제어 구조 II.
반복

Control Structure : Iteration

반복 구조

동일 코드블록의 반복 실행을 표현하는 구조

대표적인 유형 3가지

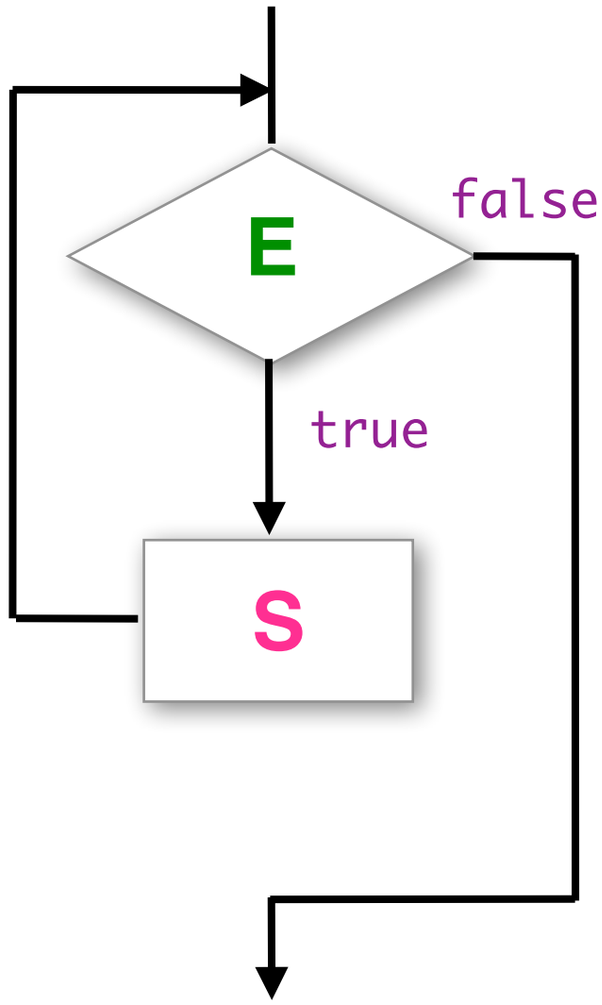
- 반복 횟수 고정
- 반복 횟수 사전 예측 불가
- 무한 반복 diverge

while 루프

<논리식>

```
while ( E )
    S
```

<코드블록>



반복 횟수 고정

패턴

```
int n = GOAL_VALUE;
int count = 0;
while (count < n) {
    // 코드
    count += 1;
}
```

