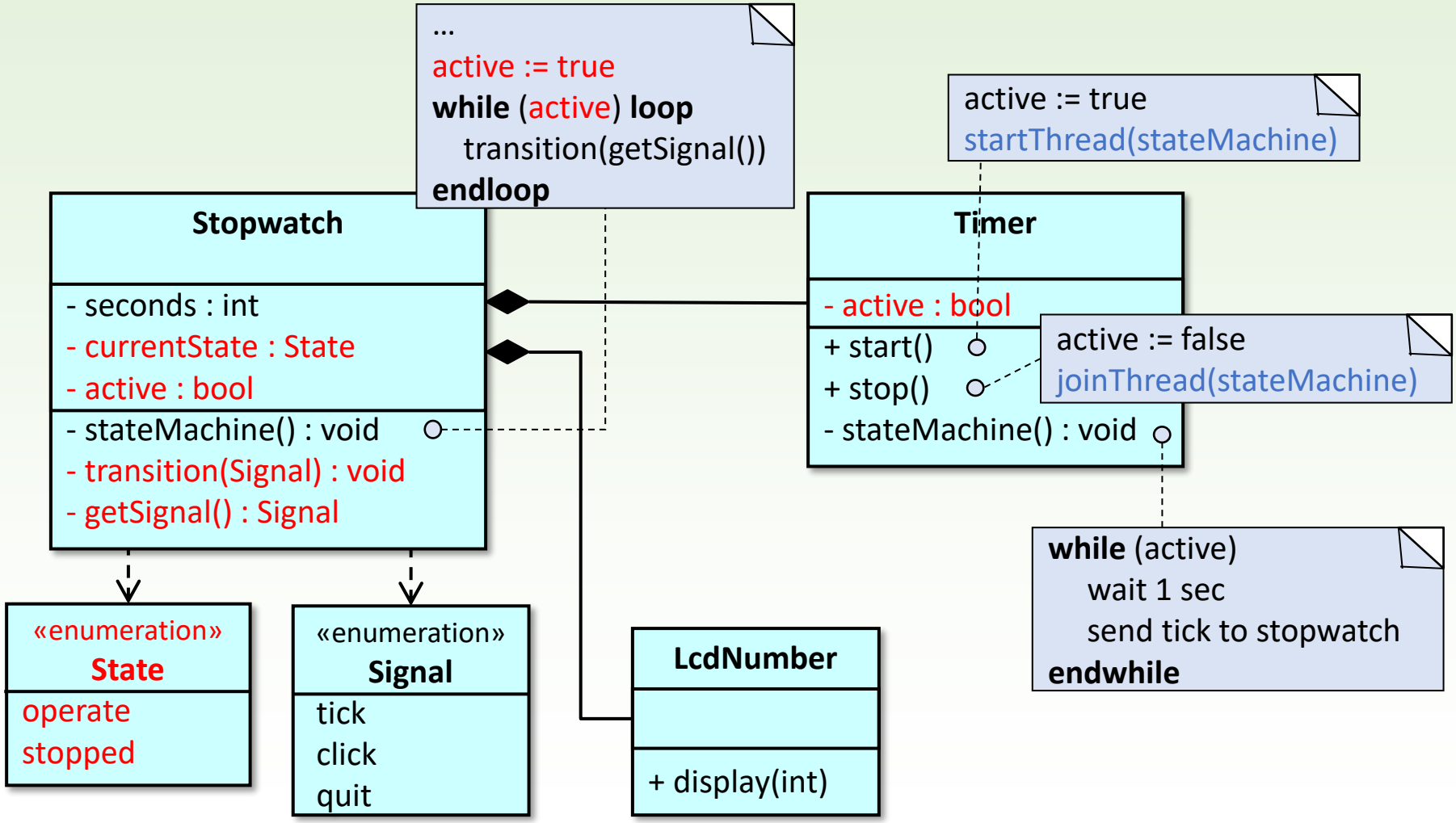



```
start()
    active := true
    stateMachine()
```

```
stateMachine()
    while  (active) loop
        wait 1 sec
        send tick to stopwatch
    endloop
```

```
stop()
    active := false
```

