(INTERMEDIATE) JAVA PROGRAMMING

Multithreading: Wait and Notify

Chapter 13

JAVA:
REVIEW: SYNCHRONIZATION

Review: Thread Synchronization

- 멀티스레드 프로그램 작성시 주의점
  - 다수의 스레드가 공유 데이터에 동시에 접근하는 경우
    - 공유 데이터의 값에 예상치 못한 결과 발생 가능
- 스레드 동기화
  - 멀티스레드의 공유 데이터의 동시 접근 문제 해결책
    - 공유 데이터를 접근하는 모든 스레드의 한 줄 세우기
    - 한 스레드가 공유 데이터에 대한 작업을 끝낼 때까지 다른 스레드가 대기하도록 함

Review: 스레드 동기화 기법

- 스레드 동기화
  - 공유 데이터에 동시에 접근하는 다수의 스레드가 공유 데이터를 배타적으로 접근하기 위해 상호 협력(coordination)하는 것
  - 동기화의 핵심
    - 스레드의 공유 데이터에 대한 배타적 독점 접근 보장
- 자바에서 스레드 동기화를 위한 방법
  - synchronized로 동기화 블록 지정
  - **wait()-notify() 메소드로 스레드 실행 순서 제어**

Review: synchronized 블록 지정

- synchronized 키워드
  - 한 스레드가 독점 실행해야 하는 부분(동기화 코드)을 표시하는 키워드
    - 임계 영역(critical section) 표기 키워드
  - 메소드 전체 혹은 코드 블록
- synchronized 블록에 대한 컴파일러의 처리
  - 먼저 실행한 스레드가 모니터 소유
    - 모니터란 해당 객체를 독점적으로 사용할 수 있는 권한
  - 모니터를 소유한 스레드가 모니터를 내놓을 때까지 다른 스레드 대기

```
synchronized void add() {
    int n = getCurrentSum();
    n+=10;
    setCurrentSum(n);
}
```

synchronized 메소드

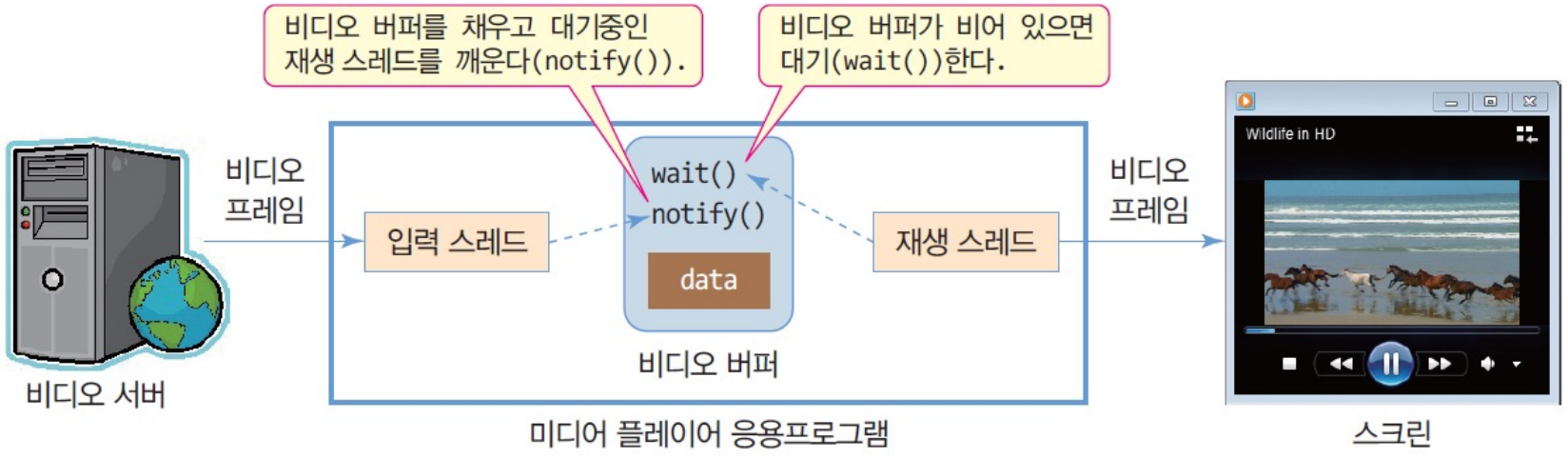
```
void execute() {
    // 다른 코드들
    //
    synchronized(this) {
        int n = getCurrentSum();
        n+=10;
        setCurrentSum(n);
    }
    //
    // 다른 코드들
}
```

