**Lab 4**

**1분반**

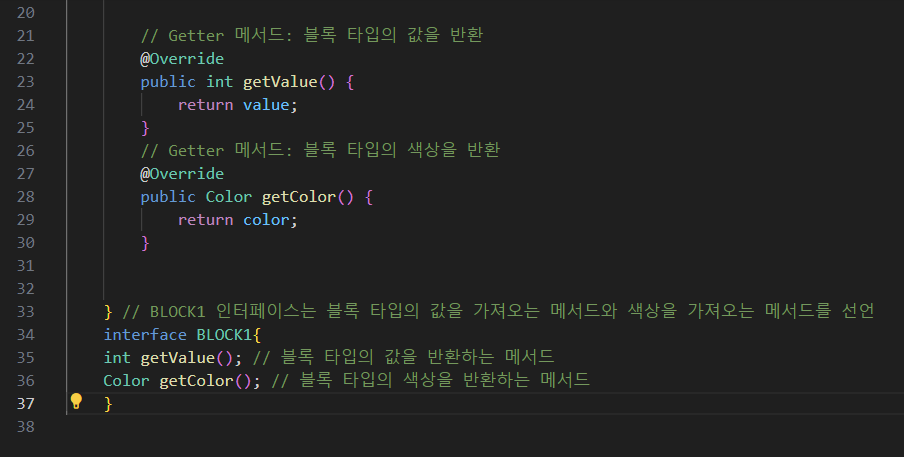
**2024년 05월 21일**

**32211792**

**박재홍**

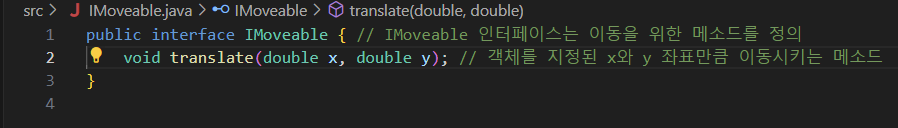
**BlockType 열거형)**

****

****

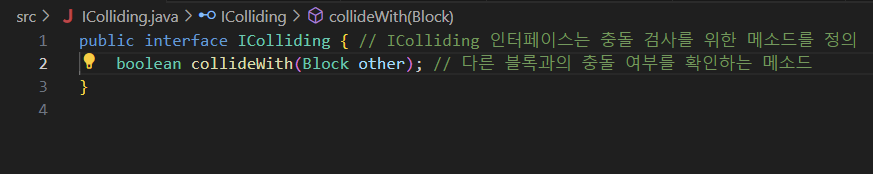
**BlockType을 열거형으로 만들어야 하기에 enum으로 만든 후 뒤에 Block추상 클래스가 있어 BLOCK1이라는 이름으로 인터페이스를 만들어 그 메소드를 상속받게 해주었고 RED\_BLOCK, TEAL\_BLOCK, DOGERS\_BLUE\_BLOCK, YELLOW\_BLOCK, ORANGE\_BLOCK, PURPLE\_BLOCK, GREEN\_BLOCK 들의 각 블록의 타입 상수와 색상을 정의해주었다. 그 후 블록 타입의 값을 나타내는 변수인 Value, 블록 타입의 색상을 나타내는 변수인 Color를 정의 해주었다. 그 후 각 블록 타입의 상수들을 초기화 시켜주는 생성자를 구현하였고, 블록 타입의 값과 색상들을 반환해주는 Getter 메소드를 써주었다. 그리고 블록타입의 값과 색상을 가져오는 메소드를 선언하는 BLOCK1 이라는 인터페이스를 구현하였다.**

**IMoveable 인터페이스)**

****

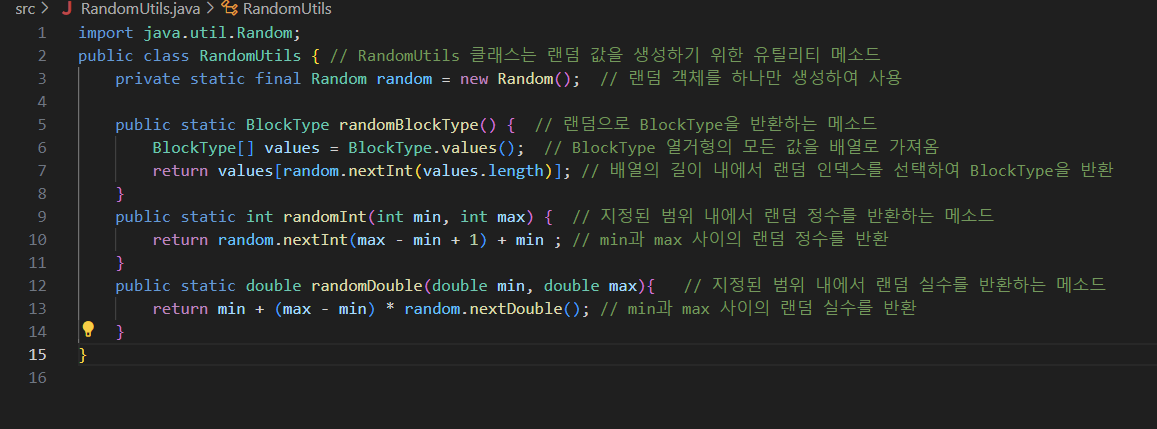
**블록의 이동을 위한 IMoveable 인터페이스를 정의 하였고 translate를 사용해 지정된 x와 y좌표 만큼 이동시키는 메소드를 정의해 구현해주었다.**