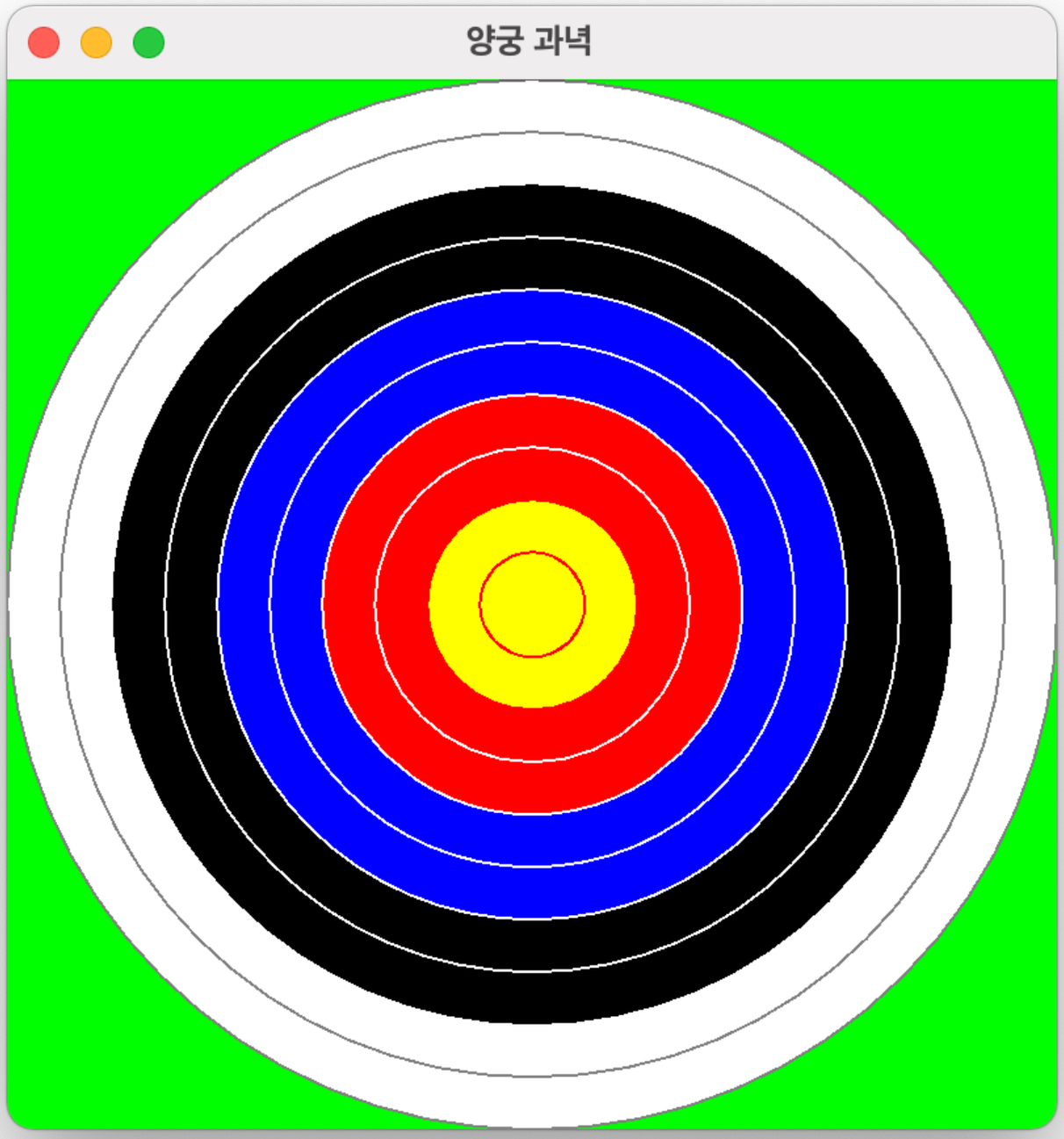
반복 횟수 고정

사례 - 시험 점수 평균 구하기

```
import javax.swing.*;

public class CourseManagement {

    public double calculateAverage(int n) {
        double sum = 0.0;
        int count = 0;
        while (count < n) {
            String input = JOptionPane.showInputDialog("시험 점수?");
            int score = Integer.parseInt(input);
            sum += score;
            count += 1;
            // loop invariant : sum == count개 점수의 합
        }
        return sum / count; // n == count
    }

    public static void main(String[] args) {
        CourseManagement course_mgmt = new CourseManagement();
        String message = "평균 점수 = " + course_mgmt.calculateAverage(5);
        JOptionPane.showMessageDialog(null, message);
    }

}
```

반복 횟수 고정

사례 - 양궁 과녁 그리기



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

public class Archery extends JPanel {

    private final int RINGS = 10; // 원의 개수
    private final int TARGET_DIAMETER;

    public Archery(int d) {
        TARGET_DIAMETER = d;
        JFrame f = new JFrame();
        f.getContentPane().add(this);
        f.setTitle("양궁 과녁");
        f.setSize(TARGET_DIAMETER, TARGET_DIAMETER + 28);
        f.setVisible(true);
        f.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }

    public void paintComponent(Graphics g) {
        g.setColor(Color.green);
        g.fillRect(0,  0, TARGET_DIAMETER, TARGET_DIAMETER);
        final int OFFSET = TARGET_DIAMETER / RINGS;
        int number = 1;
        int diameter = TARGET_DIAMETER;
        int new_x_position = 0;
        int new_y_position = 0;
        while (number <= RINGS) {
            // loop invariant (루프 불변 성질): 지금까지 number-1 개의 링을 그렸음
            if (number <= 2) {
                g.setColor(Color.white);
                g.fillOval(new_x_position, new_y_position, diameter, diameter);
                g.setColor(Color.gray);
                g.drawOval(new_x_position, new_y_position, diameter, diameter);
            }
            else if (number <= 4) {
                g.setColor(Color.black);
                g.fillOval(new_x_position, new_y_position, diameter, diameter);
```

```
            else if (number <= 4) {
                g.setColor(Color.black);
                g.fillOval(new_x_position, new_y_position, diameter, diameter);
                if (number == 4) {
                    g.setColor(Color.white);
                    g.drawOval(new_x_position, new_y_position, diameter, diameter);
                }
            }
            else if (number <= 6) {
                g.setColor(Color.blue);
                g.fillOval(new_x_position, new_y_position, diameter, diameter);
                g.setColor(Color.white);
                g.drawOval(new_x_position, new_y_position, diameter, diameter);
            }
            else if (number <= 8) {
                g.setColor(Color.red);
                g.fillOval(new_x_position, new_y_position, diameter, diameter);
                g.setColor(Color.white);
                g.drawOval(new_x_position, new_y_position, diameter, diameter);
            }
            else if (number <= 10) {
                g.setColor(Color.yellow);
                g.fillOval(new_x_position, new_y_position, diameter, diameter);
                g.setColor(Color.red);
                g.drawOval(new_x_position, new_y_position, diameter, diameter);
            }
            new_x_position = OFFSET * number / 2;
            new_y_position = OFFSET * number / 2;
            number += 1;
            diameter = diameter - OFFSET;
        }
    }

    public static void main(String[] args) {
        new Archery(400);
    }

}
```

반복 횟수 사전 예측 불가

입력 값에 종속

```
boolean processing = true;
while (processing) {
    // 입력
    if (/* 종료 신호 수신 */)
        processing = false;
    else {
        // 코드
    }
}
```

반복 횟수 사전 예측 불가

사례 - 시험 점수 평균 구하기

```
import javax.swing.*;

public class CourseManagement {

    public double calculateAverage() {
        double sum = 0.0;
        int count = 0;
        boolean processing = true;
        while (processing) {
            // loop invariant : sum == count개 점수의 합
            String message = "다음 시험 점수? (입력 완료시 Cancel 버튼 누름)";
            String input = JOptionPane.showInputDialog(message);
            if (input == null) // Cancel 버튼을 눌렀음
                processing = false;
            else {
                int score = Integer.parseInt(input);
                sum += score;
                count += 1;
            }
        }
        if (count == 0) return 0;
        else return sum / count;
    }

    public static void main(String[] args) {
        CourseManagement course_mgmt = new CourseManagement();
        String message = "평균 점수 = " + course_mgmt.calculateAverage();
        JOptionPane.showMessageDialog(null, message);
    }
```

반복 횟수 사전 예측 불가

검색 결과에 종속

```
int index = 0;
boolean found = false;
while (!found && index < s.length()) {
    if (s.charAt(index) == c)
        found = true;
    else
        index = index + 1;
}
```

반복 횟수 사전 예측 불가

문자열 검색

```
public class TextSearch {

    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;
            // loop invariant:
            // (1) found == false : s[0], .., s[index-1]은 모두 c가 아님
            // (2) found == true : s.charAt(index) == c
        }
        if (!found)
            index = -1;
        return index;
    }

    public static void main(String[] args) {
        TextSearch text_search = new TextSearch();
        System.out.println(text_search.findChar('a', "Hanyang"));
        System.out.println(text_search.findChar('e', "Hanyang"));
    }
}
```

for 루프

