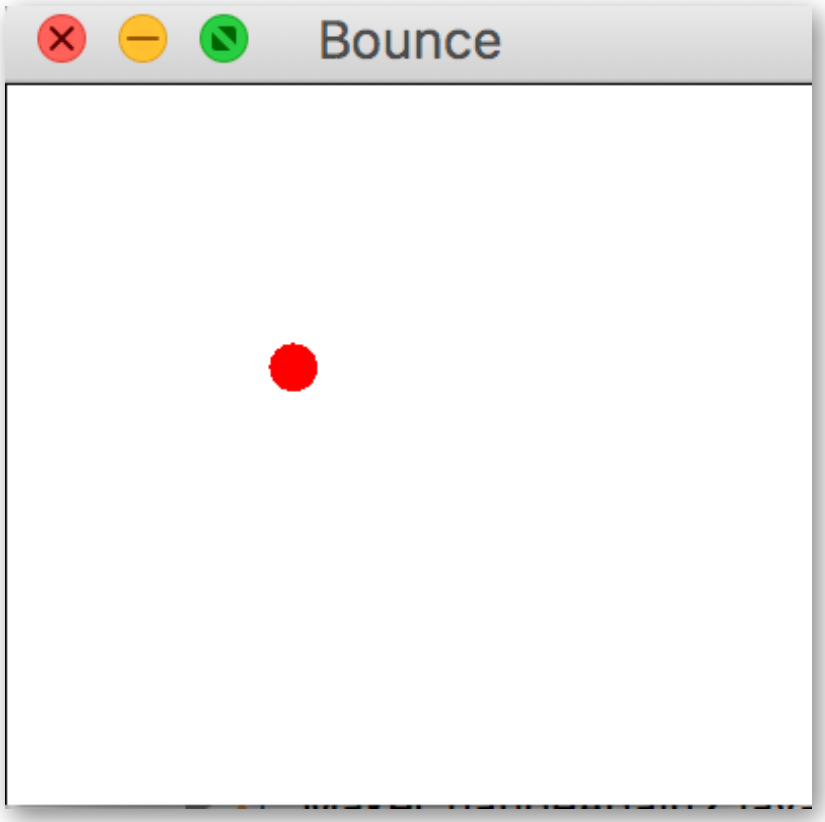
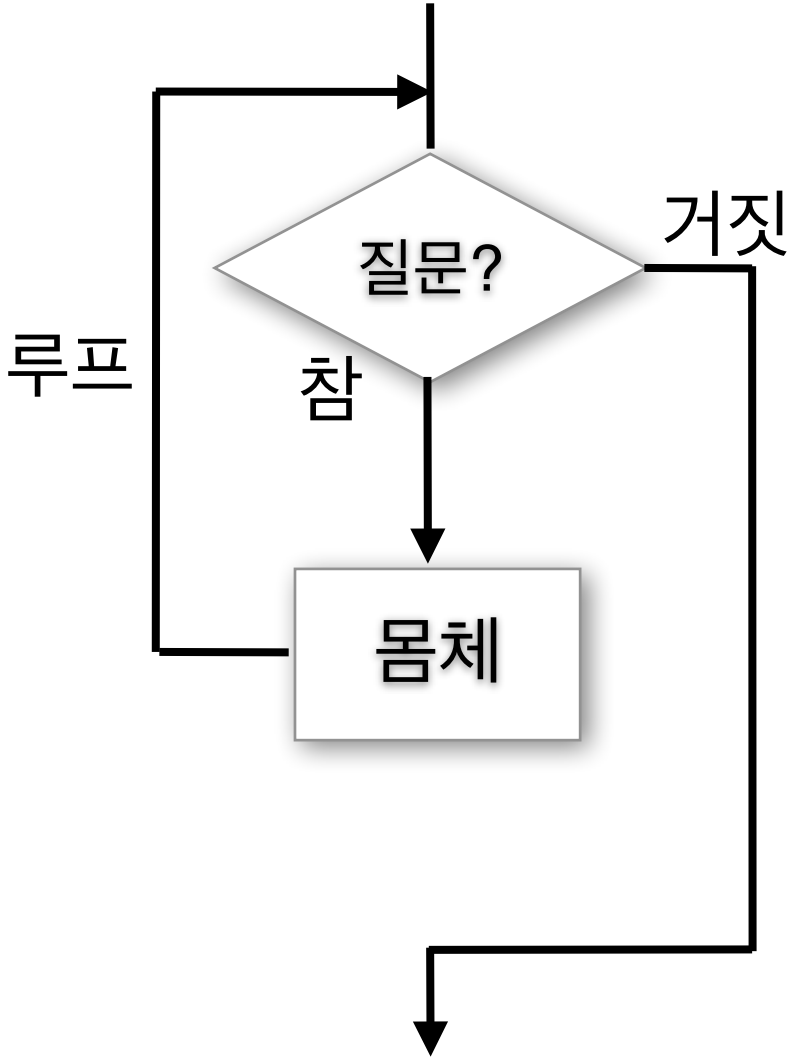
7
반복: 루프와 재귀호출

공 튀기기



while 루프

- 문법
  - while (질문?) { 몸체 }
  - 질문?은 논리계산식
  - 몸체는 명령문의 나열
- 의미구조
  - 질문을 계산한다.
  - 질문이 참이면 몸체를 실행하고 처음부터 다시
  - 질문이 거짓이면 루프 종료



루프는 반복

- 정해진 반복 (definite iteration)
  - 반복 횟수가 시작시 알고 있는 경우
- 정해지지 않은 반복 (indefinite iteration)
  - 반복 횟수가 시작시 모르는 경우
- 제한없는 반복 (unbounded iteration)
  - 끝없이 반복하는 경우
  - 루프가 발산한다(diverge)고 한다.

정해진 반복 예

- 평균을 구하자
  - average = (exam1+...+examN)/N
- 알고리즘
  - “시험횟수” N을 알고 있다고 가정
  - 총점 = 0, 완료횟수=0
    - 시험횟수와 완료횟수가 같을 때까지 아래를 반복
      - 다음 시험점수를 입력받고,
      - 시험점수를 총점에 누적하고,
      - 완료횟수를 1 증가시킨다.
    - 총점/시험횟수를 계산하여 평균을 구한다.

```
public double computeAverage(int how_many)
{
   double total_points = 0.0; // 총점
   int count = 0; // 얼마나 읽었나? 루프 카운터
   while (count != how_many)
   // 매 반복마다 다음을 만족한다. 루프 불변식 (loop invariant)
   //   total_score == exam_1 + exam_2 + ... + exam_count
   {
      // 시험점수를 입력 받는다.
      String input = JOptionPane.showInputDialog
                           ("Type next exam score:");
      int score = new Integer(input).intValue();
      total_points = total_points + score; // 누적
      count = count + 1;
      // 진행결과를 인쇄
      System.out.println("count = " + count +
                         "; total = " + total_points);
   }
   // 결론: total_points == exam_1 + exam_2 + ... + exam_how_many
   return (total_points / how_many);
}
```

루프 불변식 (Loop Invariant)

- 루프가 돌면서 매번 시작지점에서 만족하는 성질
- 이 성질을 알지 못한다면 우리가 루프를 제대로 작성하고 있는 것이 아니다.
  - 예, sum이 1~count까지 더한 수이다.
- 루프 불변식으로부터 루프가 끝났을 때 만족하는 성질을 결론 지을 수 있다.