**IColliding 인터페이스)**

****

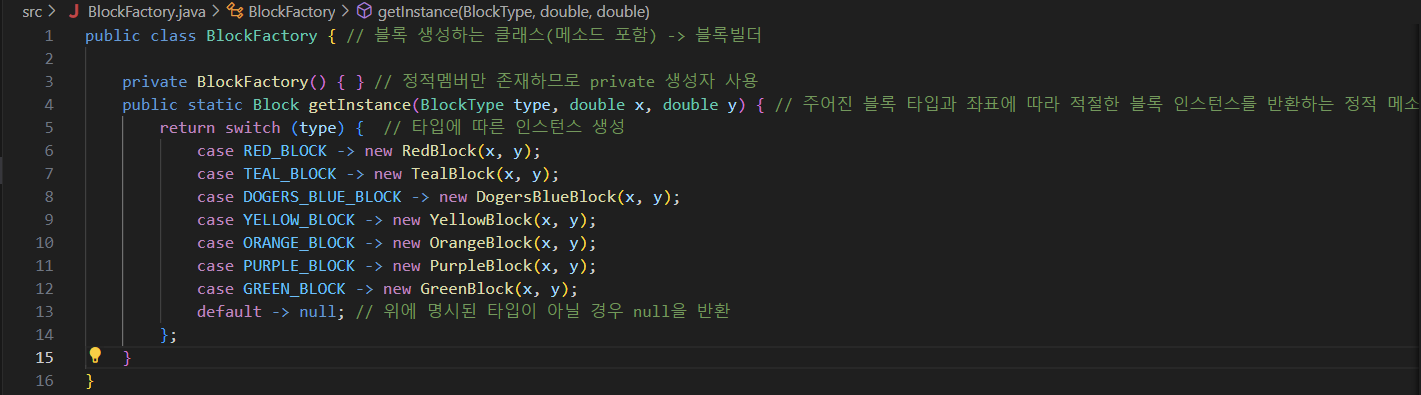
**블록의 충돌을 확인하기 위한 IColliding 인터페이스를 정의 하였고 그 안에 다른 블록과의 충돌 여부를 확인하기 위해 boolean을 사용해서 구현해주었다.**

**RandomUtils 클래스)**

****

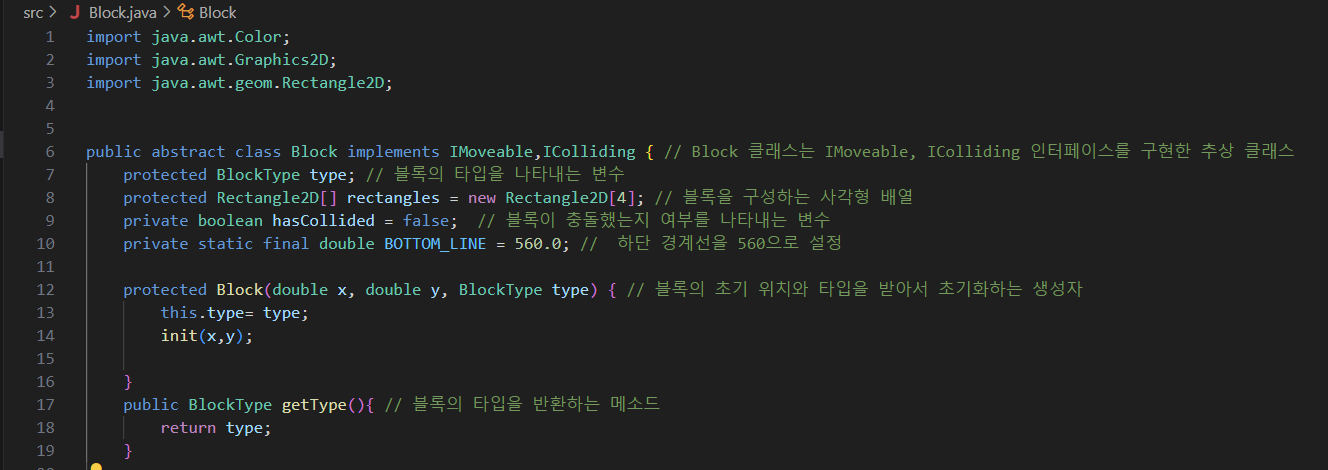
**RandomUtils 클래스는 랜덤값을 생성하기위한 메소드이고, 우선 랜덤값을 받기위해 import java.util.Random 을 선언해주었고, 랜덤값을 받기 위해 private static final Random random = new Random(); 을 써서 랜덤값을 받는 객체를 하나 만들어주었고, public static BlockType randomBlockType() 를 선언해 랜덤으로 BlockType을 반환할 수 있게 해주었고 BlockType[] = values = BlockType.values() 를 사용해 BlockType의 값들을 배열로 가져올 수 있게 해주었다. 그 후 배열의 길이 내에서 랜덤 인덱스를 선택해서 BlockType의 값을 반환 할 수 있게 해주었다. 그 다음 max와 min 사이의 랜덤 정수를 반환해주는 메소드를 정의 해주었다. 마지막으로는, max와 min 사이의 랜덤 실수를 반환해주는 메소드를 정의해주었다.**