```
int count = 0;
while (count < n) {
    // 코드
    count += 1;
}
```

```
for (int count = 0; count < n; count += 1) {
    // 코드
}
```

반복 횟수 사전 예측 불가

문자열 검색

```
public class TextSearch {

    public int findChar(char c, String s) {
        int index;
        for (index = 0; index < s.length() && s.charAt(index) != c; index++)
            // loop invariant: s[0], .., s[index-1]은 모두 c가 아님
            ;
        if (index == s.length())
            index = -1;
        return index;
    }

    public static void main(String[] args) {
        TextSearch text_search = new TextSearch();
        System.out.println(text_search.findChar('a', "Hanyang"));
        System.out.println(text_search.findChar('e', "Hanyang"));
    }
}
```

while vs. for

```
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;
        // loop invariant:
        // (1) found == false : s[0], .., s[index-1]은 모두 c가 아님
        // (2) found == true : s.charAt(index) == c

    }
    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++)
        // loop invariant: s[0], .., s[index-1]은 모두 c가 아님
        ;
    if (index == s.length())
        index = -1;
    return index;
}
```

중첩 루프

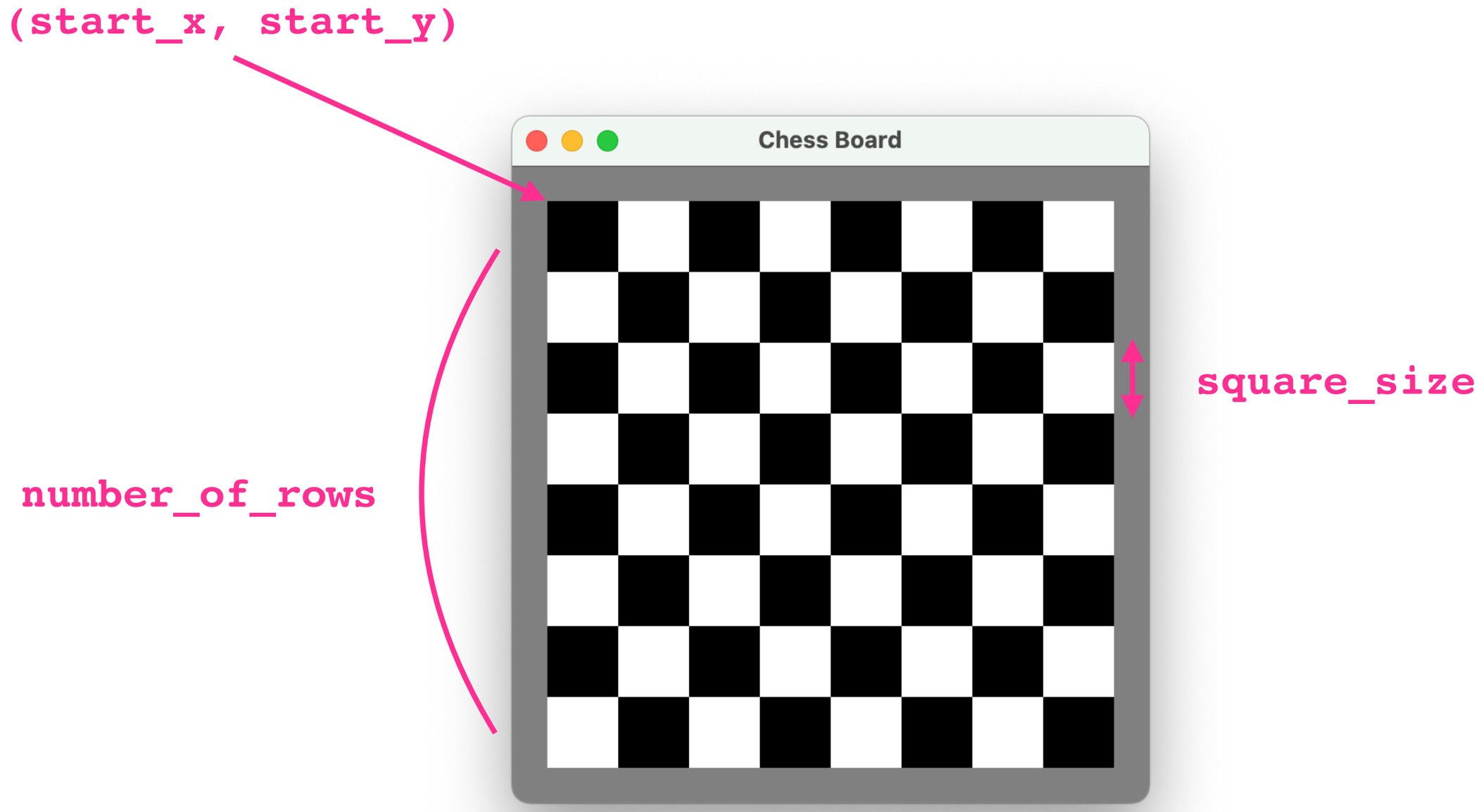
Nested Loop

```
for (int i = 1; i < 10; i++) {
    // loop invariant: 구구단의 i-1 단까지 출력했음
    for (int j = 1; j < 10; j++) {
        // loop invariant: 구구단의 i-1 단까지 출력하고, i단의 j항까지 출력했음
        System.out.print(i + "x" + j + "=" + (i * j) + " ");
    }
    System.out.println();
}
```