주목: 정해진 반복

- 정해진 반복은 보통 다음과 같은 패턴
  - 몇 번 돌았는지 루프 카운터(loop counter)로 기억
  - 질문은 루프 카운터가 충분히 돌았는지 검사
  - 몸체에서 루프 카운터를 증가

- 패턴

```
int count = 초기값;
while (count 질문) {
  몸체
  count 증가
}
```

```
int count = 초기값;
while (count 질문) {
  count 증가
  몸체
}
```

잘못하면, 무한반복 (non-terminate) 가능

처음에 총 횟수를 -1로 주면?

```
public double computeAverage (int how_many)
{
   double total_points = 0.0;
   int count = 0;
   while ( count != how_many ) {
      String input = JOptionPane.showInputDialog
         ("Type next exam score:");
      int score = new Integer(input).intValue();
      total_points = total_points + score;
      count = count + 1;
       System.out.println("count = " + count +
                          "; total = " + total_points);
   }
   return (total_points / how_many);
}
```

수정 (Remedy)

처음에 총 횟수를 -1로 주면?

```
public double computeAverage (int how_many)
{
   double total_points = 0.0;
   int count = 0;
   while ( count < how_many ) {
      String input = JOptionPane.showInputDialog
         ("Type next exam score:");
      int score = new Integer(input).intValue();
      total_points = total_points + score;
      count = count + 1;
       System.out.println("count = " + count +
                          "; total = " + total_points);
  }
  return (total_points / how_many);
}
```

정해지지 않은 반복: 입력 처리

- 사용자에게 입력을 여러 개 받는 경우, 처음에 그 갯수를 정하지 않고 입력받는 도중에 끝을 아는 경우가 편리한 경우가 많다.
  - "취소" 단추 또는 "끝내기" 단추

```
boolean processing = true;
while (processing)
  { 입력을 받고
    if (종료하라고 해?) processing = false;
    else                { 여기서 실제로 일을 하자 }
  }
```

예제, 평균 내기

- 사용자에게 입력을 취소 단추 누를때까지 받는다.

```
public double computeAverage()
{
   double total_points = 0.0;
   int count = 0;
   boolean processing = true;

   while (processing) {
      String input = JOptionPane.showInputDialog
              ("Type next exam score (or press Cancel to quit):");
      if (input == null) processing = false; // 취소 단추를 누르셨군요
      else { // 정상적으로 수행 ... }
   }

   if ( count == 0 ) // 처음부터 취소 단추를 누른 경우
      throw new RuntimeException("error: no input supplied");

  return (total_points / count);
}
```

정해지지 않은 반복: 검색

- 여러 값 들의 모임에서 검색을 할 때는 찾을 때까지, 또는 끝까지 반복한다.

```
boolean item_found = false;
모임에서 첫번째 원소를 정한다.
while (!item_found && 찾을것이 더 있니?)
  { 현재 것을 검사
    if (원하는 것이면) item_found = true;
    else               { 다음 것을 정한다. }
  }
```

예제, 문자열 검색

- 문자열 s와 문자 c가 주어지면 s를 왼쪽부터 보아서 처음 c가 나오는 위치를 찾아라.

- 알고리즘
  - 위치(index)를 0으로 둔다.
  - 위치가 문자열 길이보다 작을 때까지 다음을 반복한다.
    - s.charAt(index)가 c이면 반복을 종료한다.
    - 아닌 경우, 위치(index)를 증가시킨다.

```
public int findChar(char c, String s)
{
   boolean found = false;  // s에서 c를 찾았는지?
   int index = 0;          // 지금 s에서 어느 위치를 보고 있는지?

   while (!found && index<s.length()) {
   // 루프 불변식:
   // (1) found==false 경우, c가 s의 0...(index-1)까지 없었다.
   // (2) found==true  경우, c==s.charAt(index)
      if (s.charAt(index) == c)
         found = true;
      else
         index = index + 1;
     }
   if (!found) index = -1; // 못찾은 경우 -1를 반환
   return index;
}
```

예제, 소수(Prime Number)니?

- 정수 n을 받아서 소수인지 답하여라.

- 알고리즘
  - 현재(current)를 n/2로 둔다.
  - current가 2보다 작아질 때까지 다음을 반복한다.
    - n % current ==0 이면 반복을 종료한다. (소수가 아니다)
    - 아닌 경우, current를 1 감소시킨다.

```
public int isPrime(int n)
{
   if (n<2)
      throw new RuntimeException("error: invalid " + n );
   else {
      boolean found = false; // 인수를 찾았니?
      int c = n/2;           // 현재 검색 중인 것
      while (!found && c>1) {
      // 루프 불변식:
      // (1) found==false 경우, n이 c+1...n/2로 나누어 지지 않는다.
      // (2) found==true  경우, n이 c로 나누어진다.
      if (n%c == 0)
            found = true;
         else
            c = c-1;
   }
   if (!found) return 1; // 못찾은 경우 1을 반환
   return c;
}
```

7.6 세는 루프 for-문

- 앞서 예제에서 보았듯이 다음 패턴의 루프가 자주 사용된다.
  - 변수를 초기화하고
  - 매 반복마다 그 변수에 대한 질문을 하고
  - 매 반복마다 그 변수를 변경하는

- 세는 루프 편하게 작성하기
  - `for(초기화; 질문; 변경) { 루프 몸체 }`

```
public int findChar(char c, String s)
{
   boolean found = false;
   int index = 0;
   while (!found && index<s.length()) {
      if (s.charAt(index) == c)
         found = true;
      else
         index = index + 1;
   if (!found) index = -1;
   return index;
}
```

```
public int findChar(char c, String s)
{
   int index;
   for(index=0; index<s.length() && s.charAt(index)!=c; index++);
   if (index==s.length()) index = -1;
   return index;
}
```

```
public int isPrime(int n) {
   boolean found = false; // 인수를 찾았니?
   int c = n/2;           // 현재 검색 중인 것
   while (!found && c>1) {
      if (n%c == 0) found = true;
      else c = c-1;
   }
   if (!found) return 1; // 못찾은 경우 1을 반환
   return c;
}
```

```
public int isPrime(int n) {
   int c;
   for(c=n/2; c>1 && n%c!=0; c=c-1);
   return c;
}
```

주의: 세는 변수의 선언

- 세는 변수의 선언을 어떻게 하느냐에 따라 세는 변수의 유효 범위가 달라진다.

```
for(int i=0; i<10; i++)
  { <body> }
```

=

```
{
   int i=0;
   while (i<10)
   {
     <body>
     i++
   }
}
```

```
int i;
for(i=0; i<10; i++)
  { <body> }
```

=

```
int i;
{
   i=0;
   while (i<10)
   {
     <body>
     i++
   }
}
```

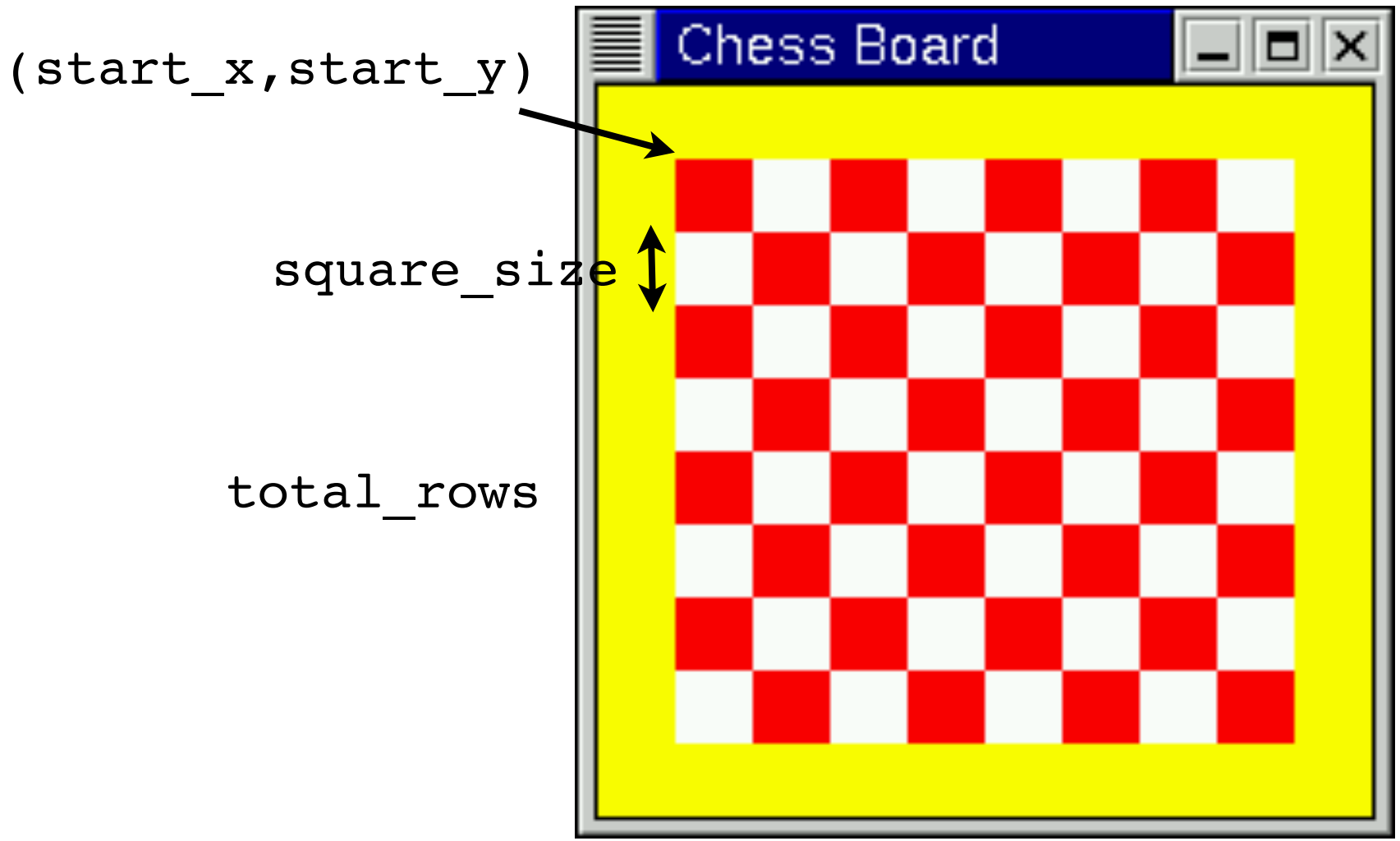
중첩 루프 (Nested Loops)