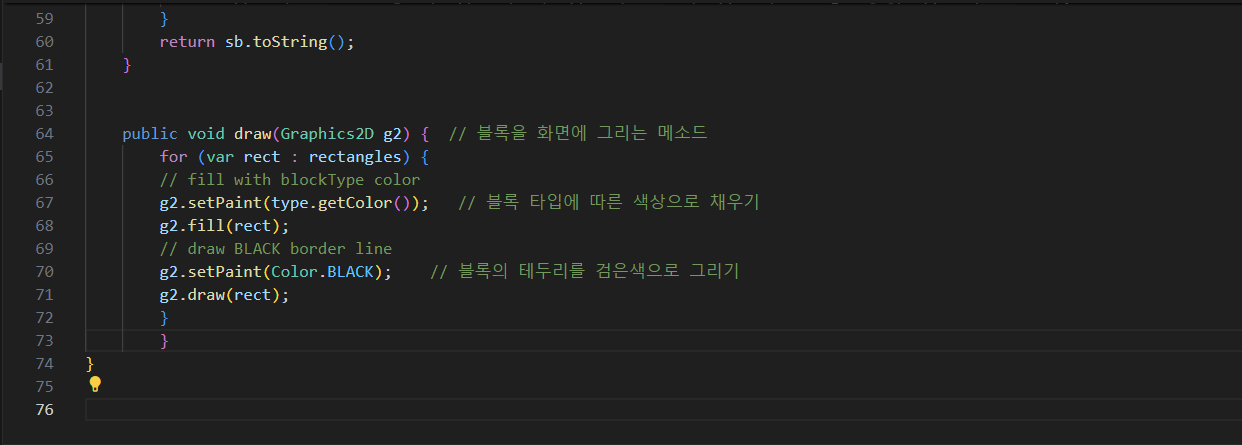
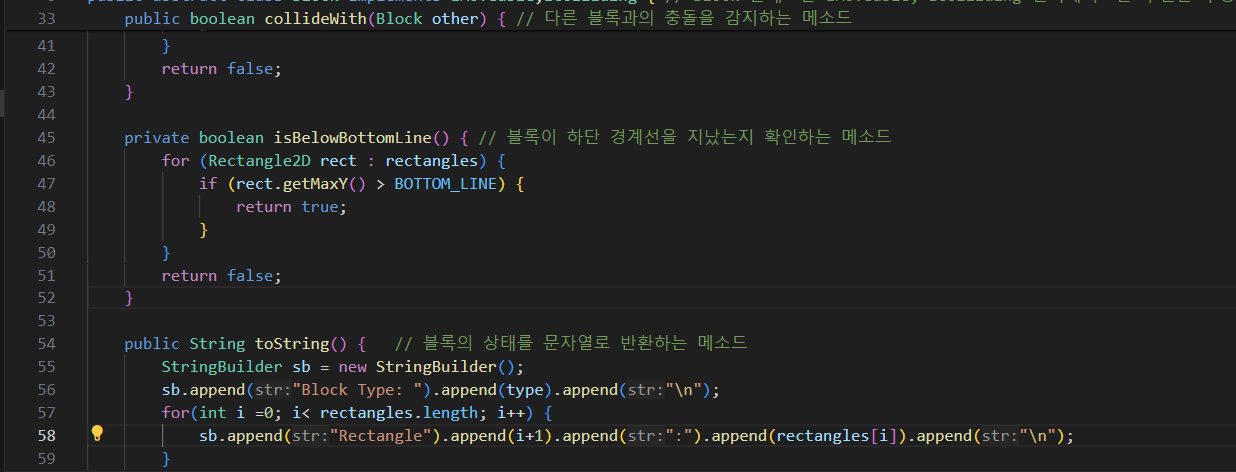
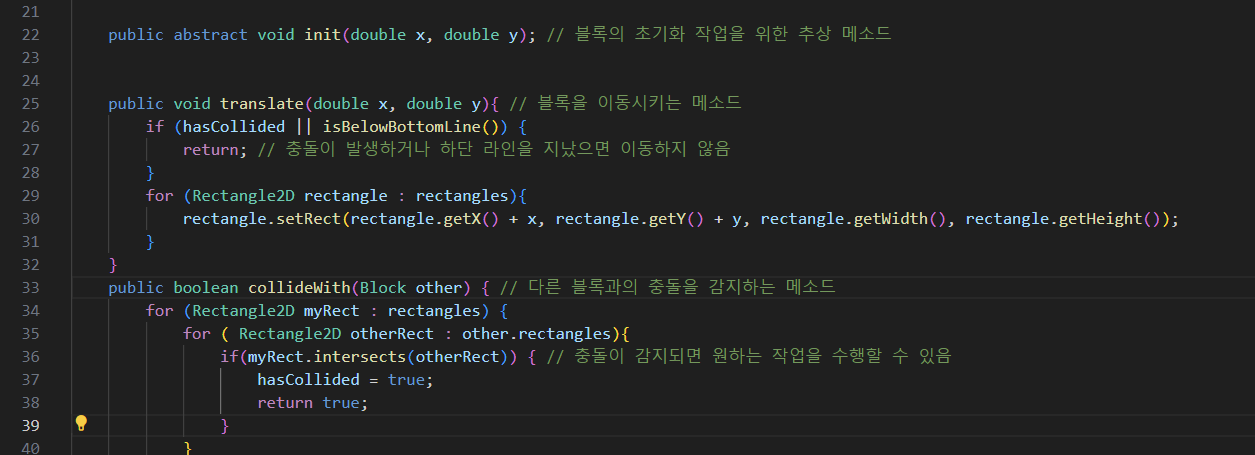
**BlockFactory 클래스)**