```
1x1=1 1x2=2 1x3=3 1x4=4 1x5=5 1x6=6 1x7=7 1x8=8 1x9=9
2x1=2 2x2=4 2x3=6 2x4=8 2x5=10 2x6=12 2x7=14 2x8=16 2x9=18
3x1=3 3x2=6 3x3=9 3x4=12 3x5=15 3x6=18 3x7=21 3x8=24 3x9=27
4x1=4 4x2=8 4x3=12 4x4=16 4x5=20 4x6=24 4x7=28 4x8=32 4x9=36
5x1=5 5x2=10 5x3=15 5x4=20 5x5=25 5x6=30 5x7=35 5x8=40 5x9=45
6x1=6 6x2=12 6x3=18 6x4=24 6x5=30 6x6=36 6x7=42 6x8=48 6x9=54
7x1=7 7x2=14 7x3=21 7x4=28 7x5=35 7x6=42 7x7=49 7x8=56 7x9=63
8x1=8 8x2=16 8x3=24 8x4=32 8x5=40 8x6=48 8x7=56 8x8=64 8x9=72
9x1=9 9x2=18 9x3=27 9x4=36 9x5=45 9x6=54 9x7=63 9x8=72 9x9=81
```

중첩 루프

체스 보드 그리기



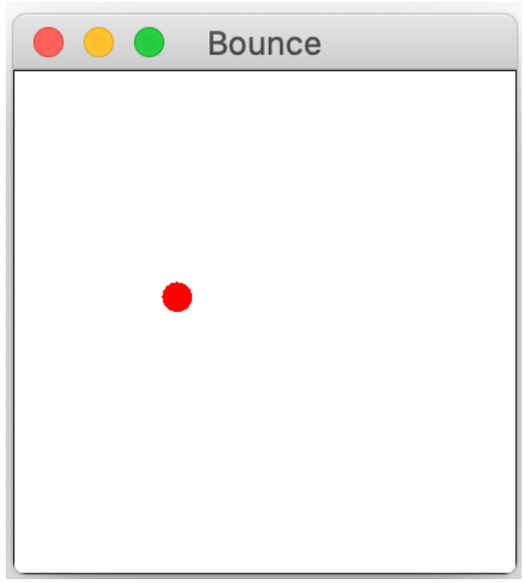

```java
public static void main(String[] args) {
    new ChessBoardWriter(8, 40);
}
```

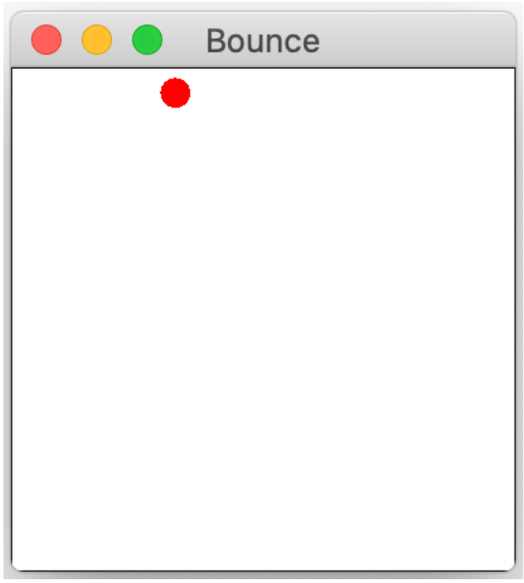
체스 보드 그리기

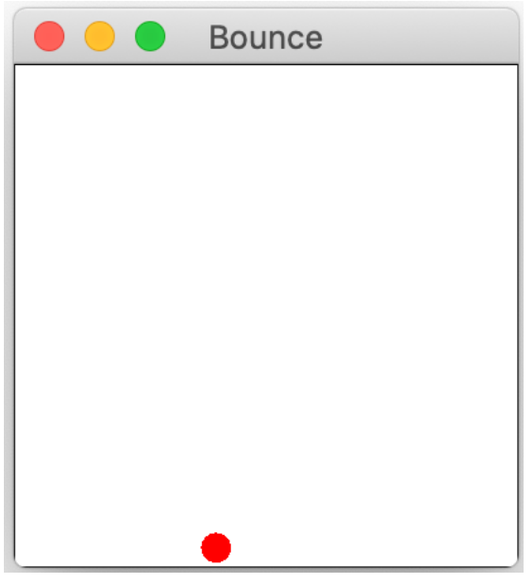
```
import javax.swing.*;
import java.awt.*;

public class ChessBoardWriter extends JPanel {
    private int number_of_rows;
    private int square_size;
    private int panel_width;
    private int offset = 20;

    /* Constructor ChessBoardWriter - 패널을 만들고 프레임을 씌움
     * @param rows - 각 열별 칸의 갯수
     * @param size - 한 칸의 길이  */
    public ChessBoardWriter(int rows, int size) {
        number_of_rows = rows;
        square_size = size;
        panel_width = number_of_rows * square_size + 2 * offset;

        JFrame f = new JFrame();
        f.getContentPane().add(this);
        f.setTitle("Chess Board");
        f.setSize(panel_width, panel_width + 28);
        f.setVisible(true);
        f.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }

    /* paintComponent - 패널에 그림을 그림
     * @param g - 그래픽스 펜 */
    public void paintComponent(Graphics g) {
        g.setColor(Color.gray);
        g.fillRect(0,  0, panel_width, panel_width);
        paintBoard(offset, offset, number_of_rows, square_size, g);
```

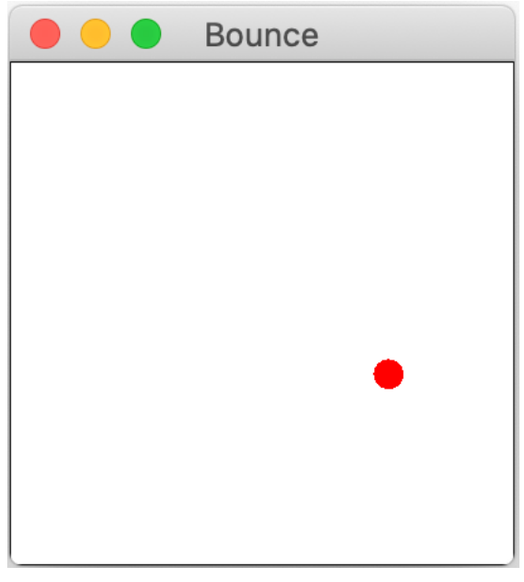
```
    /* paintComponent - 패널에 그림을 그림
     * @param g - 그래픽스 펜 */
    public void paintComponent(Graphics g) {
        g.setColor(Color.gray);
        g.fillRect(0,  0, panel_width, panel_width);
        paintBoard(offset, offset, number_of_rows, square_size, g);
    }