- 루프 안에 루프가?!
  - 지금까지는 1차원 평면을 돌아 다녔다.
  - 2차원 또는 고차원 평면을 돌아 다니려면 중첩된 루프를 사용할 필요가 있다.

예제, 구구단을 외자

```
for (int i=1; i<10; i=i+1) {
   // 루프 불변식: 구구단의 i-1단까지 출력했다.
   for (int j=1; j<10; j=j+1) {
      // 루프 불변식: 구구단의 i-1단까지 출력했고, i단의 j항까지 출력했다.
      System.out.print( i + "*" + j + "=" + (i*j) + " ");
   }
   System.out.println();
}
```

예제, 체스판 그리기



```
private void paintBoard (int start_x, int start_y,
                         int total_rows, int square_size, Graphics g)
{
    for (int x = 0; x < total_rows; x++) {
    // 루프불변식: x열까지 그렸음
        int x_position = start_x + (x * square_size);
        for ( int y = 0; y < total_rows; y++ ) {
        // 루프불변식: x열의 y칸까지 그렸음
            int y_position = start_y + (y * square_size);
            if ( ((x + y) % 2) == 0 )
                g.setColor(Color.red);
            else
                g.setColor(Color.white);
            g.fillRect (x_position, y_position,
                    square_size,square_size);
        }
    }
}
```

재귀호출 프로그래밍

- n!을 구하는 함수를 작성하세요.

- 루프 (bottom-up: 작은 것부터 계산하여 결과 쌓아올리기)
  - 0! = 1
  - 1! = 1
  - 2! = 1 · 2

    …
  - n! = 1 · 2 · 3 · ... · (n-1) · n

- 재귀 (top-down: 주어진 큰 문제를 더 작은 문제들로 쪼개기)
  - n! → n · (n-1)! → n · (n-1) · (n-2)! → n · (n-1) · ... · 2 · 1

재귀로 문제풀기

- 가장 작은 문제에 대해 풀자. 일반적으로 큰 일 없이 풀릴 것이다.
  - 예, 1! = 1
- 보다 큰 문제를 풀때, 보다 작은 문제에 대해서 놀랍게도 이미 풀렸다고 가정하자. 그러면 큰 문제가 풀릴 것이다.
  - 예, n! = n · (n-1)!
- 이 두 가지를 엮으면 재귀적으로 문제가 풀린 것이다.

수학적 귀납법 (Mathematical Induction)

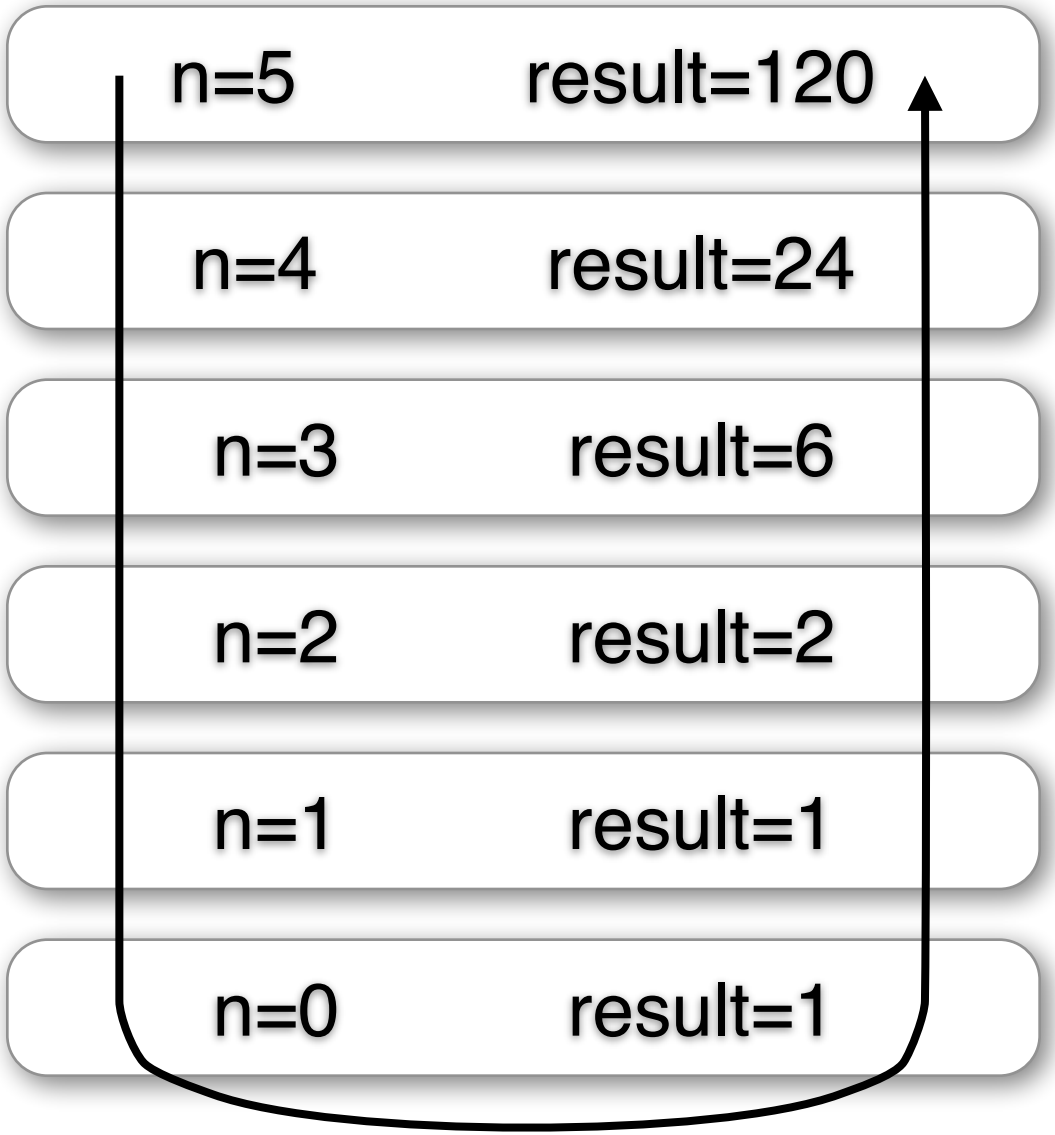
- 모든 자연수 n에 대해 증명하고자 하는 성질을 P(n)이라고 하자.
- P(0)를 증명한다.
- i<k인 모든 i에 대해 P(i)가 참이라고 가정하고, P(k)를 증명한다.
- 수학적 귀납법에 의해 모든 자연수 n에 대해 P(n)이 사실이다.