****

**BlockFactory 클래스는 블록을 생성하는 클래스이고, 정적 메소드인 getInstance의 파라미터 안에 Block type, double x, double y를 써주어 주어진 블록타입과 좌표에 따라 블록 객체를 반환하게 해주었다. 그 후 Switch문을 사용해 블록 타입에 따라 블록 객체를 생성할 수 있게 해주었다.**

**Block 클래스)**

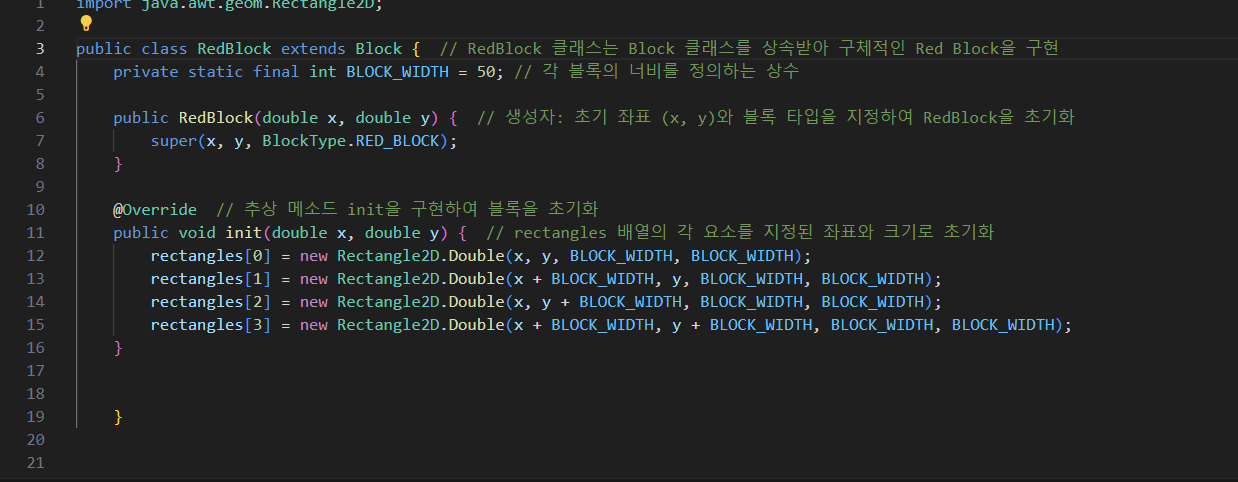
****

****

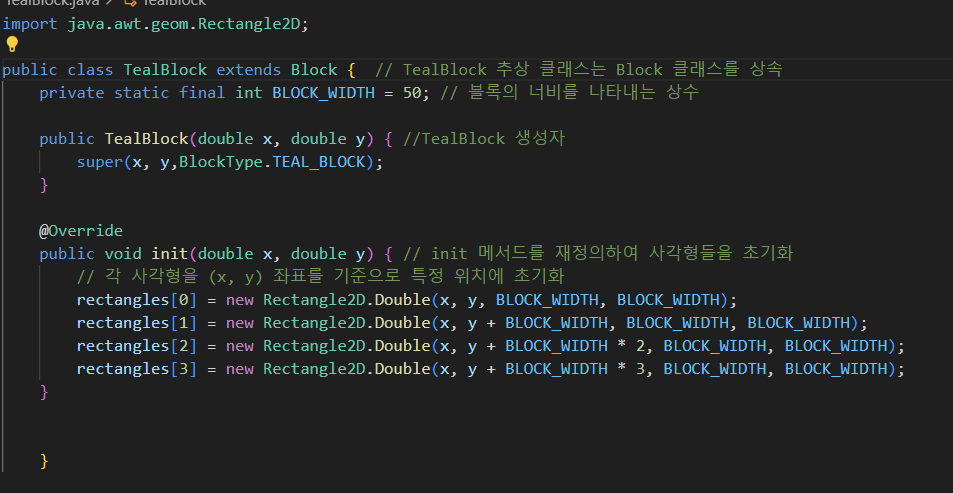
**Block 클래스는 추상클래스로 정의해주었고 IMoveable 과 IColliding 인터페이스를 상속 받게 해주었고 아래에 protected 를 사용해 Block을 상속받는 하위클래스에서 접근할 수 있게 블록 타입을 나타내는 변수를 선언해주었고 동일하게 블록을 구성하는 사각형 객체들을 배열로 선언해주었다. 배열의 길이는 4로 한 블록이 4개의 사각형으로 구성되어 있음을 보여준다. private boolean hasCollided = false ; 를 써서 블록이 충돌했는지 여부를 나타내는 변수를 선언해주었다.**

**그 후 실행창의 맨 하단 경계선에 도달시 멈추도록 하단 경계선을 560으로 선언해주었다. protected Block(double x, double y, BlockType type) 를 써서 블록의 초기 위치와 타입을 받아서 초기화해주는 생성자를 만들어주었다. 그 후 this.type = type; 를 사용해 매개변수로 전달받은 type의 값을 클래스의 type 필드에 할당해주고 init 메소드를 호출하여 블록의 초기위치를 설정해주었다. public BlockType getType() 를 선언해 블록의 타입을 반환할 수 있게 해주었고 