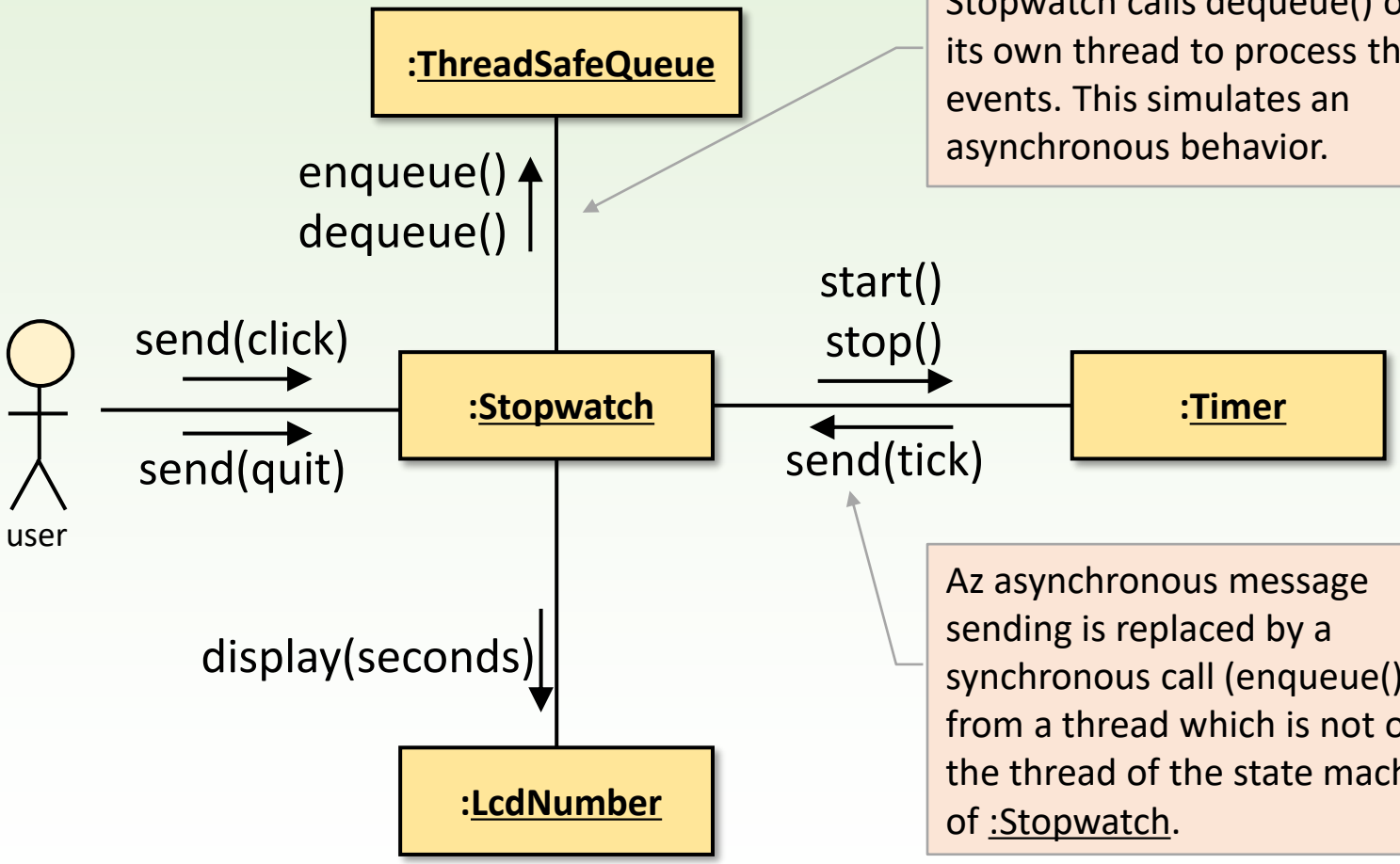
Class diagram: planning



Realization

- Multithread application is needed, as there are several active objects.
  - Different threads are needed for the state machines of stopwatch and timer.
- The stopwatch object collects the asynchronous messages (signals) arriving from different sources into one event queue.
  - Method send() belonging to object stopwatch pushes the signals into the event queue. This method is called on the own thread of the sender.
  - From the event queue, signals are got by the state machine of the stopwatch (getSignal()) which runs on its own thread.
  - Operations of the queue, send() and getSignal(), have to be synchronized: they have to work in a mutually exclusive way. In addition, in case of empty queue, getSignal() has to be blocked with a waiting command.