    /* paintBoard - 체스보드를 그림
     * @param start_x - 체스보드의 좌상단 구석의 x 좌표
     * @param start_y - 체스보드의 좌상단 구석의 y 좌표
     * @param rows - 체스보드 열의 갯수
     * @param size - 체스보드 칸의 너비
     * @param g - 그래픽스 펜 */
    private void paintBoard(int start_x, int start_y,
                            int rows, int size, Graphics g) {
        for (int x = 0; x < rows; x = x + 1) {
            // loop invariant: x열까지 그렸음 (x 증가 전)
            int x_position = start_x + x * size;
            for (int y = 0; y < rows; y = y + 1) {
                // loop invariant: x열의 y칸까지 그렸음 (x 증가 후, y 증가 전)
                int y_position = start_y + y * size;
                if ((x + y) % 2 == 0) // 빨간색 칠할 차례
                    g.setColor(Color.black);
                else
                    g.setColor(Color.white);
                g.fillRect(x_position, y_position, size, size);
            }
        }
    }

    public static void main(String[] args) {
        new ChessBoardWriter(8, 40);
    }
}
```

사례 학습 - 상자 속 공 굴리기 애니메이션

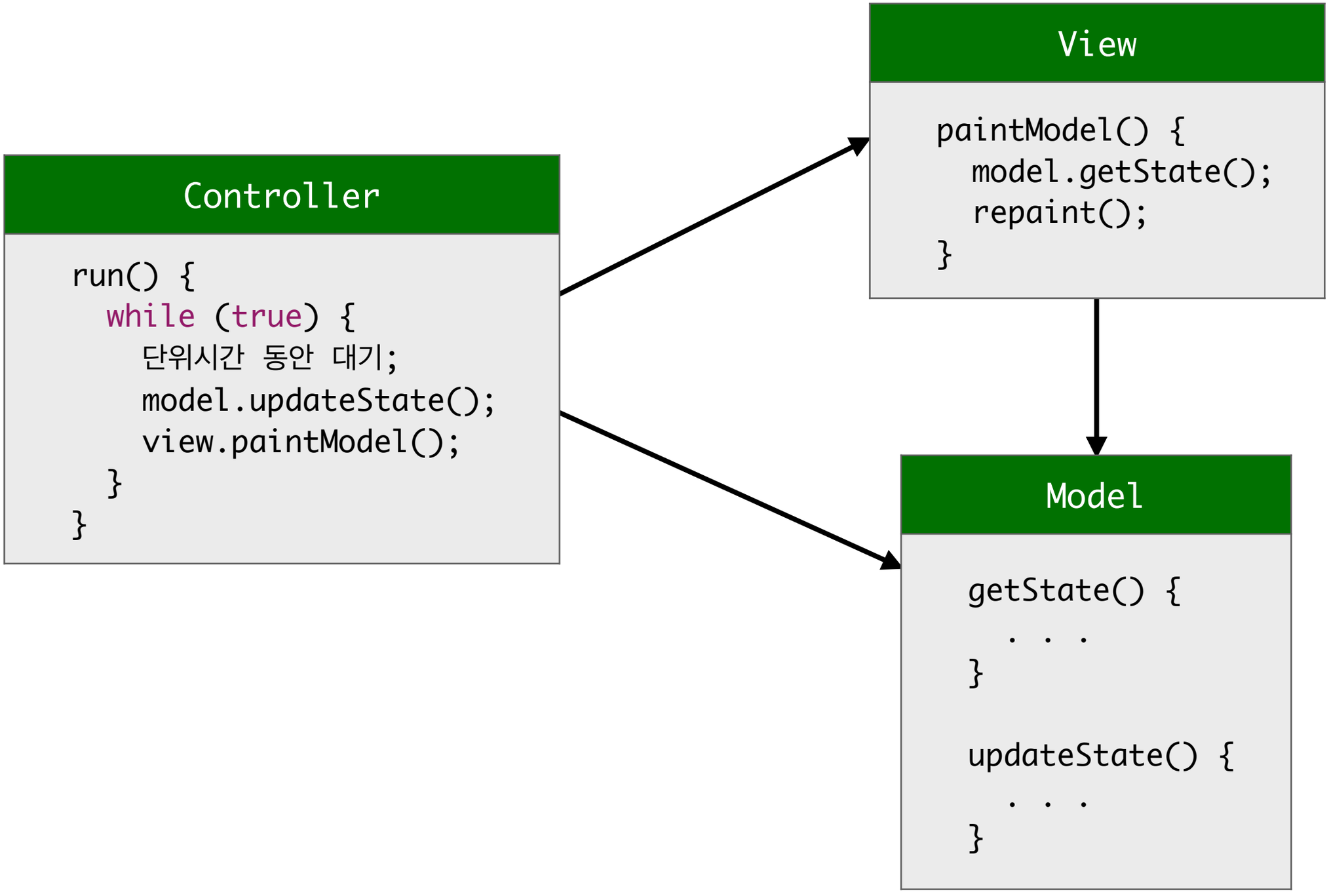




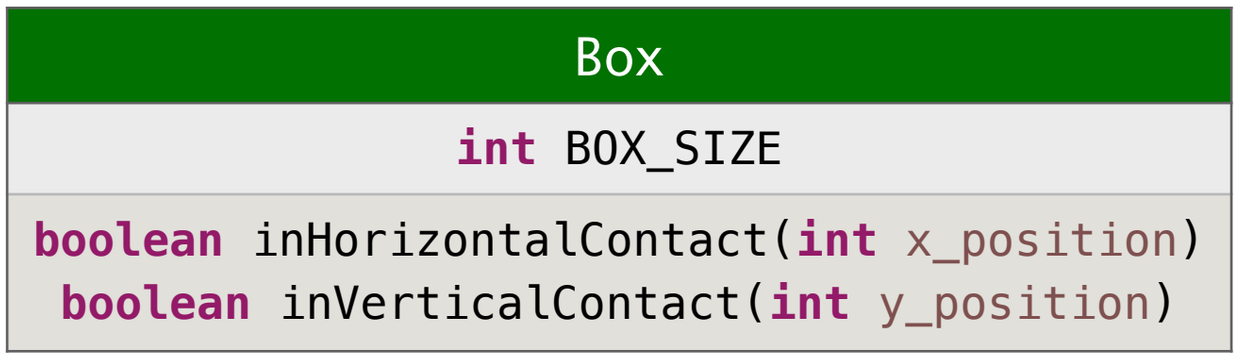


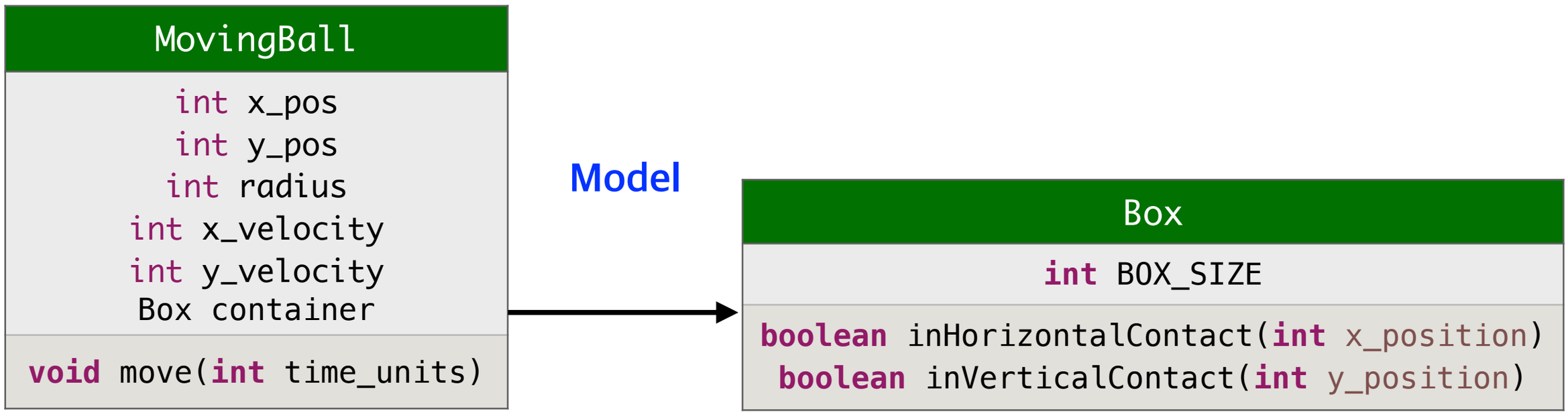


애니메이션 애플리케이션의 아키텍처 일반 패턴



Model


Box
int BOX_SIZE
boolean inHorizontalContact(int x_position)
boolean inVerticalContact(int y_position)


MovingBall
int x_pos
int y_pos
int radius
int x_velocity
int y_velocity
Box container
void move(int time_units)