public abstract void init(double x, double y) 를 사용해 블록의 초기화 작업을 위한 추상 메소드를 선언해주었다. public void translate(double x, double y) 를 사용해 블록을 이동시키는 메소드를 선언해주었고, if(hasCollied || isBelowBottomLine()) {return;} 를 사용해 블록끼리의 충돌이 발생하거나 블록의 하단이 실행창의 하단라인에 닿으면 이동하지 않도록 설정해주었다. for (Rectangle2D rectangle : rectangles) { rectangle.setRect(rectangle.getX() + x, rectanlge.getY() + y, rectangle.getWidth(), rectangle.getHeight())}; 를 사용해 각 사각형의 위치를 일관되게 업데이트하여 블록의 이동을 구현하였다. public boolean collideWith(Block other)를 선언해 다른 블록과의 충돌을 감지하는 메소드를 선언해주었다. 다음 private boolean isBelowBottomLine()를 사용해 블록이 하단 경계선을 지났는지 확인하는 메소드를 선언했다. public String tostring()을 사용해 블록이 계속 이동하게 되면서 계속해서 변화하는 블록의 상태를 문자열로 반환시켜주는 메소드를 선언해주었다. 마지막으로 public void draw(Graphics2D g2)를 사용해 블록을 화면에 그리는 메소드를 선언해주었고 그 아래에서 g2.setPaint(type.getColor()); 를 사용해 블록타입에 따른 색상으로 채울 수 있게 선언해주었고 g2.setPaint(Color.BLACK)을 사용해 블록의 테두리를 검은색으로 그리도록 선언해주었다.**

