Timers

CPRE 388

Timers

- A timer provides a way to perform a delayed action or a periodic action
- The timer waits until a certain time interval has elapsed and then fires, sending a specified message to a specified object

Timers and Run Loops

- *NSRunLoop* objects control loops that wait for input, and they use timers to help determine the maximum amount of time they should wait.
- When the timer's time limit has elapsed, the run loop fires the timer (causing its message to be sent), then checks for new input.

Timing Accuracy

- the effective resolution of the time interval for a timer is limited to on the order of 50-100 milliseconds
- A repeating timer reschedules itself based on the scheduled firing time, not the actual firing time

Alternatives to Timers

- use *performSelector:withObject:afterDelay:* and related methods to invoke a method directly on another object
- Some variants, such as *performSelectorOnMainThread:withObject:waitUntilDone:*, allow you to invoke the method on a different thread.

Using Timers

- When you create a timer, you must configure it so that it knows what message to send to what object when it fires
- Three ways to create a timer:
  - scheduling a timer with the current run loop;
  - creating a timer that you later register with a run loop;
  - initializing a timer with a given fire date.

Timer Creation

- two ways to tell a timer what message it should send and the object to which it should send the message
  - specifying each independently
  - using an instance of *NSInvocation*
- - (void) *timerFireMethod:(NSTimer*)theTimer*

- timer controller object that declares methods to start / stop four timers configured in different ways.
- properties for two of the timers and a timer count, and three timer-related methods (*timerFireMethod*:, invocationMethod:, and *countedTimerFireMethod:*)
- also provides a method to supply a user info dictionary

```
@interface TimerController : NSObject {
    NSTimer *repeatingTimer;
    NSTimer *unregisteredTimer;
    NSUInteger timerCount;
}
@property (assign) NSTimer *repeatingTimer;
@property (retain) NSTimer *unregisteredTimer;
@property NSUInteger timerCount;
- (IBAction)        startOneOffTimer:sender;
-  (IBAction)       startRepeatingTimer:sender;
- (IBAction)        stopRepeatingTimer:sender;
```

```
- (IBAction) createUnregisteredTimer:sender;
- (IBAction) startUnregisteredTimer:sender;
- (IBAction) stopUnregisteredTimer:sender;
- (IBAction) startFireDateTimer:sender;
- (void) timerFireMethod:(NSTimer*)theTimer;
- (void) invocationMethod:(NSDate *)date;
- (void)
  countedTimerFireMethod:(NSTimer*)theTime
  r;
- (NSDictionary *)userInfo;
@end
```

```
- (NSDictionary *) userInfo {
    return [NSDictionary dictionaryWithObject:[NSDate
    date] forKey:@"StartDate"];
}
- (void) targetMethod:(NSTimer*) theTimer {
    NSDate *startDate = [[theTimer userInfo]
    objectForKey:@"StartDate"];
    NSLog(@"Timer started on %@", startDate);
}
- (void) invocationMethod:(NSDate *)date {
NSLog(@"Invocation for timer started on %@", date);
}
```

Scheduled Timers

The following two class methods automatically register the new timer with the current NSRunLoop object in the default mode (NSDefaultRunLoopMode):

- *scheduledTimerWithTimeInterval:invocation:repeats:*
- *scheduledTimerWithTimeInterval:target:selector:userInfo:repeats:*

```
- (IBAction)startOneOffTimer:sender {
    [NSTimer scheduledTimerWithTimeInterval:2.0
    target:self
    selector:@selector(targetMethod:)
    userInfo:[self userInfo]
    repeats:NO];
}
```

- The timer is automatically fired by the run loop after 2 seconds, and is then removed from the run loop.

- schedule a repeating timer, that again uses a selector:

```
- (IBAction)  startRepeatingTimer:sender {
   NSTimer *timer = [NSTimer
   scheduledTimerWithTimeInterval:0.5
   target:self
   selector:@selector(timerFireMethod:)
   userInfo:[self userInfo] repeats:YES];
   self.repeatingTimer = timer;
}
```

Unscheduled Timers

- create timers that you may schedule at a later time by sending the message *addTimer:forMode:* to an NSRunLoop object.
- *timerWithTimeInterval:invocation:repeats:*
- *timerWithTimeInterval:target:selector:userInfo:repeats:*

```
- (IBAction)   createUnregisteredTimer:sender {
    NSMethodSignature *methodSignature = [self
    methodSignatureForSelector:@selector(invocationMethod:
    )];
    NSInvocation *invocation = [NSInvocation
    invocationWithMethodSignature:methodSignature];
    [invocation setTarget:self];
    [invocation setSelector:@selector(invocationMethod:)];
    NSDate *startDate = [NSDate date];
    [invocation setArgument:&startDate   atIndex:2];
    NSTimer *timer = [NSTimer timerWithTimeInterval:0.5
    invocation:invocation
    repeats:YES];
    self.unregisteredTimer = timer;
}
```

```
- (IBAction)startUnregisteredTimer:sender {
   if (unregisteredTimer != nil) {
     NSRunLoop *runLoop = [NSRunLoop
        currentRunLoop];
   [runLoop addTimer:unregisteredTimer
   forMode:NSDefaultRunLoopMode];
   }
}
```

- create a timer with a given start time (one second in the future), and then start the timer by adding it to a run loop:

```
- (IBAction) startFireDateTimer:sender {
NSDate *fireDate = [NSDate
    dateWithTimeIntervalSinceNow:1.0];
NSTimer *timer = [[NSTimer alloc]
    initWithFireDate:fireDate
    interval:0.5
    target:self
    selector:@selector(countedtargetMethod:)
    userInfo:[self userInfo]
    repeats:YES];
```

```
    timerCount = 1;
    NSRunLoop *runLoop = [NSRunLoop
        currentRunLoop];
    [runLoop addTimer:timer
        forMode:NSDefaultRunLoopMode];
    [timer release];
}
```

- although the timer is configured to repeat, it will be stopped after it has fired three times by the *countedtargetMethod:*

Stopping a Timer

- If you create a repeating timer, however, you stop it by sending it an *invalidate* message.
- You can also send a non-repeating timer an *invalidate* message before it fires to prevent it from firing

```
- (IBAction) stopRepeatingTimer:sender {
    [repeatingTimer invalidate];
    self.repeatingTimer = nil;
}
- (IBAction) stopUnregisteredTimer:sender {
    [unregisteredTimer invalidate];
    self.unregisteredTimer = nil;
}
```

- You can also invalidate a timer from the method it invokes.

```
- (void)  countedtargetMethod:(NSTimer*) theTimer  {
    NSDate *startDate = [[theTimer userInfo]
          objectForKey:@"StartDate"];
    NSLog(@"Timer started on %@; fire count %d",
    startDate, timerCount);
    timerCount++;
    if (timerCount  >  3) {
       [theTimer invalidate];
    }
}
```