synchronized 코드 블록

JAVA:
THREAD: WAIT / NOTIFY

producer-consumer 문제와 동기화

- producer-consumer 문제
  - producer : 공유 메모리에 데이터를 공급하는 스레드
  - consumer : 공유 메모리의 데이터를 소비하는 스레드
  - 문제의 본질
    - producer와 consumer 가 동시에 공유 데이터를 접근하는 문제
- producer-consumer 문제 사례
  - 미디어 플레이어
    - producer:입력스레드, consumer:재생스레드, 공유데이터:비디오버퍼

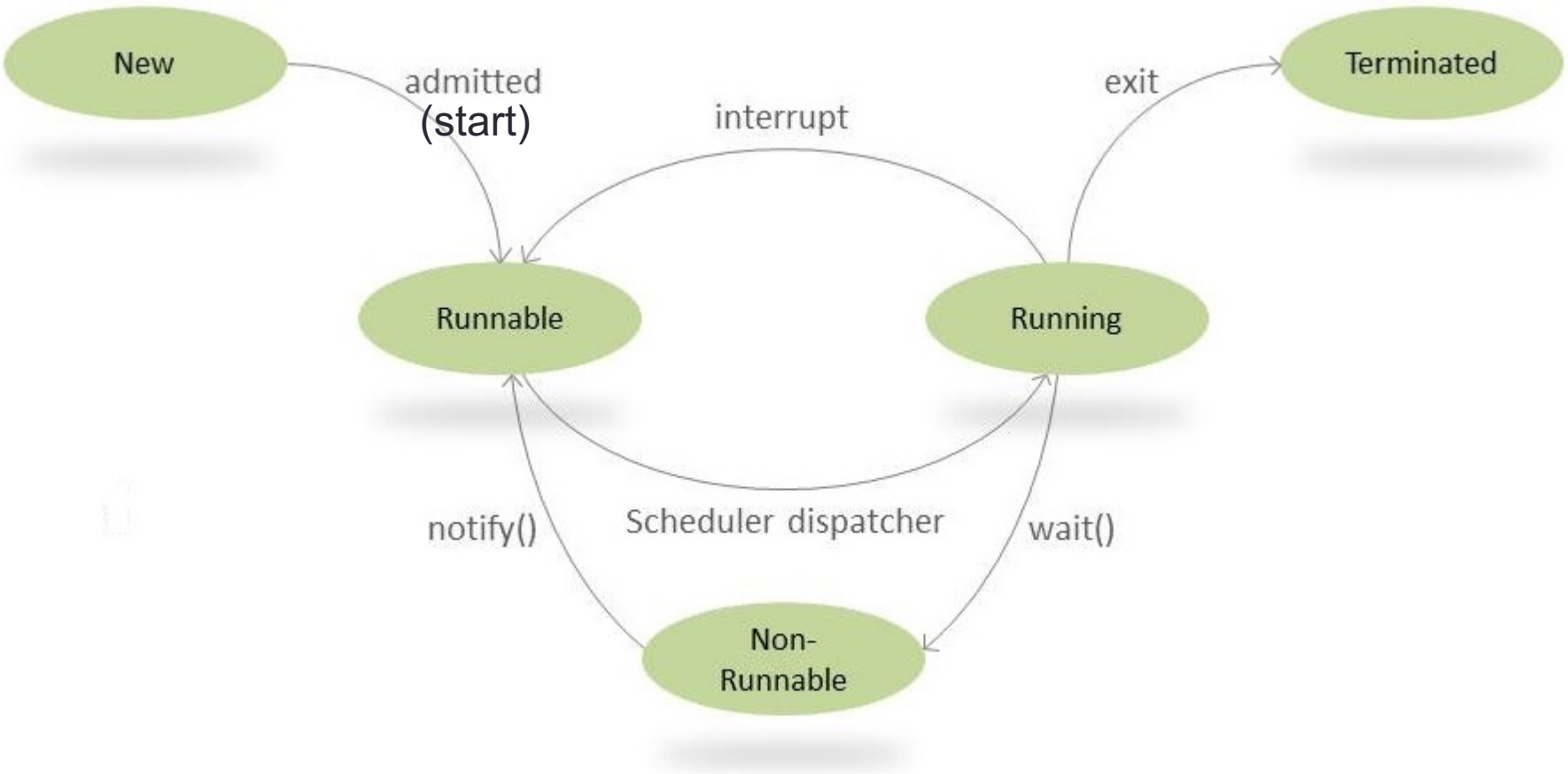


wait(), notify(), notifyAll()를 이용한 동기화

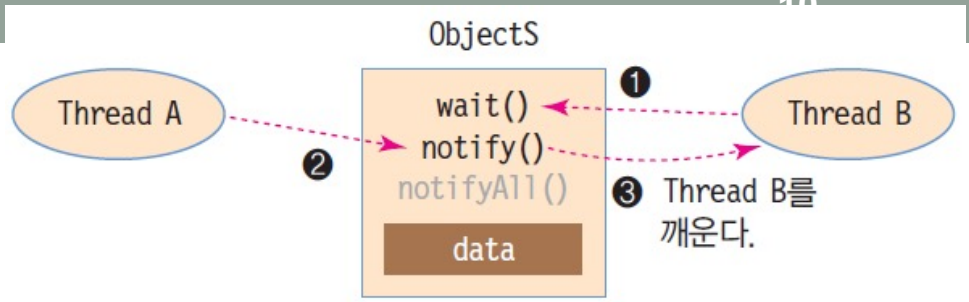
- 동기화 객체
  - 두 개 이상의 스레드 동기화에 사용되는 객체
- 동기화 메소드
  - wait()
    - 다른 스레드가 notify()를 불러줄 때까지 기다린다.
  - notify()
    - wait()를 호출하여 대기중인 스레드를 깨우고 RUNNABLE 상태로 만든다.
    - 2개 이상의 스레드가 대기중이라도 오직 한 스레드만 깨운다.
  - notifyAll()
    - wait()를 호출하여 대기중인 모든 스레드를 깨우고 모두 RUNNABLE 상태로 만든다.
  - synchronized 블록 내에서만 사용되어야 함

- wait(), notify(), notifyAll()은 Object의 메소드
  - 모든 객체가 동기화 객체가 될 수 있다.
  - Thread 객체도 동기화 객체로 사용될 수 있다.