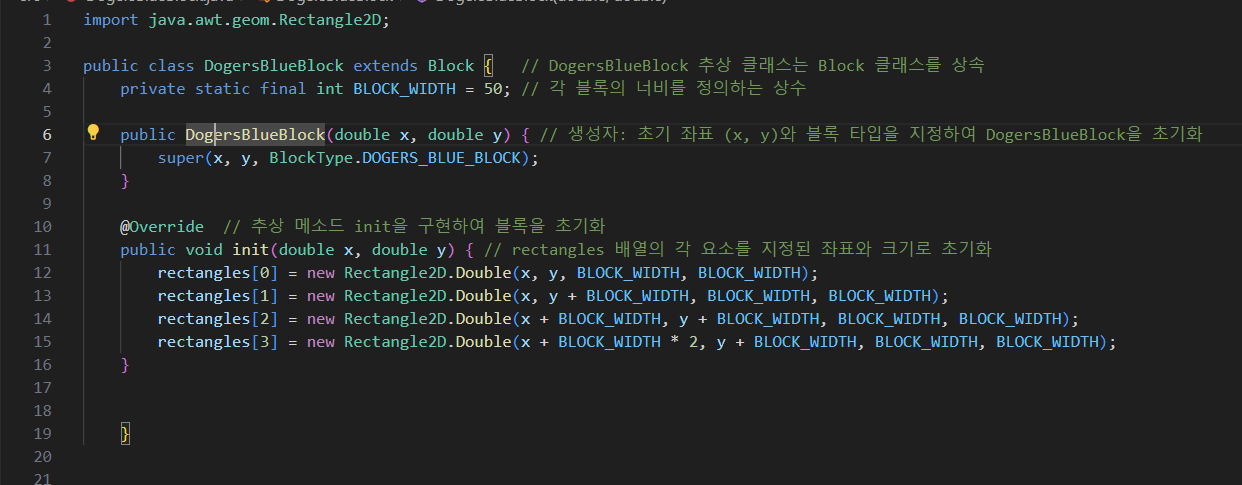
**REDBLOCK 클래스)**

****

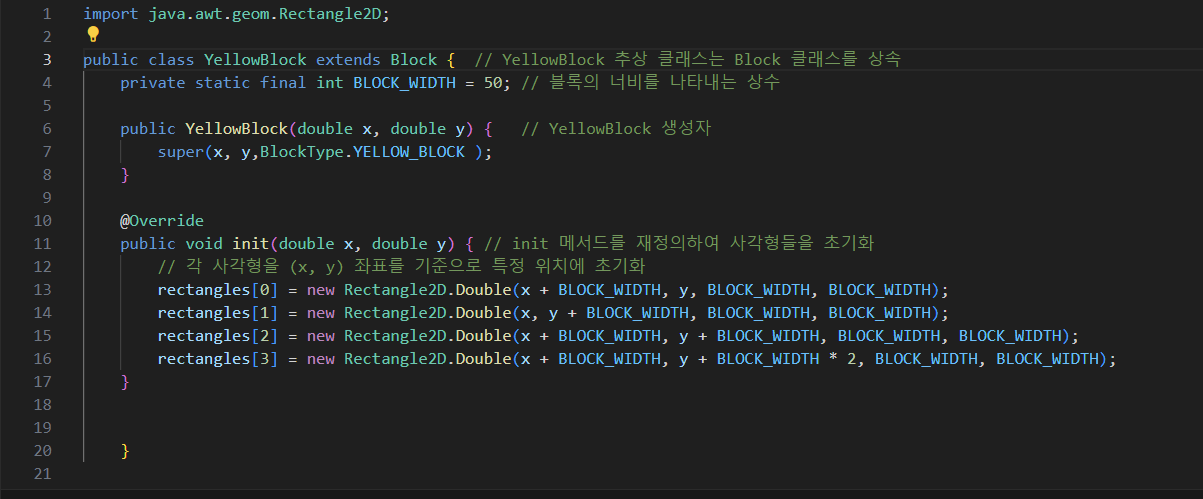
**TEALBLOCK 클래스)**

****

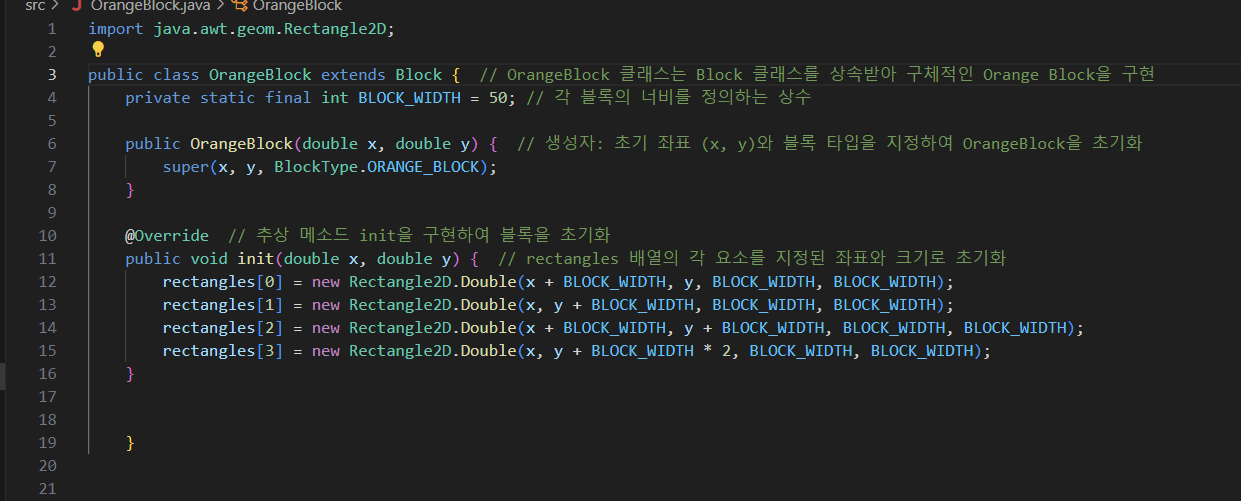