재귀로 푼 계승

```
public long fac (int n)
{
   if(n==0)
      return 1;                  // 0!=1
   else
      return (n * fac(n-1)); // n!=n*(n-1)!
}
```

재귀의 실행

```
public long fac (int n)
{
   if(n==0)
      return 1;
   else
      return (n * fac(n-1));
}
```



재귀 vs 루프

```
public long fac (int n)
{
   if(n==0)
      return 1;
   else
      return (n * fac(n-1));
}
```

```
public long fac (int n)
{
   answer = 1;
   for(int i=1; i<=n; i++)
      answer = answer * i;
   return answer;
}
```

루프로 구현한 것이 일반적으로 더 성능이 빠르다

재귀로 구현할 때 생각을 단순하게 할 수 있다.

특별한 경우꼬리재귀(tail-recursive) 재귀 → 루프로 쉽게 변환가능

피보나치 수 (Fibonacci Numbers)

- 토끼 개체 수의 증가를 모델링
  - 첫 달에는 새로 태어난 토끼 한쌍만 존재
  - 두 달 이상된 토끼는 번식 가능
  - 번식 가능한 토끼 한쌍은 매달 새끼 한쌍 낳음
  - 토끼는 죽지 않음
- n (n > 0) 번째 달에 a 쌍의 토끼,
  n+1 번째 달에 새로 태어난 토끼 포함, b 쌍의 토끼가 있었으면
  n+2 번째 달에는 a+b 쌍의 토끼가 있음 (왜?)
- 공식
  - fib(1) = fib(2) = 1
  - fib(n) = fib(n-1) + fib(n-2)

재귀로 푼 피보나치 수

```
public long fib (int n) {
{
   if( n==1 || n==2)
      return 1;                          // fib(1)=fib(2)=1
   else
      return (fib(n-1) + fib(n-2)); // fib(n)=fib(n-1)+fib(n-2)
}
```

반복으로 푼 피보나치 수

```
public long fib (int n) {
   // If the value of N is one or two return 1
    if( n == 1 || n == 2) return 1;
    // Keep track of the fibonacci values for N-1 and N-2
    int n_1 = 1;
    int n_2 = 1;

    // From the bottom-up calculate all the fibonacci values
    // until you reach the n-1 and n-2 values of the target fib(n)
    for(int i = 3; i < n; i++) {
       int temp = n_1;
       n_1 = n_2 + n_1;
       n_2 = temp;
    }
    return n_1 + n_2;
}
```

유클리드 호제법

- 최대 공약수(greatest common divisor) 계산 알고리즘
- 정수 a와 b 의 최대 공약수 (gcd(a, b))
  = 정수 b 와 a를 b로 나눈 나머지의 최대 공약수
  (gcd(b, a % b))
- 예: 1071 과 1029 의 최대 공약수
  - gcd(1071, 1029)
    = gcd(1029, 42)
    = gcd(42, 21)
    = gcd(21, 0) = 21

재귀로 푼 유클리드 호제법

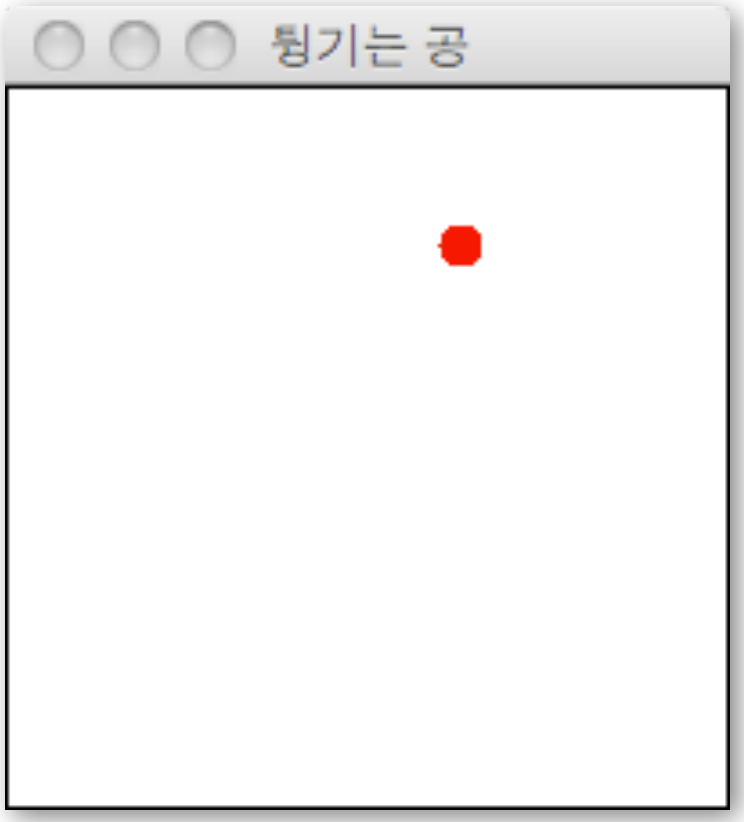
```
public int findGCD(int k, int m) {

    // base case
    if(m == 0) {
      return k;
    }

    return findGCD(m, k % m);
}
```

루프로 변환할 수 있을까?

반복으로 푼 유클리드 호제법

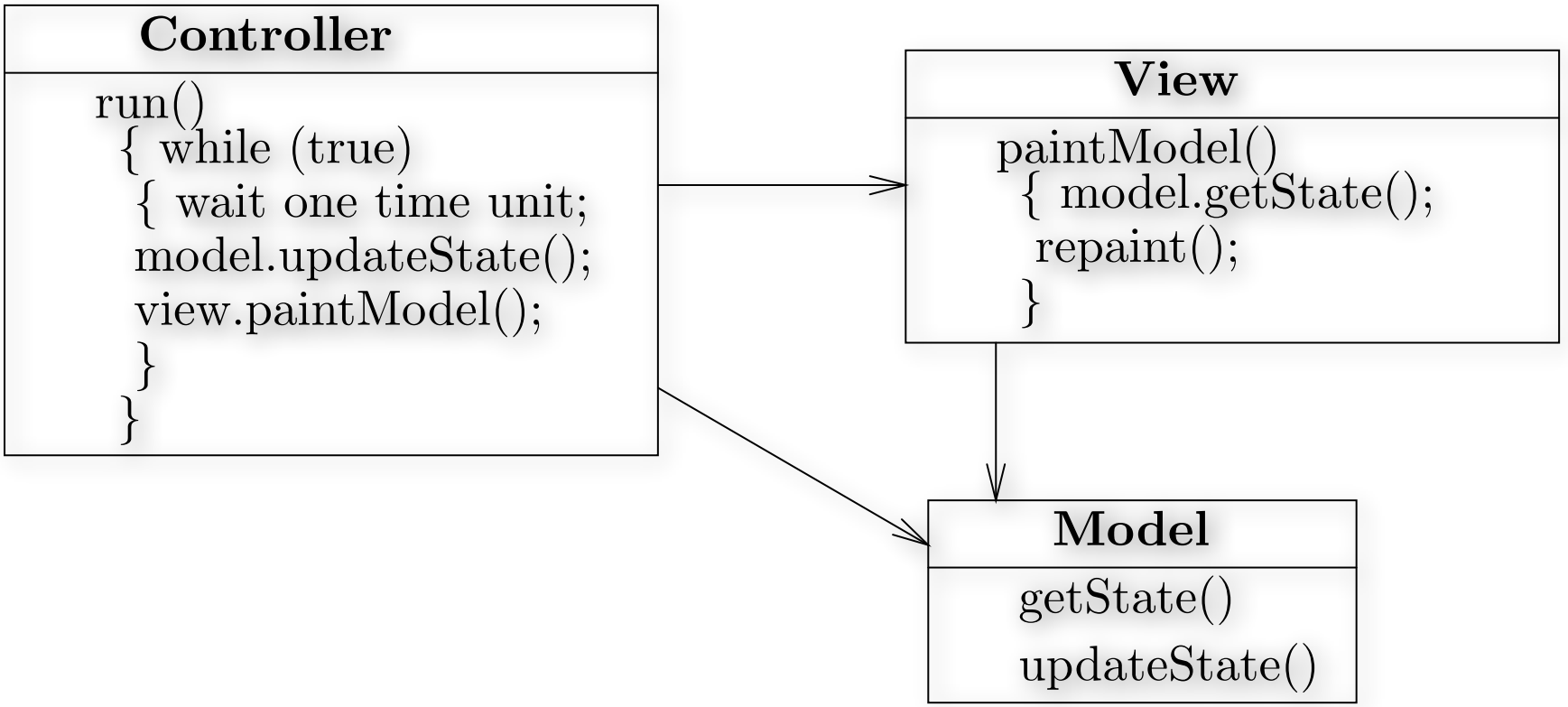
```
public int findGCD(int k, int m) {

      while (m != 0) {
         int r = k % m;
         k = m;
         m = r;
      }

      return k;
   }
```