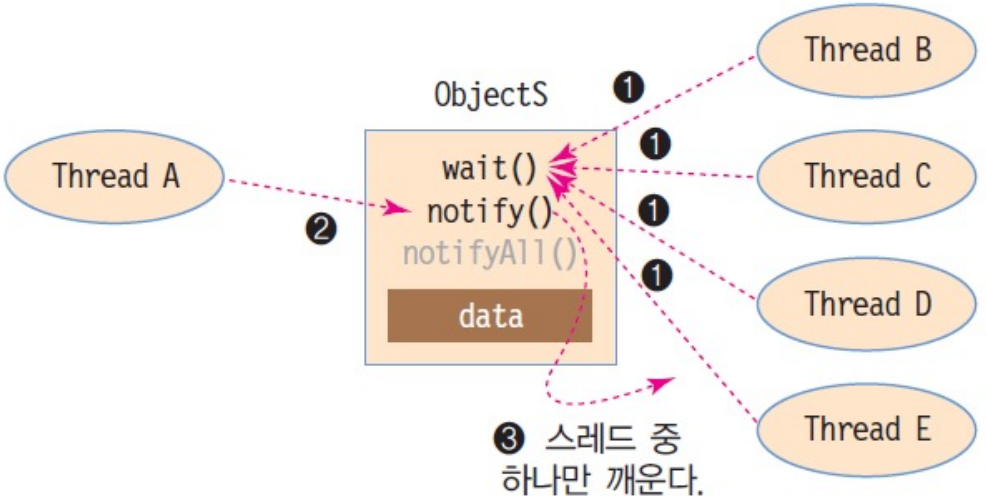
Lifecycle of a Thread



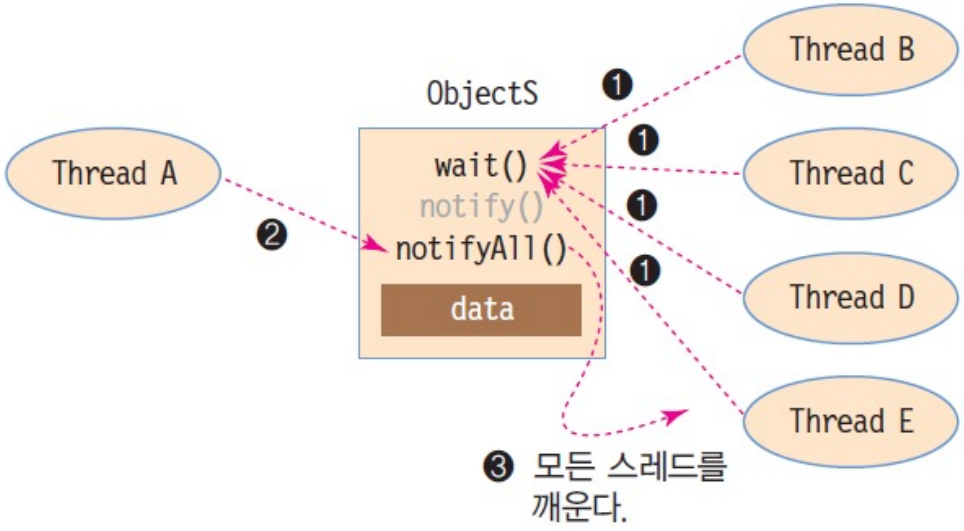
Thread A가 ObjectS.wait()를 호출하여 무한 대기하고, Thread B가 ObjectS.notify()를 호출하여 ObjectS에 대기하고 있는 Thread A를 깨운다.



4 개의 스레드가 모두 ObjectS.wait()를 호출하여 대기하고, ThreadA는 ObjectS.notify()를 호출하여 대기 중인 스레드 중 하나만 깨우는 경우



4 개의 스레드가 모두 ObjectS.wait()를 호출하여 대기하고, ThreadA는 ObjectS.notifyAll()를 호출하여 대기중인 4개의 스레드를 모두 깨우는 경우



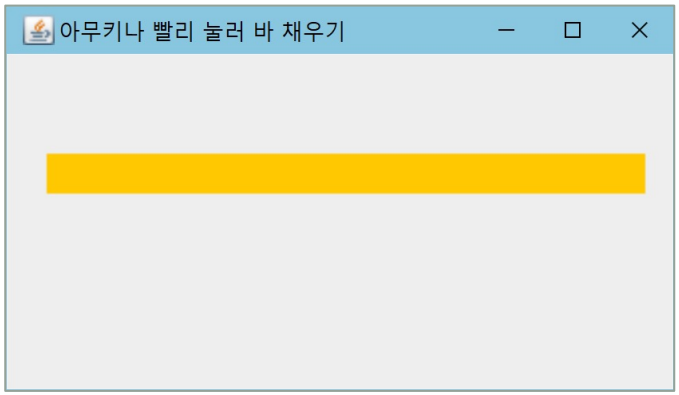
예제 13-6 : wait(), notify()를 이용한 바 채우기

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

class MyLabel extends JLabel {
    int barSize = 0; // 바의 크기
    int maxBarSize;