**DOGERBLUEBLOCK 클래스)**

****

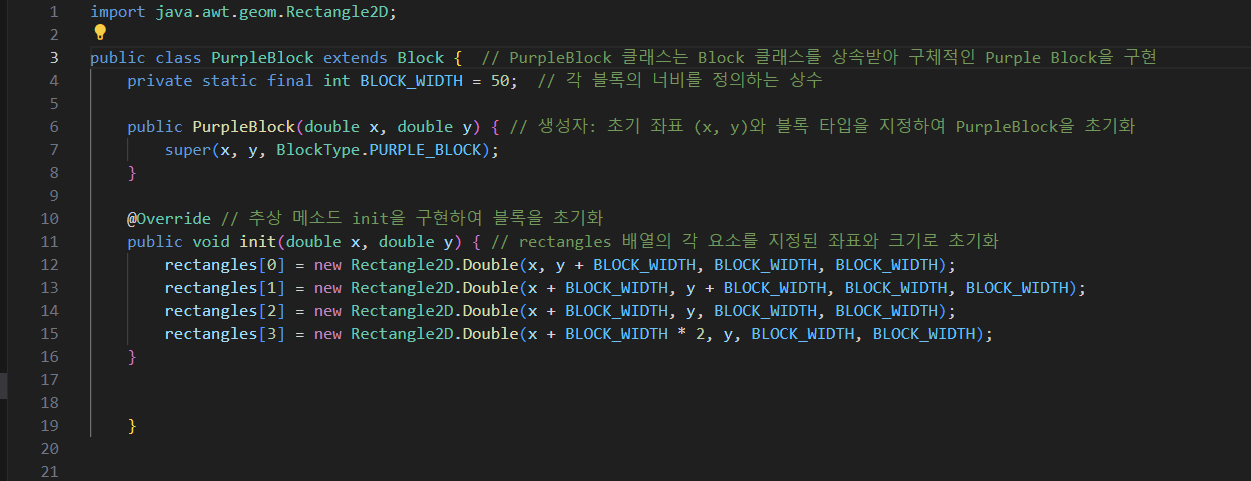
**YELLOW BLOCK 클래스)**

****

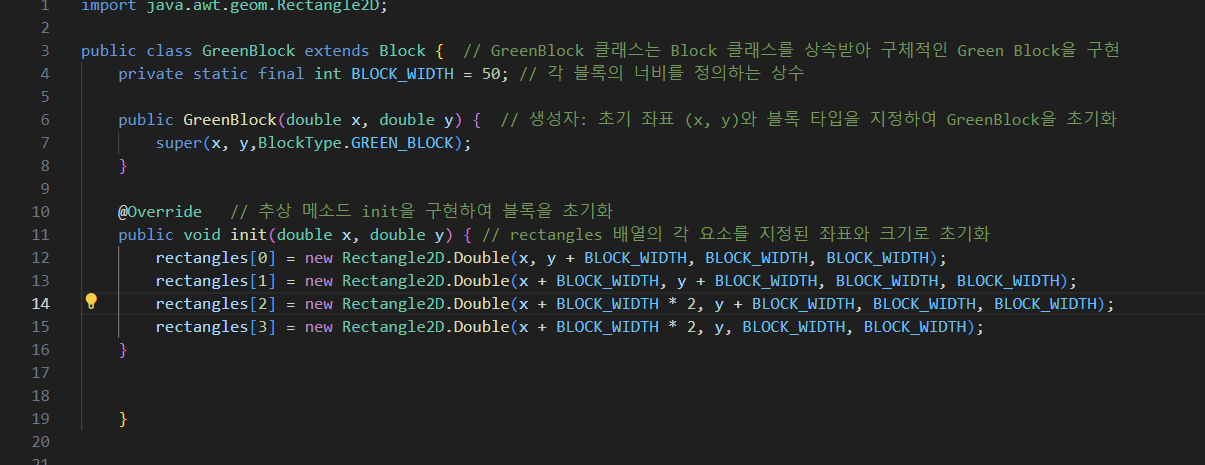