Model
Box
int BOX_SIZE
boolean inHorizontalContact(int x_position)
boolean inVerticalContact(int y_position)

Model

class	Box	공이 돌아다니는 상자
field	`int BOX_SIZE`	상자의 크기
method	`boolean inHorizontalContact(int x_position)`	공이 x축 방향으로 좌/우 벽에 도달 여부를 리턴
	`boolean inVerticalContact(int y_position)`	공이 y축 방향으로 아래/위 벽에 도달 여부를 리턴

class	MovingBall	2차원 상자에서 움직이는 공
field	`int x_pos, int y_pos`	공의 중심 x 좌표, 공의 중심 y 좌표
	`int radius`	공의 반지름
	`int x_velocity,` `int y_velocity`	속도 x축, 속도 y축
	`Box container`	돌아다닐 상자
method	`void move(int time_units)`	time_unit 만큼 공을 이동, 벽에 부딪치면 방향을 바꿈

```java
/** Box - 공이 돌아다니는 상자 */
public class Box {

    private int BOX_SIZE;  // 상자의 크기

    /** Constructor Box - 상자 생성
     * @param size - 상자의 크기 */
    public Box(int size) {
        BOX_SIZE = size;
    }

    /** inHorizontalContact - 공이 x축 방향으로 좌/우 벽에 도달 여부를 리턴
     * @param x_position - 공의 x 좌표
     * @return true, 공의 x 좌표가 좌우 벽의 x 좌표와 같거나 벗어났으면 true, 그렇지 않으면 false */
    public boolean inHorizontalContact(int x_position) {
        return (x_position <= 0 ) || (x_position >= BOX_SIZE);
    }

    /** inVerticalContact - 공이 y축 방향으로 아래/위 벽에 도달 여부를 리턴
     * @param y_position - 공의 y 좌표
     * @return true, 공의 y 좌표가 아래위 벽의 y 좌표와 같거나 벗어났으면 true, 그렇지 않으면 false */
    public boolean inVerticalContact(int y_position) {
        return (y_position <= 0 ) || (y_position >= BOX_SIZE);
    }

