**CIS 700: Final Project**

# ACSR – Java code Translation (Stopwatch Example)

**Su-Ting Chuang**
**Taehyun Kim**

## ACSR code

```
-----------------------------------------------------------------------------------------------------------------------
STOPWATCH = (SWATCH || FRACT(0) || SEC(0) || MIN(0) || HR(0)) \ {incrementFract,
incrementSec, incrementMin, incrementHr}

FRACT(f) = (incrementFract?,1).((f < 9)-> FRACT(f+1) + (f = 9) -> (incrementSec!,1).FRACT(0))
SEC(s) = (incrementSec?,1).((s < 59) -> SEC(s+1) + (s = 59) -> (incrementMin!,1).SEC(0))
MIN(m) = (incrementMin?,1).((m < 59) -> MIN(m+1) + (m = 59) -> (incrementHr!,1).MIN(0))
HR(h) = (incrementHr?,1).((h < 23) -> HR(h+1) + (h = 23) -> HR(0))

SWATCH = READY
READY = (start,1).RUNNING
RUNNING={(timer,2)}:(incrementFract!,1).RUNNING + (pause,1).PAUSED + (stop,1).STOPPED
PAUSED = (resume,1).RUNNING + (stop,1).STOPPED
STOPPED = (reset,1).READY
-----------------------------------------------------------------------------------------------------------------------
```

### *specification*

SWATCH, FRACT(0), SEC(0), MIN(0), and HR(0) operate concurrently, and ‘incrementFract’, ‘incrementSec’, ‘incrementMin’, and ‘incrementHr’ are synchronization events among these processes.

FRACT(f) waits for ‘incrementFract’ event from SWATCH, and when the event triggers, 0.1 sec increases. If the current fraction time is 9, this process sends event ‘incrementSec’ to SEC(s) to increase time by 1 sec. SEC(s), MIN(m), and HR(h) work in the similar way.

The initial state of the SWATCH process is ‘READY’. When event ‘start’ occurs, it moves to ‘RUNNING’ state. ‘RUNNING’ has three possible actions among which it can choose nondeterministically. If either ‘pause’ or ‘stop’ button is pressed, this process goes to ‘PAUSED’ and ‘STOPPED’ state, respectively. Otherwise, it waits for 0.1 sec, and then generates ‘incrementFract’ event to trigger FRACT(f) and goes back to RUNNING state.

# How to translate the ACSR code to Java code

## *Java classes*

StopwatchDisplay – GUI of stopwatch

Stopwatch – stopwatch implementation based on the acsr code

## *Implementation*

This program implements 6 threads:

- stopwatch (instance of Stopwatch class) for SWATCH
- fractThread for FRACT(f)
- secThread for SEC(s)
- minThread for MIN(m)
- hrThread for HR(h)
- tmrThread for timer

Stopwatch implements Runnable interface so that its instance is executed by a thread. This thread works as SWATCH. Also, four other threads for each time unit are created in Stopwatch class. To make synchronize these threads, we added the following code in each thread.

```
synchronized(this) {
    while (fractThreadSuspended)
        wait();
}
```

Only one thread at a time can own the lock for the given object **`this`**. Thus, if more than one thread tries to enter a section of code for which the lock must be acquired, then only one thread will get the lock and all other threads will block. When the current thread relinquishes the lock and makes another thread resumed, the corresponding thread can run. In this program, whenever one of 'incrementFract', 'incrementSec', 'incrementMin', and 'incrementHr' events is triggered, the thread which generates the event gives the lock to the thread which gets the event by assigning **`false`** to the boolean-type variable xThreadSuspended and call notifyAll().

We added another thread tmrThread to handle the timed action. This thread keeps sleeping for 0.1 sec and then waking up to increase time by 0.1 sec. When SWATCH is in 'READY' state and 'start' event occurs, this thread starts running. When 'pause' or 'stop' events are triggered, it stops increasing time by setting `tmrThreadSuspended = true;`, and when 'resume' button is pressed, this thread starts running again.