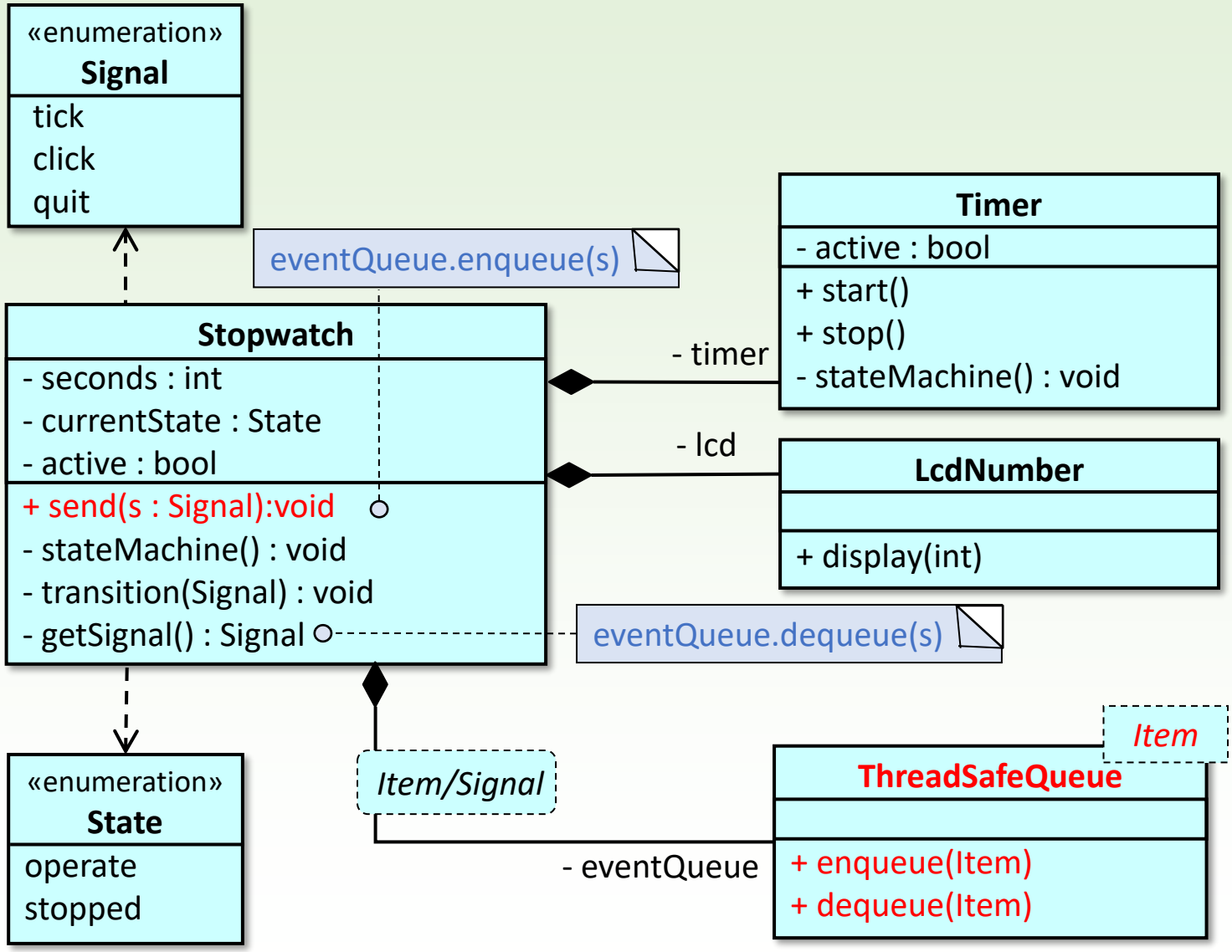
Communication diagram: realization



Stopwatch calls dequeue() on its own thread to process the events. This simulates an asynchronous behavior.

Az asynchronous message sending is replaced by a synchronous call (enqueue()) from a thread which is not on the thread of the state machine of :Stopwatch.

Class diagram: realization



Class Stopwatch

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

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

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

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

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

New thread for Stopwatch::stateMachine()
#include <thread>

stopwatch.h

Class Stopwatch

state machine of Stopwatch is run on separate thread

```
Stopwatch::Stopwatch() : _timer(this)
{
    _thread = new std::thread(&Stopwatch::stateMachine, this);
    _eventQueue.startQueue();
}
Stopwatch::~Stopwatch()
{
    _thread->join();
    _eventQueue.stopQueue();
}
```

waits for the ending of the state machine

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

stopwatch.cpp

Event handler of Stopwatch

```
void Stopwatch::transition(Signal signal)
{
    switch (_currentState) {
        case stopped:
            switch (signal) {
                case click: _currentState = operate; break;
                case tick : break;
                case quit : _active = false; break;
            }
            break;
        case operate:
            switch (signal) {
                case click: _currentState = stopped; break;
                case tick : _lcd.display(++_seconds); break;
                case quit : _active = false; break;
            }
            break;
    }
}
```

stopwatch.cpp

Class Timer

```
class Stopwatch;

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

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

    Stopwatch* _owner;

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

timer.h is already included by stopwatch.h, and Stopwatch is needed here.

new thread for Timer::stateMachine()
#include <thread>

timer.h

Class Timer