    MyLabel(int maxBarSize) {
        this.maxBarSize = maxBarSize;
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(Color.MAGENTA);
        int width = (int)(((double)(this.getWidth()))
                /maxBarSize*barSize);
        if(width==0) return;
        g.fillRect(0, 0, width, this.getHeight());
    }

    synchronized void fill() {
        if(barSize == maxBarSize) {
            try {
                wait();
            } catch (InterruptedException e) { return; }
        }
        barSize++;
        repaint(); // 바 다시 그리기
        notify();
    }
```

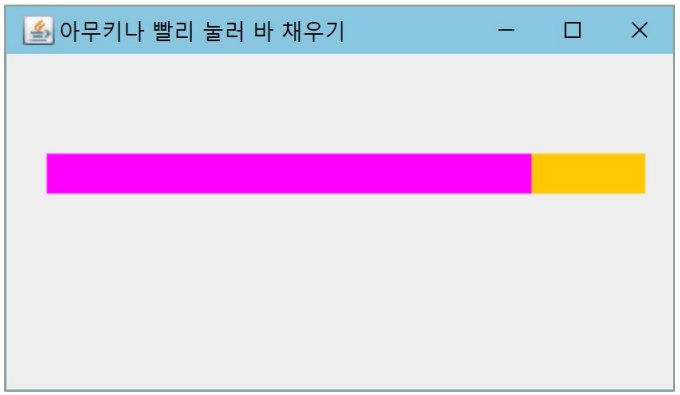
```
    synchronized void consume() {
        if(barSize == 0) {
            try {
                wait();
            } catch (InterruptedException e)
                { return; }
        }
        barSize--;
        repaint(); // 바 다시 그리기
        notify();
    }
}

class ConsumerThread extends Thread {
    MyLabel bar;

    ConsumerThread(MyLabel bar) {
        this.bar = bar;
    }
    public void run() {
        while(true) {
            try {
                sleep(200);
                bar.consume();
            } catch (InterruptedException e)
                { return; }
        }
    }
}
```

```
public class TabAndThreadEx  extends
JFrame {
    MyLabel bar = new MyLabel(100);
    TabAndThreadEx(String title) {
        super(title);
        this.setDefaultCloseOperation
                (JFrame.EXIT_ON_CLOSE);
        Container c = getContentPane();
        c.setLayout(null);
        bar.setBackground(Color.ORANGE);
        bar.setOpaque(true);
        bar.setLocation(20,  50);
        bar.setSize(300, 20);
        c.add(bar);

        c.addKeyListener(new KeyAdapter() {
            public void keyPressed(KeyEvent e)
            {
                bar.fill();
            }
        });
        setSize(350,200);
        setVisible(true);

        c.requestFocus();
        ConsumerThread th = new
            ConsumerThread(bar);
        th.start(); // 스레드 시작
    }