설계 예제, 튕기는 공

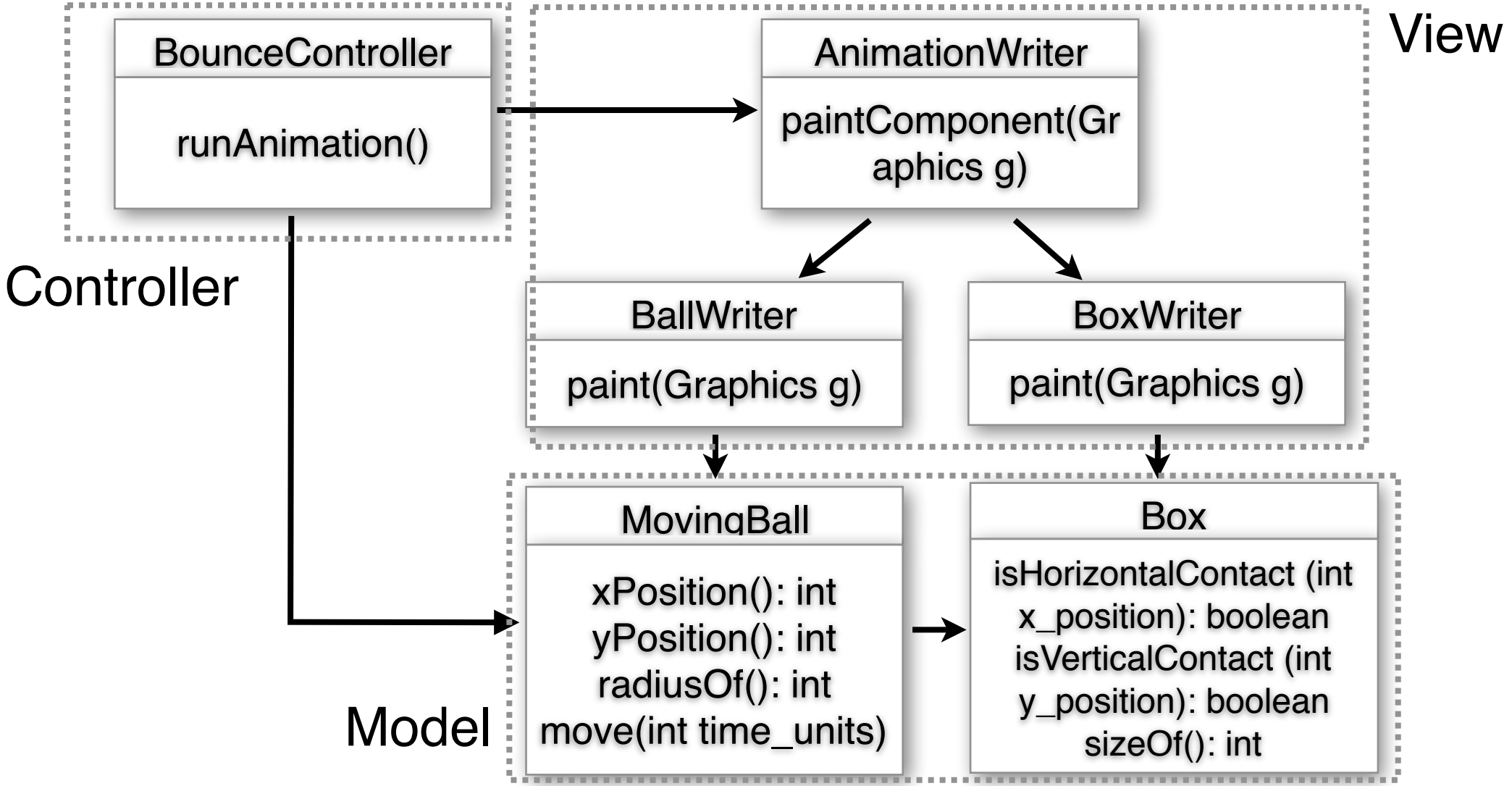


공이 상자안에서 튕기면서 움직이도록 하자.

움직이는 그림의 소프트웨어 구조



튕기는 공의 소프트웨어 구조



명세: class MovingBall (모델)

생성자	
`MovingBall(int x_initial, int y_initial, int r, Box box)`	해당 인수들로 객체를 초기화
속성	
`private int x_pos`	공의 x좌표
`private int y_pos`	공의 y좌표
`private int radius`	공의 반지름
`private int x velocity`	공의 수평 이동속도: 양수면 오른쪽으로 이동
`private int y velocity`	공의 수직 이동속도: 양수면 아래쪽으로 이동
`private Box container`	공이 그 안에서 움직이고 있는 테두리 박스
메소드	
`move(int time_units)`	`time_units`: 공이 마지막으로 움직인 이후 흐른 시간. 이 값과 현재 공의 위치, 속도에 따라 공의 위치를 갱신
`getState()`	공의 현재 위치, 속도, 크기 등 반환

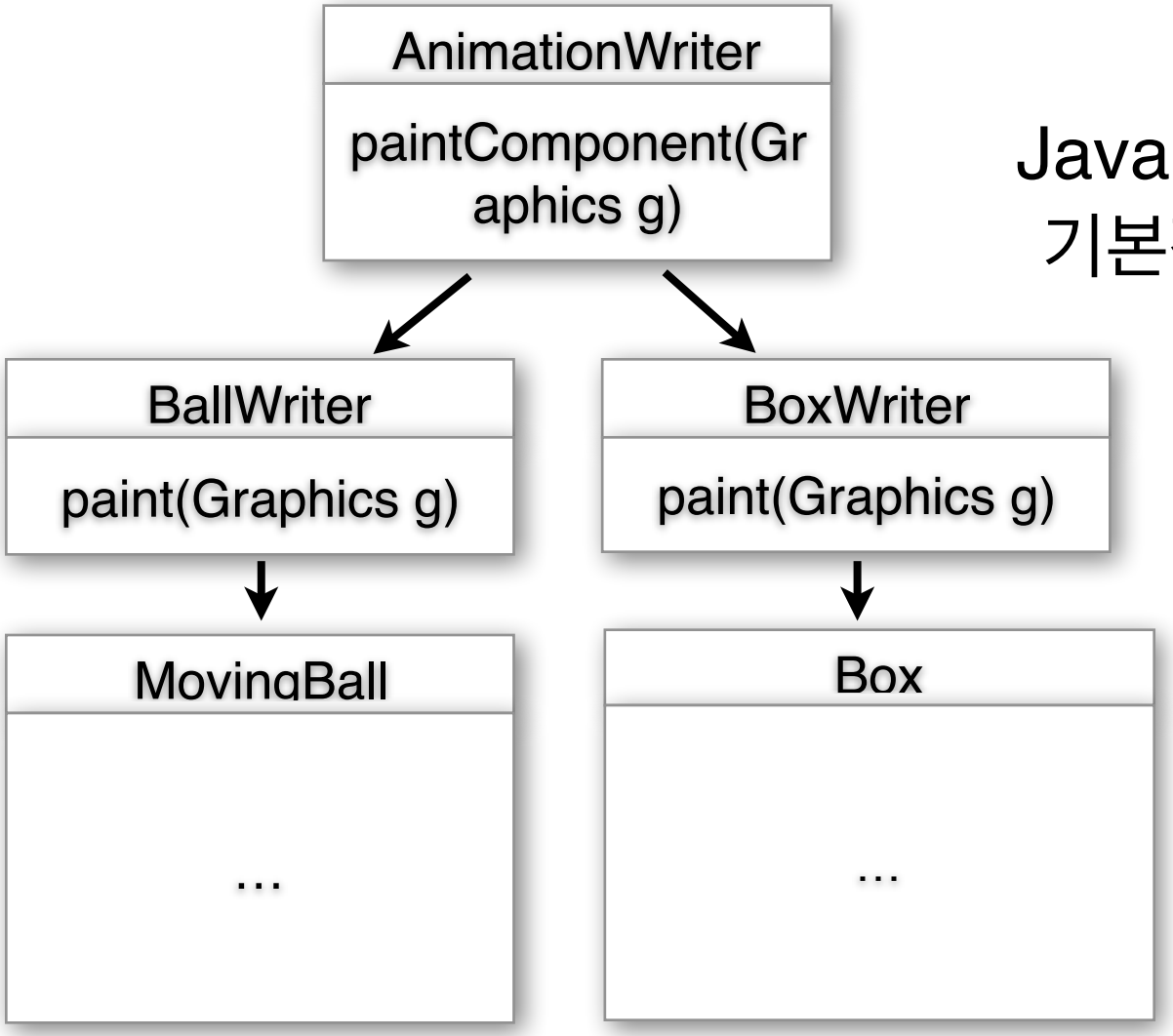
명세: class Box (모델)