    /** sizeOf - 상자의 크기를 리턴 */
    public int sizeOf() {
        return BOX_SIZE;
    }
}
```

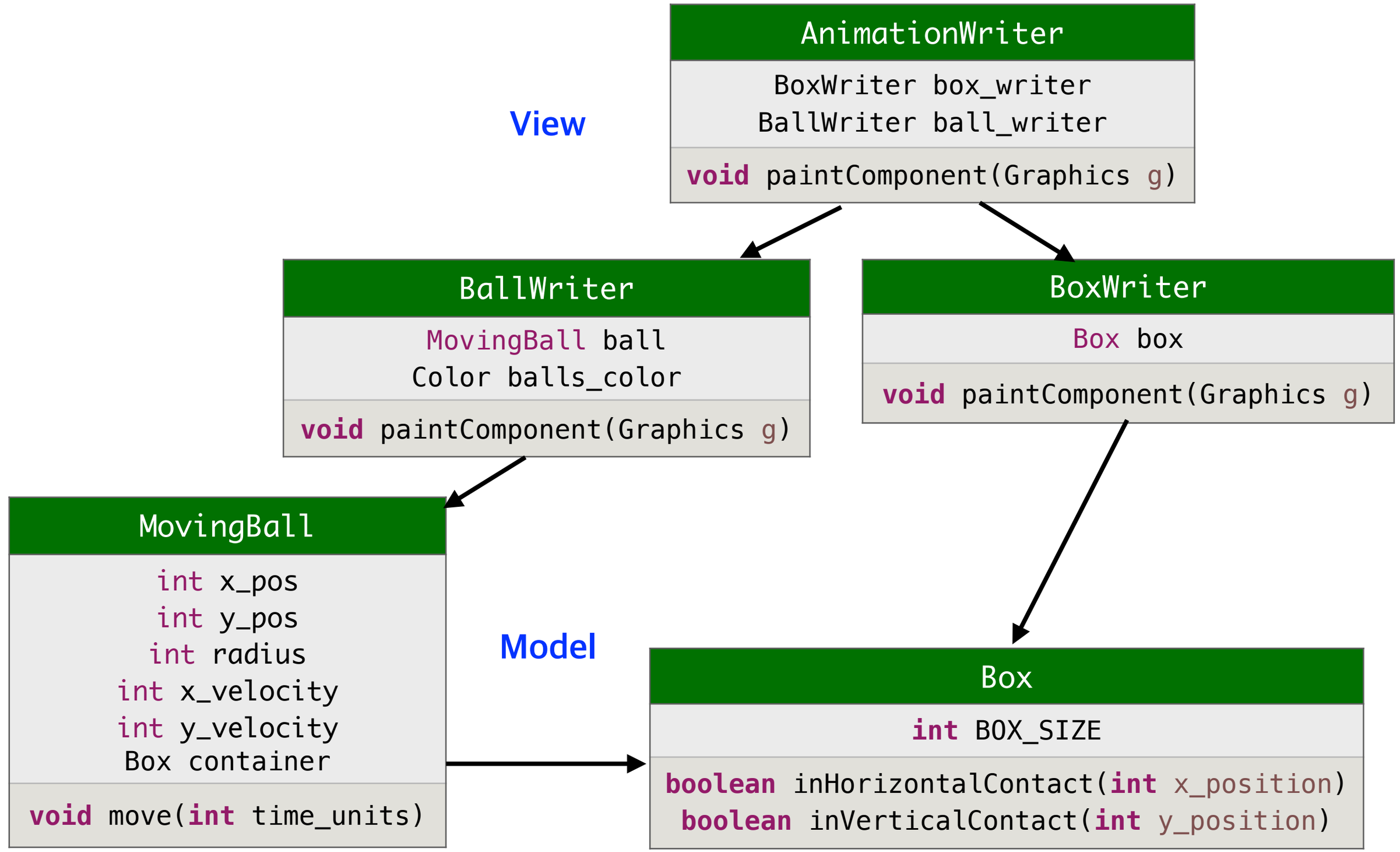
```java
/** MovingBall - 2차원 상자에서 움직이는 공 */
public class MovingBall {
    private int x_pos; // 공의 중심 x 좌표
    private int y_pos; // 공의 중심 y 좌표
    private int radius; // 공의 반지름

    private int x_velocity = +5; // 속도 x축
    private int y_velocity = +2; // 속도 y축

    private Box container;

    /** Contructor MovingBall - 공 만들기
     * @param x_initial - 공의 중심 x 좌표
     * @param y_initial - 공의 중심 y 좌표
     * @param r - 공의 반지름
     * @param box - 공이 살고 있는 상자 */
    public MovingBall(int x_initial, int y_initial, int r, Box box) {
        x_pos = x_initial;
        x_pos = y_initial;
        radius = r;
        container = box;
    }

    /** xPosition - 공의 x축 위치 리턴 */
    public int xPosition() {
        return x_pos;
    }

    /** yPosition - 공의 y축 위치 리턴 */
    public int yPosition() {
        return y_pos;
    }
```

```
    /** radiusOf - 공의 반지름 리턴 */
    public int radiusOf() {
        return radius;
    }

    /** move - time_unit 만큼 공을 이동, 벽에 부딪치면 방향을 바꿈
     * @param time_units - 프레임 사이의 시간 */
    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 TestModel {

    public static void main(String[] args) {
        Box box = new Box(50);
        MovingBall ball = new MovingBall(25, 25, 10, box);
        for (int i = 0; i < 10; i++) {
            ball.move(1);
            System.out.print("x = " + ball.xPosition());
            System.out.println(", y = " + ball.yPosition());
        }
    }
}
```