    public static void main(String[] args) {
        new TabAndThreadEx(
            "아무키나 빨리 눌러 바 채우기");
    }
}
```

실행 결과



초기 화면



키를 반복하여 빨리 누른 화면

(INTERMEDIATE) JAVA PROGRAMMING

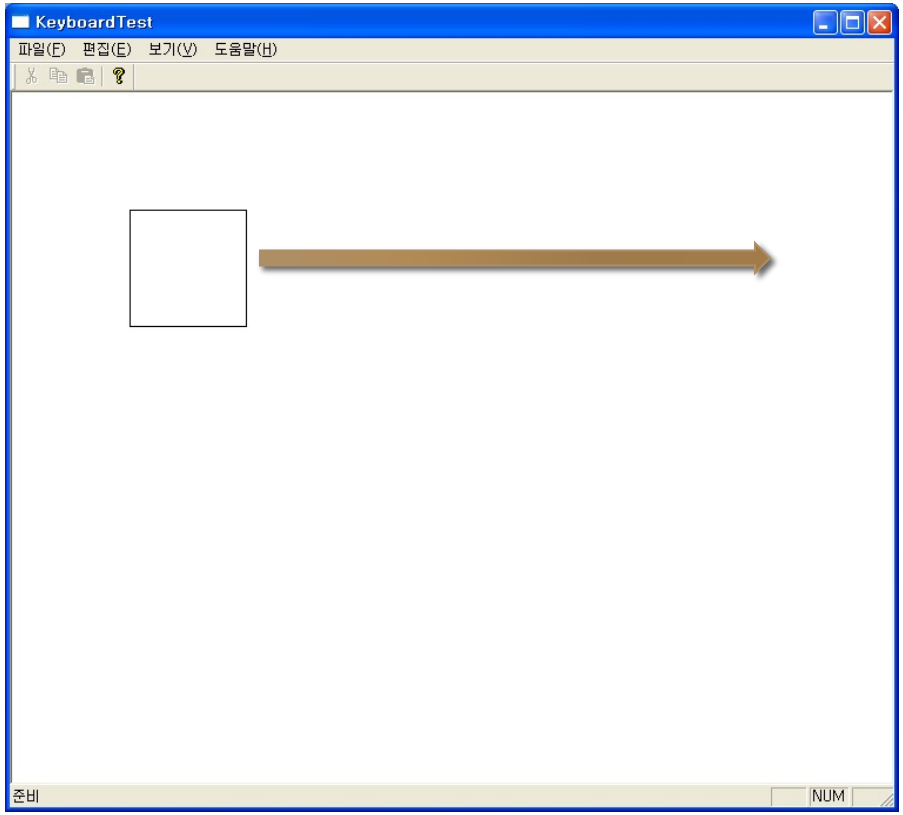
Thread Practice

Chapter 13

JAVA:
THREAD PRACTICE: DYNAMICS

코딩 연습

- 사각형이 죽~ 움직이는 장면 만들기



1. 위치 저장 변수 선언
   Point pt;

2. 위치에 사각형그리기
   (paintComponent)
   g.drawRect(pt.x, …);

3. 패널에 **Runnable** 인터페이스
   (public void run())

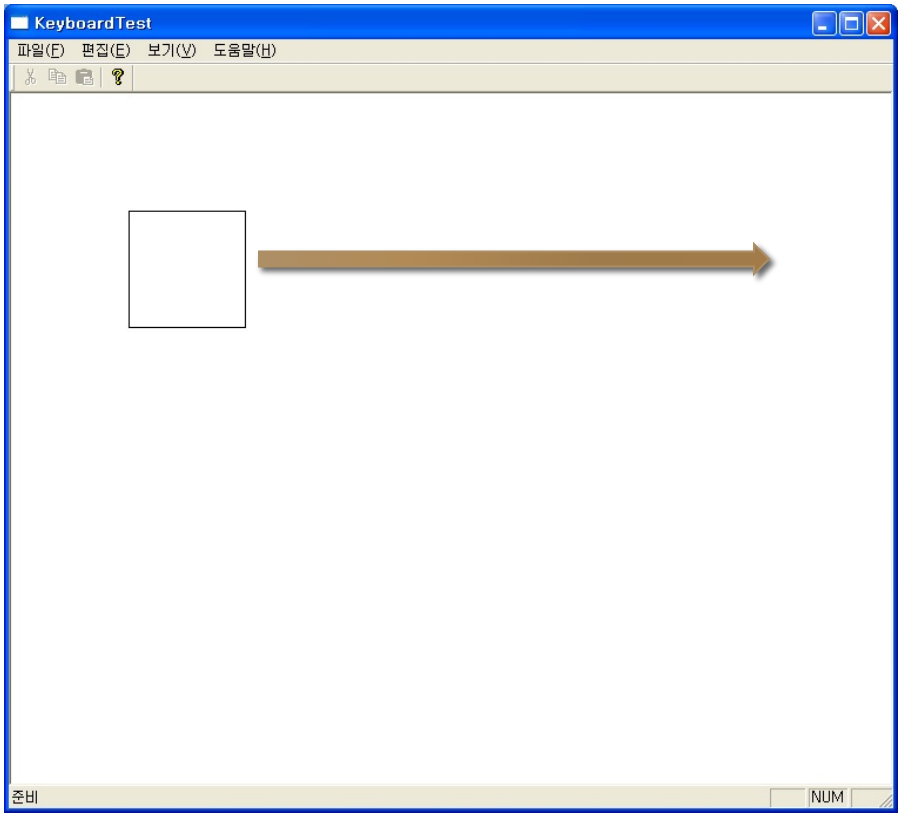
4. **Thread** 정의 및 시작
   Thread t = new Thread(this);

5. **run** 함수 구현
   (run)
   while(true)
   …..

운동학 (= Dynamics)

- 도형의 위치를 의미하는 변수의 값을 시간에 따라 변화 시켜 주는 것

- Ex)



속도 일정
➔재미 없다!

보다 복잡한 움직임은?

Having more fun with the Thread

- Still Image
  - One Image

Cézanne, Paul
Still Life

- Animation
  - Lots of Images!
  - http://www.youtube.com/watch?v=UocF4ycBnYE

Having more fun with the Timer

- 시간에 따라 그림이 변한다 = **Dynamics**

- 도형의 값(values/properties)?
  - 색
  - 모양
  - 위치

보다 재미있는 운동을 위한 약간의 물리 수업!

- 물체의 운동을 기술하기 위해 필요한 값들
  - 위치 (position) : 보통 p 로 표현

    p(t) : t초 때 위치
  - 속도 (velocity) : 위치의 시간에 따른 변화 (dp/dt)

    v(t)= p(t+1) – p(t)
  - 가속도 (acceleration) : 속도의 시간에 따른 변화 (dv/dt)