`class Box`	models a square box
Attribute	
`int BOX_SIZE`	the width of the square box
Methods	
`inHorizontalContact(int x_position): boolean`	responds whether `x_position` (a horizontal coordinate) is in contact with one of the Box's (leftmost or rightmost) walls
`inVerticalContact(int y_position): boolean`	responds whether `y_position` (a vertical coordinate) is in contact with one of the Box's (upper or lower) walls

```
public class MovingBall
{
    private int x_pos, y_pos, radius;
    private int x_velocity = 5;
    private int y_velocity = 2;
    private Box container;

    public MovingBall(int x, int y, int r, Box box) {
        x_pos=x; y_pos=y; radius=r; container=box;
    }
    public int xPosition() { return x_pos; }
    public int yPosition() { return y_pos; }
    public int radiusOf() { return radius; }

    public void move(int time_units) {
        x_pos = x_pos + x_velocity * time_units;
        if(container.inHorizontalContact(x_pos))
            x_velocity = -x_velocity;
        y_pos = y_pos + y_velocity * time_units;
        if(container.inVerticalContact(y_pos))
            y_velocity = -y_velocity;
    }
}
```

```
public class Box {
    private int box_size;
    public Box(int size) { box_size = size; }
    public boolean inHorizontalContact(int x_position) {
        return x_position <=0 || x_position >= box_size;
    }
    public boolean inVerticalContact(int y_position) {
        return y_position <=0 || y_position >= box_size;
    }
    public int sizeOf() { return box_size; }
}
```

뷰 (View)



Java GUI 에 대한 기본적 이해 필요

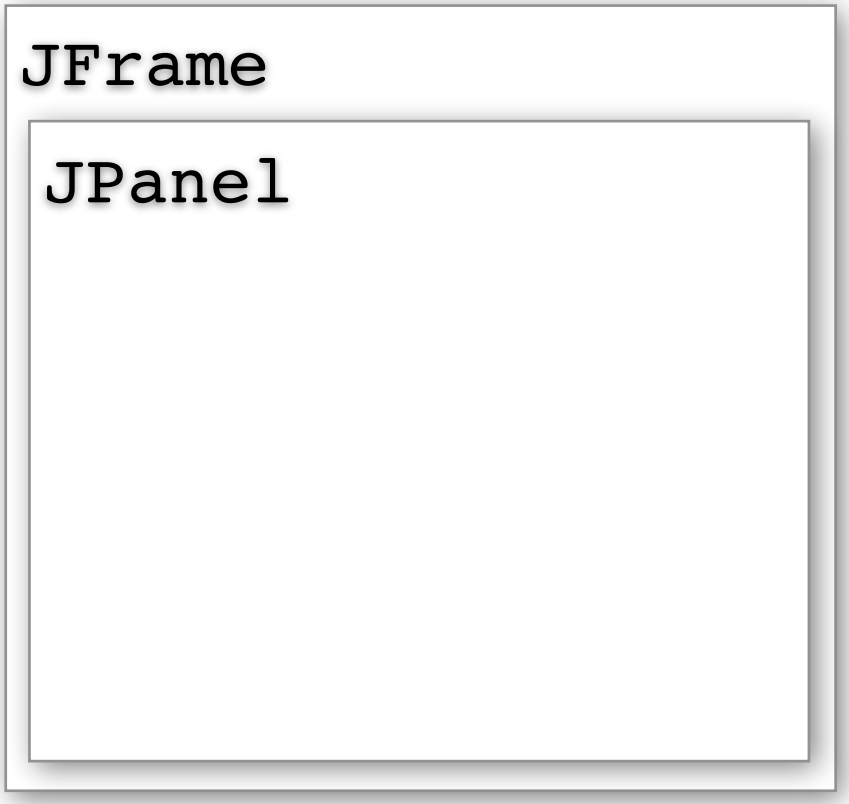
GUI: 그림 그리기

- 바닥부터 내가 만들면
  - 내 맘대로 할 수 있겠지만
  - 창을 그리기 위해 선도 긋고, 색칠도 하고
  - 할 일이 너무 많다!

- 라이브러리를 이용하자!
  - `javax.swing` 패키지
  - 남이 작성한 것이니까, 이 패키지에서 창을 그리는 방법을 이해해야 한다.

GUI: 프레임과 패널

- `javax.swing` 패키지에서 창은 `JFrame` 객체이다.
- 그 안에 `JPanel` 객체를 넣어서 다양한 그림을 그릴 수 있다.

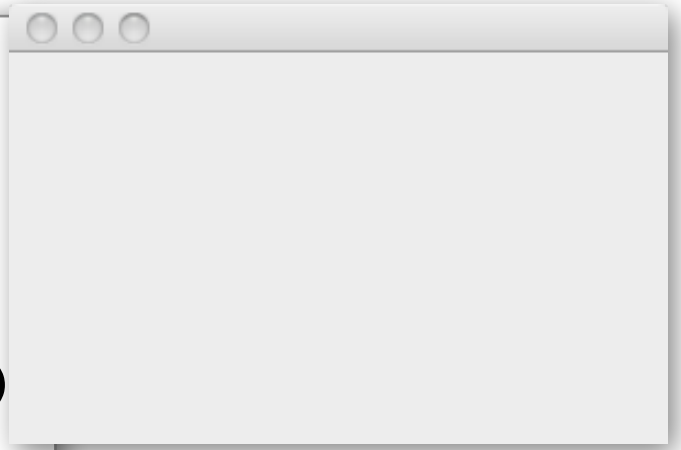


GUI: 프레임 만들기

- 프레임은 창의 바탕
- 만들기
  - `new JFrame()`
- 반드시 보여 주어야 한다.
  - `<프레임>.setVisible(true);`
- 크기 지정 가능
  - `<프레임>.setSize(<가로크기>,<세로크기>);`
  - 크기는 점의 수 (pixels)

GUI: 빈 프레임 만들어서 보이기

```
import javax.swing.*;
/* 빈 프레임 보이기  */

public class FrameTest2
{ public static void main(String[] args)
  { JFrame f = new JFrame();
    f.setSize(300, 200);
       // 크기: 가로, 세로 점 수
    f.setVisible(true);
       // 보여줘!
    }
}
```



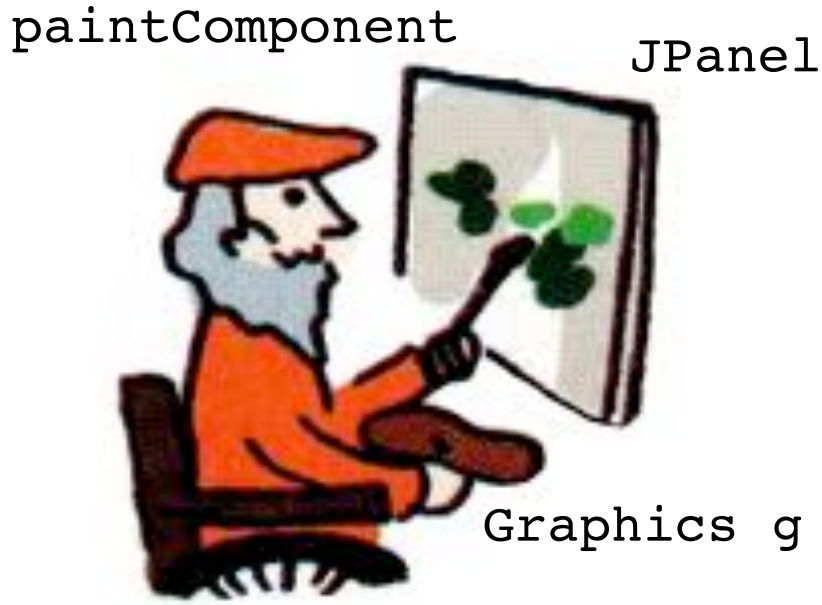
GUI: 패널 (JPanel)