```
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);
    }
}
```

new thread for the state machine of Timer

semaphore for waiting
#include <condition_variable>
#include <mutex>

blocks the thread for 1 second

timer.cpp

Class LcdNumber

```
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

Template ThreadSafeQueue

```
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;
};
```

all threads blocked by _cond (state machine of stopwatch) are permitted to continue

threadsafequeue.h

Template ThreadSafeQueue

```
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();
    }
}
```

enqueue() and dequeue() may be called mutually exclusively

one of the threads blocked at _cond is allowed to continue (there is only one)

waits as long as the queue is empty and active

threadsafequeue.hpp

main()

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

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

    return 0;
}
```

main.cpp

```
Choice:
00:00
s
00:01
00:02
00:03
00:04
ss
00:05
00:06
00:07
s
s
00:08
00:09
00:10
s
q
```

Unique and general elements of an event-driven application

unique

- creating the necessary objects for the application and for this
  - (inheritance of) unique classes
  - plan their position on the user interface
- creating event handler functions
- connecting signals and event handler functions

general

- objects with typical appearance and signal sending habits (window, LCD display, timer)
- plan of the user interface
- event handling mechanism
  - asynchronous signal sending, (sender and receiver on different threads)
  - safe handling of the event queue

Stopwatch user interface

User interface is usually drawn by a visual planner, by which

- the stopwatch may be created on an individual window
- components of the stopwatch may be defined and instantiated (LCD display, timer)
- visible components may be arranged

The stopwatch is a window-like object the close of which triggers signal stop. It contains an LCD display, an invisible timer sending tick signals and a push button sending click signals.

Event handling of Stopwatch

To the signals triggered by the events handling methods have to be connected and the methods have to be implemented.

```
switch (currentState) {
    case stopped:
        switch (signal)
            case click: currentState := operate
            case tick: skip
            case quit:  timer.stop()
        endswitch
    case operate:
        switch (signal)
            case click: currentState := stopped
            case tick:   seconds := seconds +1
                         lcd.display(seconds)
            case quit: timer.stop()
        endswitch
endswitch
```

```
switch (signal) {
    case
        sw
        en
    case
        sw
        en
    case
        tir
endswitch
```

click:

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

tick:

```
switch (currentState) {
    case stopped:
    case operate: seconds := seconds +1
                  lcd.display(seconds)
endswitch
```

quit:

```
timer.stop()
```

Stopwatch developed by Qt

Stopwatch : public QWidget
Window-like controller object containing other controllers (timer, display, push button).
Its closure triggers signal quit.

QLCDNumber with method display

QPushButton-type object triggering signal click

```
#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
Among others, it takes care of the events to get the proper controllers where they trigger signals.

Class Stopwatch as a 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;
        QString Stopwatch::extend(int n) const;
    private slots:
        void oneSecondPass(); // tick
        void buttonPressed(); // click
};
```

event handler of signal quit triggered on exit

QTimer-type object triggers signal tick

event handler of the other signals

stopwatch.h

Qt event handlers of class Stopwatch

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

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

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

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

event handler of signal tick (same as the native C++ code in void transition())

event handler of signal tick (same as the native C++ code in void transition())

same as the native C++ code in class LcdNumber

stopwatch.cpp

Qt constructor of class Stopwatch