```
public void paintComponent(Graphics g) {
    box_writer.paintComponent(g);
    ball_writer.paintComponent(g);
}
```

View



Model

```java
import java.awt.*;

/** BoxWriter - 상자를 그림  */
public class BoxWriter {

    private Box box;    // 상자 객체

    /** Constructor BoxWriter
     * @param b - 상자 객체 */
    public BoxWriter(Box b) {
        box = b;
    }

    /** paint - 상자 그리기
     * @param g - 그래픽스 펜 */
    public void paintComponent(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);
    }
}
```

```java
import java.awt.*;

/** BallWriter - 움직이는 공을 그림 */
public class BallWriter {

    private MovingBall ball;   // 공 객체
    private Color balls_color; // 공의 색깔

    /** Constructor BallWriter
     * @param x - 공 객체
     * @param c - 공의 색깔 */
    public BallWriter(MovingBall x, Color c) {
        ball = x;
        balls_color = c;
    }

    /** paint - 공 그리기
     * @param g - 그래픽스 펜  */
    public void paintComponent(Graphics g) {
        g.setColor(balls_color);
        int radius = ball.radiusOf();
        g.fillOval(ball.xPosition() - radius, ball.yPosition() - radius,
                   radius * 2, radius * 2);
    }
}
```

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

/** AnimationWriter - 상자와 공의 애니메이션 디스플레이 */
public class AnimationWriter extends JPanel {
    private BoxWriter box_writer;    // 상자 그리는 객체
    private BallWriter ball_writer;  // 공 그리는 객체

    /** Constructor AnimationWriter - 상자와 공을 그리는 View 객체를 생성
     * @param b - 상자 그리는 객체
     * @param l - 공 그리는 객체
     * @param size - 프레임의 크기 */
    public AnimationWriter(BoxWriter b, BallWriter l, int size) {
        box_writer = b;
        ball_writer = l;
        JFrame f = new JFrame();
        f.getContentPane().add(this);
        f.setTitle("Bounce");
        f.setSize(size, size+22);
        f.setVisible(true);
        f.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }

    /** paintComponent - 공과 상자 그리기
     * @param g - 그래픽스 펜 */
    public void paintComponent(Graphics g) {
        box_writer.paintComponent(g);
        ball_writer.paintComponent(g);
    }
}
```

BounceTheBall

```
static void main(String[] args)
```

```
public void paintComponent(Graphics g) {
    box_writer.paintComponent(g);
    ball_writer.paintComponent(g);
}
```

BounceController

```
MovingBall ball
AnimationWriter writer
```

```
void runAnimation()
```

AnimationWriter

```
BoxWriter box_writer
BallWriter ball_writer
```

```
void paintComponent(Graphics g)
```

BallWriter

```
MovingBall ball
Color balls_color
```

```
void paintComponent(Graphics g)
```

BoxWriter

```
Box box
```

```
void paintComponent(Graphics g)
```

MovingBall

```
int x_pos
int y_pos
int radius
int x_velocity
int y_velocity
Box container
```

```
void move(int time_units)
```

Box

```
int BOX_SIZE
```

```
boolean inHorizontalContact(int x_position)
boolean inVerticalContact(int y_position)
```

```java
/** BounceController - 상자 안에서 움직이는 공 제어 */
public class BounceController {
    private MovingBall ball;  // 공 객체 (Model)
    private AnimationWriter writer; // 애니메이션 객체 (Output-View)

    /** Constructor BounceController 컨트롤러 초기화
     * @param b - 공 객체 (Model)
     * @param w - 애니메이션 객체 (Output-View)  */
    public BounceController(MovingBall b, AnimationWriter w) {
        ball = b;
        writer = w;
    }

    /** runAnimation - 내부 시계를 활용하여 애니메이션 구동 */
    public void runAnimation() {
        int time_unit = 1;     // 애니메이션 스텝의 시간 단위
        int painting_delay = 20;  // 다시 그리기 사이의 지연 시간 간격
        while (true) {
            delay(painting_delay);
              ball.move(time_unit);
              System.out.println(ball.xPosition() + ", " + ball.yPosition());
              writer.repaint();
          }
    }

    /** delay - how_long millisecond 동안 실행 정지  */
    private void delay(int how_long) {
        try { Thread.sleep(how_long); }
        catch (InterruptedException e) { }
    }
}
```

```java
import java.awt.*;

/** BounceTheBall - 애니메이션 객체를 생성하고 공 운동 시작 */
public class BounceTheBall {
    public static void main(String[] args) {
        // 모델 객체 생성
        int box_size = 200;
        int balls_radius = 6;
        Box box = new Box(box_size);
        // 공을 상자의 적절한 위치에 둠
        MovingBall ball = new MovingBall((int)(box_size / 3.0),
                                         (int)(box_size / 5.0),
                                         balls_radius, box);
        BallWriter ball_writer = new BallWriter(ball, Color.red);
        BoxWriter box_writer  = new BoxWriter(box);
        AnimationWriter writer = new AnimationWriter(box_writer, ball_writer, box_size);
        // 컨트롤러 객체를 생성하고 애니메이션 시작
        new BounceController(ball, writer).runAnimation();
    }
}
```

실습

#1. 파란 공을 하나 추가

- 두 공은 다른 장소에서 다른 방향으로 출발한다.
- 두 공이 움직이는 속도는 같다.
- 공이 충돌해도 그대로 통과한다.

#2. 충돌시 진로 수정

- 두 공이 충돌하면, 둘 다 진행 방향을 역방향으로 바꾼다.

#3. 장애물 설치

- 중앙에 다음과 같은 모양의 적당한 길이의 장애물을 설치한다.

- 공이 이 장애물 위면 또는 아래 면을 만나면 y축 진행 방향을 바꾸도록 한다.