- 패널은 그림을 그릴 수 있는 객체
- 패널 만들기
  - `new JPanel()`
- 패널은 프레임에 부착되어야 한다.
  - `<프레임>.getContentPane().add(<패널>);`

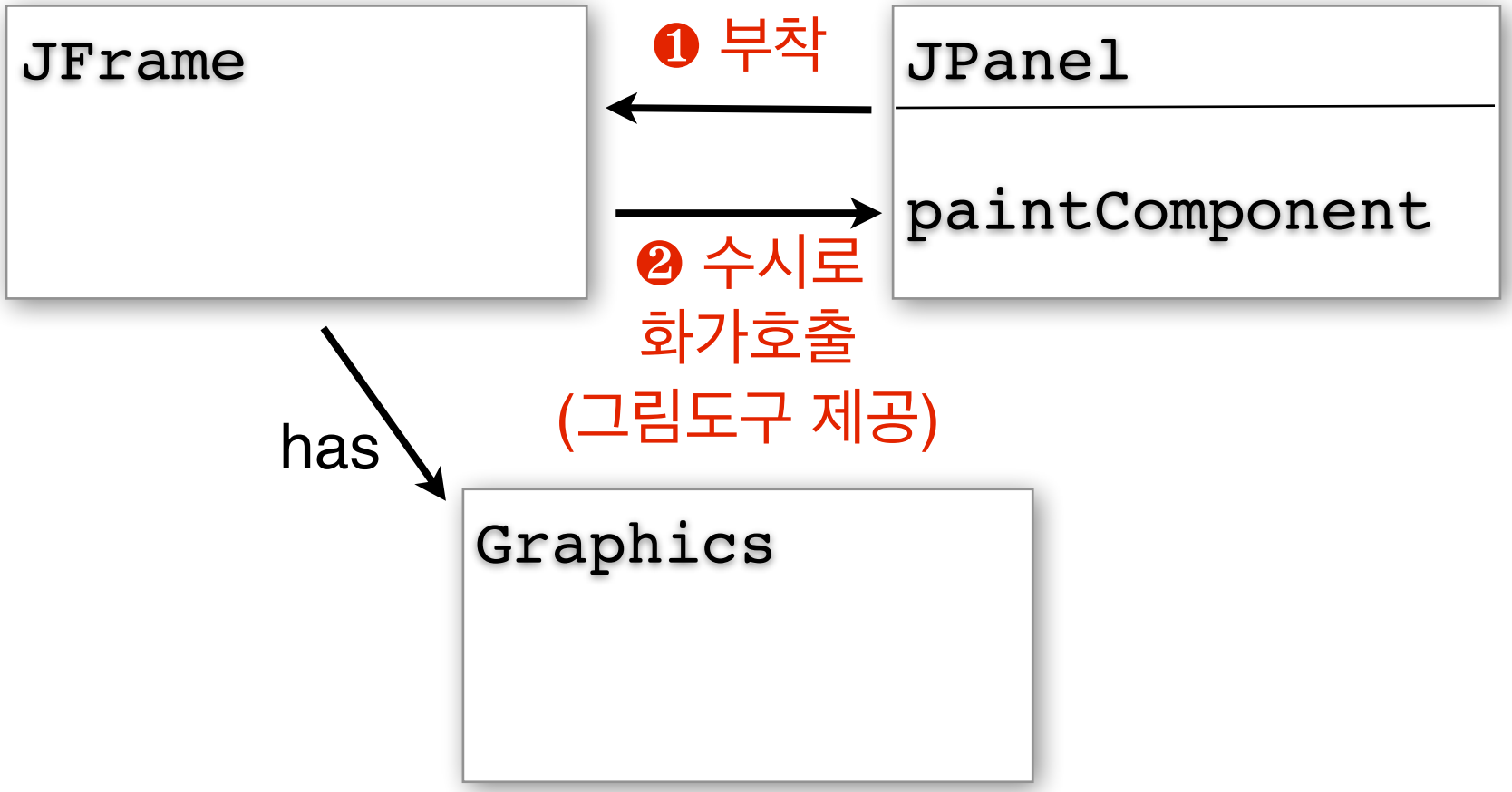
GUI: 패널과 화가

- 패널은 화가가 딸려 있다.
  - JPanel 은 메소드 paintComponent가 있다.
- 화가는 때때로 다시 그린다.
  - 필요할 때 창은 자기 패널의 paintComponent를 호출한다.
- 화가는 그림 도구가 필요하다.
  - 호출시 그림도구인 Graphics 객체를 준다.

JPanel
paintComponent(Graphics g)



GUI: 프레임, 패널, 화가의 구동 원리



GUI: 내가 원하는 패널을 만드려면

- 화가만 바꾸면 된다.
  - 즉, paintComponent 메소드만 교체해야 한다.
- 방법: 상속 (inheritance)과 메소드 재정의 (overriding)

JPanel
paintComponent

MyPanel
paintComponent

paintComponent

JPanel

paintComponent

MyPanel

GUI: 상속 (Inheritance)

- 상속: 틀을 그대로 가져다 쓰기
  - `public class A extends B { ... }`
  - B를 토대로 A를 만들어라!
  - B에 있는 모든 내용 (필드, 메소드) 포함

- 메소드 재정의 (overriding)
  - 상속받으면서 메소드를 정의하면 재정의된다!
  - 이 경우, 상위클래스(super class), 즉, 물려 주는 쪽의 메소드는 사용되지 않고, 하위 클래스(subclass)의 재정의된 메소드가 사용된다.