    a(t) = v(t+1) – v(t)

시간에 따른 위치가 주어지면 속도, 가속도를 구할 수 있다.
그 반대는?

가속도가 주어지면?

- 물체의 운동을 기술하기 위해 필요한 값들
  - 가속도 (acceleration) : 보통 a 로 표현
    a(t) : t 초 때 가속도
  - 속도 (velocity) : 1 초 후 속도차이는 가속도 만큼
    v(t+1) = v(t) + a(t)*1초
  - 위치 (position) : 1 초 후 위치 차이는 속도 만큼
    p(t+1) = p(t) + v(t)*1초

가속도를 주는 식: 운동방정식

- 뉴튼의 운동 방정식(Equation of Motion)
  - **f = ma**
  - 힘이 주어지면 가속도를 계산할 수 있다.
  - 가속도가 주어지면 Δt 초 후의 속도가 계산 가능
  - 속도가 주어지면 Δt 초 후의 위치가 계산 가능

- Ex)
  - 자유낙하
    - 힘 = 중력가속도 g ($=-9.8 m/sec^2$) * 무게
  - 스프링
    - 힘 = 기준 위치와의 차이 $f = k x$ (k: 스프링 상수)

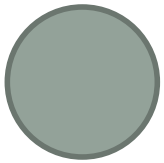
운동학을 프로그래밍하자!

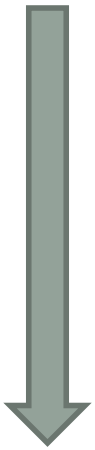
1. 위치/속도/가속도를 저장할 변수를 만든다
   (m_p, m_v, m_a)

2. 정해진 시간마다 다음의 일을 반복
   1. 주어진 상태에서의 힘 계산 (ex) 중력 or 스프링힘)
   2. 가속도 값 갱신 (a = f/m)
   3. 속도 값 갱신 (v = v + a * dt)
   4. 위치 값 갱신 (p = p + v * dt)
   5. 변경된 위치에 그림 그리기

코딩 연습

- 공 튕기기





아래와 같은 순서대로 각자 코딩 해 보자

1. 정해진 위치에서 공이 자유 낙하
   → a = g

2. 마우스로 클릭하면 공의 위치를 다시 세팅

3. 밑에 벽이 있어 공이 다시 튀겨 올라 간다
   벽에 닿는 순간 다음과 같이 값을 변경
   → p(t+dt) = 벽과 닿은 위치
   → v(t+dt) = - e * v (t) (e: 반발계수, 보통 0.8)

4. 마우스로 공을 클릭하면 그 순간만 가속도 증가 (= 드리볼)
   → a(t) = g + f (f: 임의의 값)
   → 이 후 다시 a = g로 회귀 해야 함

더 다양한 예제를 찾아보자