```
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);
}
```

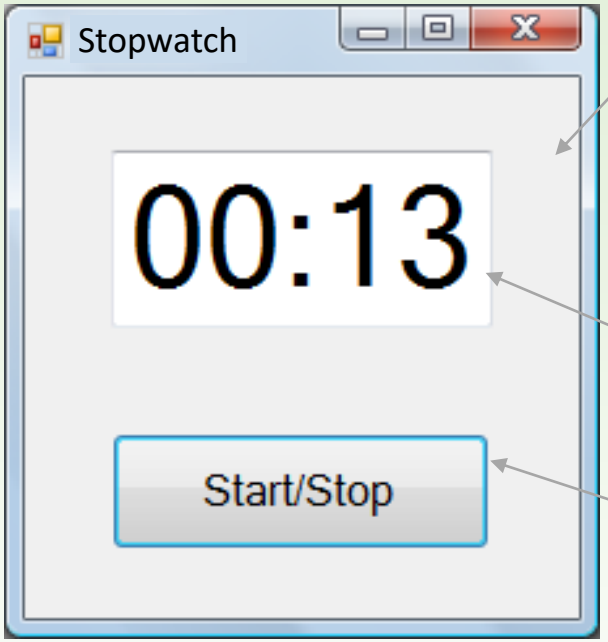
here the arrangement and the properties of the controllers are given

connecting signals and handlers:
tick (timeout()) ~ `oneSecondPass()`
click (clicked()) ~ `buttonPressed()`

part of the code may be generated automatically by using a visual designer (QtDesigner)

stopwatch.cpp

Stopwatch developed under .net



Stopwatch : Form
Window-like controller object containing other controllers (timer, display, push button).
Its closure triggers signal quit.

TextBox with method display

Button-type object triggering signal click

Application
Among others, it takes care of the events to get the proper controllers where they trigger signals.

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

program.cs

Class Stopwatch as a .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'));
    }
}
```

part of the code may be generated automatically by using a visual designer

event handlers of signals tick, click, and quit

this belongs to stopwatch instead of the LCD display

Stopwatch.cs

.net event handlers of Stopwatch

```
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();
}
```

Both belong to method transition() in native C++.

Stopwatch.cs

.net constructor of Stopwatch

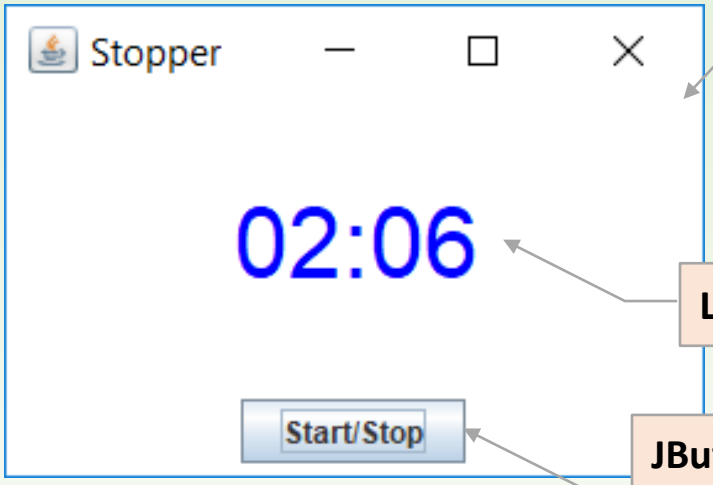
```
public Stopwatch(){
    this.components = new System.ComponentModel.Container();
    this.button = new System.Windows.Forms.Button();
    this.lcd = new System.Windows.Forms.TextBox();
    this.timer = new System.Windows.Forms.Timer(this.components);
    …
    this.Text = "Stopwatch";
    this.button.Text = "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();
}
```

Here, the arrangement and the properties of the controllers are given. It may be generated automatically with a visual designer.

connecting signals and their handlers

Stopwatch.cs

Stopwatch in Java



Stopwatch extends JFrame
Window-like controller object containing other controllers (timer, display, push button).
Its closure triggers signal quit.

LCDNumber with method display

JButton-type object triggering signal click

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

Stopwatch.java

Stopwatch in Java

```
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

Event handlers of Stopwatch in Java

```
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( format(seconds) );
            break;
        case stopped :   break;
    }
}

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

event handler of signal tick (same as the native C++ code in void transition())

event handler of signal click (same as the native C++ code in void transition())

signal quit implies the stopping of the timer

Stopwatch.java

Constructor of Stopwatch in Java

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

quit

connecting events with handlers

Stopwatch.java

LCD display in Java

```
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(String.format("%02d:%02d",
                (seconds % 3600) / 60,      // minutes
                (seconds % 3600) % 60));  // seconds
    }
}
```

formatting belongs to the display

LcdNumber.java

Object oriented program

- Object: vacation
- Oriented? YESS!
- Program:
  - 10am Breakfast
  - 11am Beach
  - 1pm Lunch
  - 2pm Siesta
  - 5pm Beach
  - 7pm Dinner
  - 8pm Go out
  - 1am Bedtime

vacation