GUI: 내가 원하는 패널, 상속으로 만들기

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

public class MyPanel extends JPanel {
   public void paintComponent(Graphics g) { 그리기 코드 }
}
```

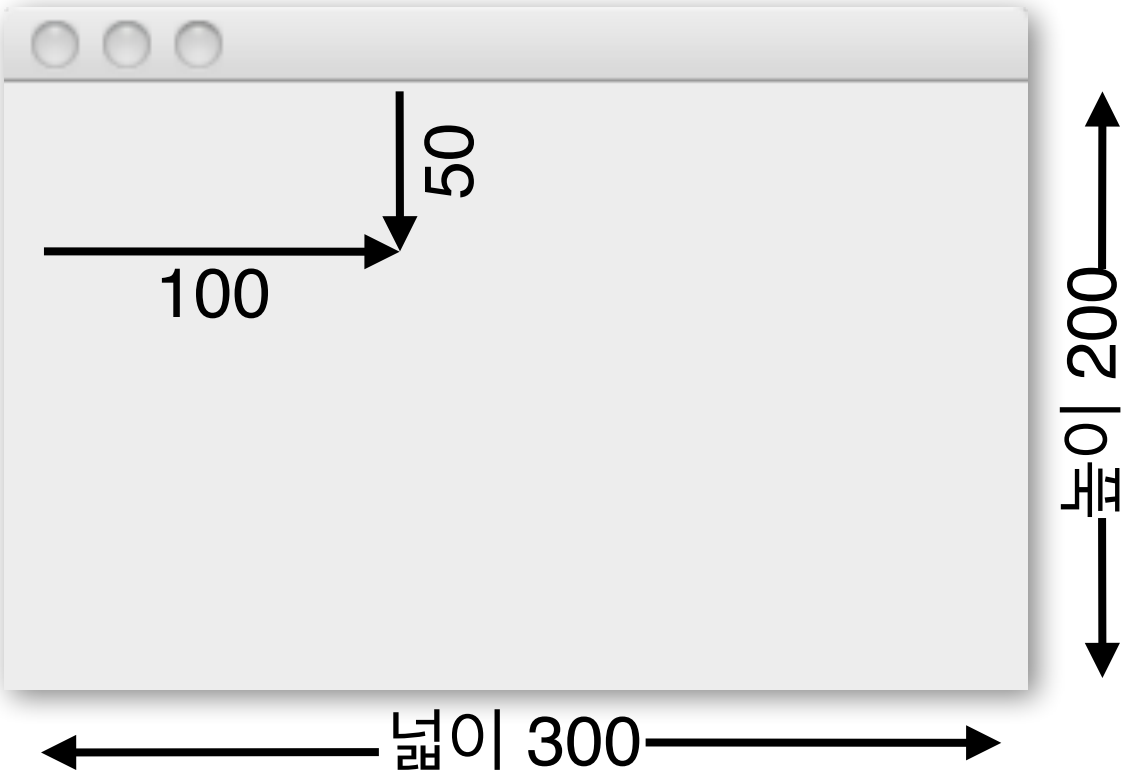
- 화가만 교체된다!
- 나머지는 기존 패널과 다를 게 없으므로 프레임에 그대로 넣어서 사용 가능!

GUI: 화가의 도구 Graphics

- Graphics 는 다양한 도구를 지원한다.
  - setColor(c)
  - drawLine(x1,y1,x2,y2)
  - drawString(s,x,y)
  - drawRect/fillRect/drawOval/
    fillOval(x,y,dx,dy)
  - drawArc/fillArc(x,y,dx,dy,a,da)
  - paintImage(i,x,y,ob)
  - 등등
- 몇 가지만 살펴보고 나머지는 설명서 참조

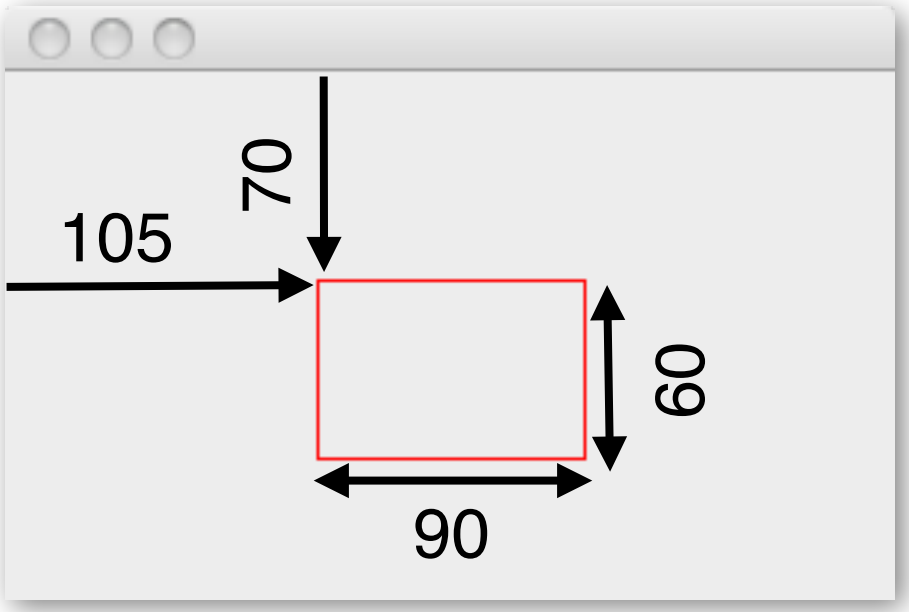
GUI: 위치 및 크기의 단위: 점 수 (Pixels)

- 크기가 (300,200) 일 때 (100,50) 위치는



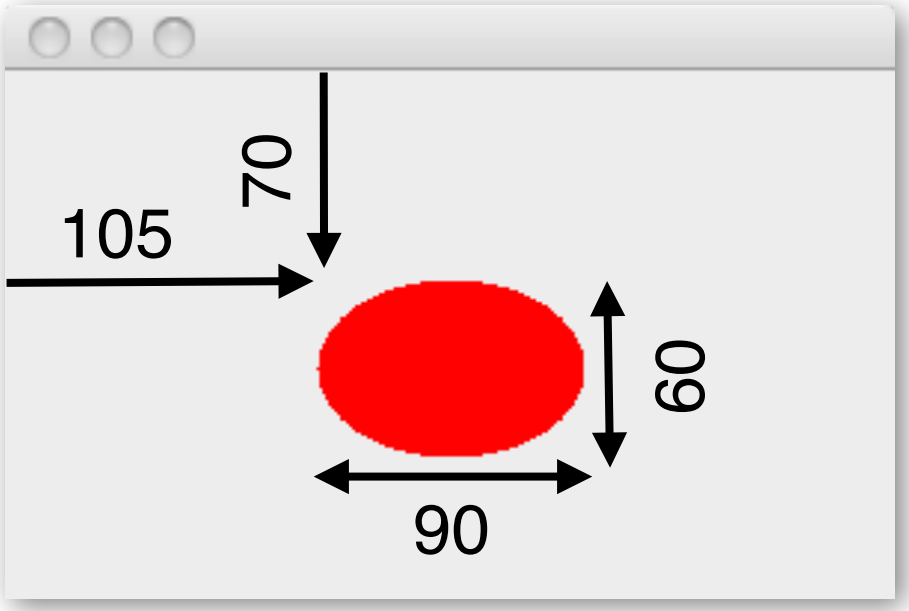
GUI: 사각형 그리기

- `g.drawRect(105, 70, 90, 60)`



GUI: 타원 그리기

- `g.fillOval(105, 70, 90, 60)`



```
public class AnimationWriter extends JPanel
{ private BoxWriter box_writer;    // the output-view of the box
  private BallWriter ball_writer;  // the output-view of the ball in the box

  /** Constructor AnimationWriter constructs the view of box and ball
    * @param b - the box's writer
    * @param l - the ball's writer
    * @param size - the frame's size */
  public AnimationWriter(BoxWriter b, BallWriter l, int size)
  { box_writer = b;
    ball_writer = l;
    JFrame my_frame = new JFrame();
    my_frame.getContentPane().add(this);
    my_frame.setTitle("Bounce");
    my_frame.setSize(size, size);
    my_frame.setVisible(true);
  }

  /** paintComponent paints the box and ball
    * @param g - the graphics pen */
  public void paintComponent(Graphics g)
  { box_writer.paint(g);
    ball_writer.paint(g);
  }
}
```

```
public class BallWriter
{ private MovingBall ball;   // the (address of the) ball object displayed
  private Color balls_color; // the ball's color

  /** Constructor BallWriter
    * @param x - the ball to be displayed
    * @param c - its color */
  public BallWriter(MovingBall x, Color c)
  { ball = x;
    balls_color = c;
  }

  /** paint paints the ball on the view
    * @param g - the graphics pen used to paint the ball */
  public void paint(Graphics g)
  { g.setColor(balls_color);
    int radius = ball.radiusOf();
    g.fillOval(ball.xPosition() - radius,
               ball.yPosition() - radius, radius * 2, radius * 2);
  }
}
```

```
public class BoxWriter
{ private Box box;   // the (address of the) box object that is
displayed

  /** Constructor BoxWriter displays the box
    * @param b - the box that is displayed */
  public BoxWriter(Box b)
  { box = b; }

  /** paint paints the box
    * @param g - the graphics pen used to paint the box */
  public void paint(Graphics g)
  { int size = box.sizeOf();
    g.setColor(Color.white);
    g.fillRect(0, 0, size, size);
    g.setColor(Color.black);
    g.drawRect(0, 0, size, size);
  }
}
```

제어기 (Controller)

- 다음을 영원히 반복:
  - 공에게 1 시간단위(time unit) 만큼 움직이라고 지시
  - 뷰에게 현재 공의 현재 상태를 다시 그리라고 지시

```
public class BounceController {
    private MovingBall ball; // model object
    private AnimationWriter writer; // output-view object
    public BounceController(MovingBall b, AnimationWriter w)
    { ball = b; writer=w; }

    public void runAnimation() {
        int time_unit = 1;
        int painting_delay = 20;
        while (true)
        {
            delay(painting_delay);
            ball.move(time_unit);
            writer.repaint();
        }
    }
    private void delay(int how_long) {
        try { Thread.sleep(how_long); } catch (InterruptedException e) { }
    }
}
```

요약

- 루프 구문
  - while (질문?) { 몸체 }
  - for (초기화; 질문; 변경) { 몸체 }
- 루프 불변식
- 재귀
  - 현재 문제 크기보다 작은 문제에 대해서 답이 있다고 가정하고 푼다.