**ORANGE BLOCK 클래스)**

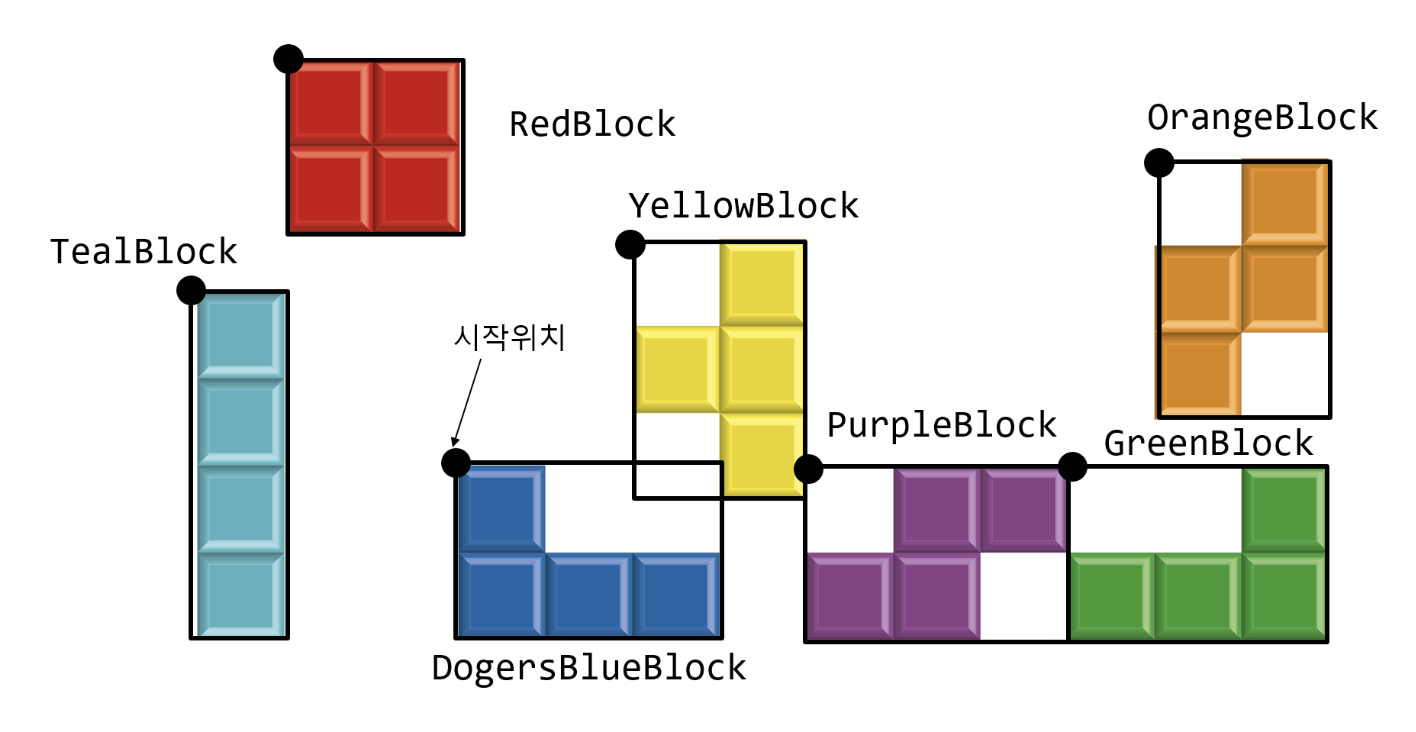
****

**PURPLE BLOCK 클래스)**

****

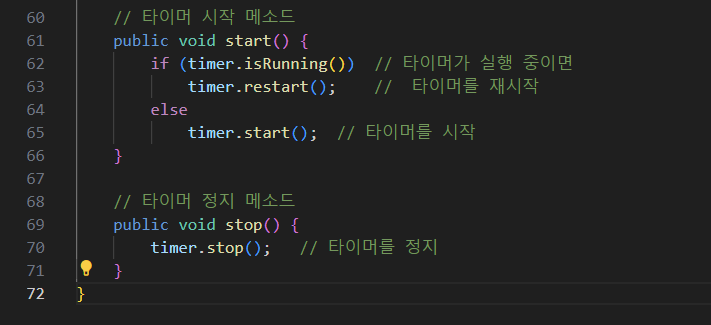
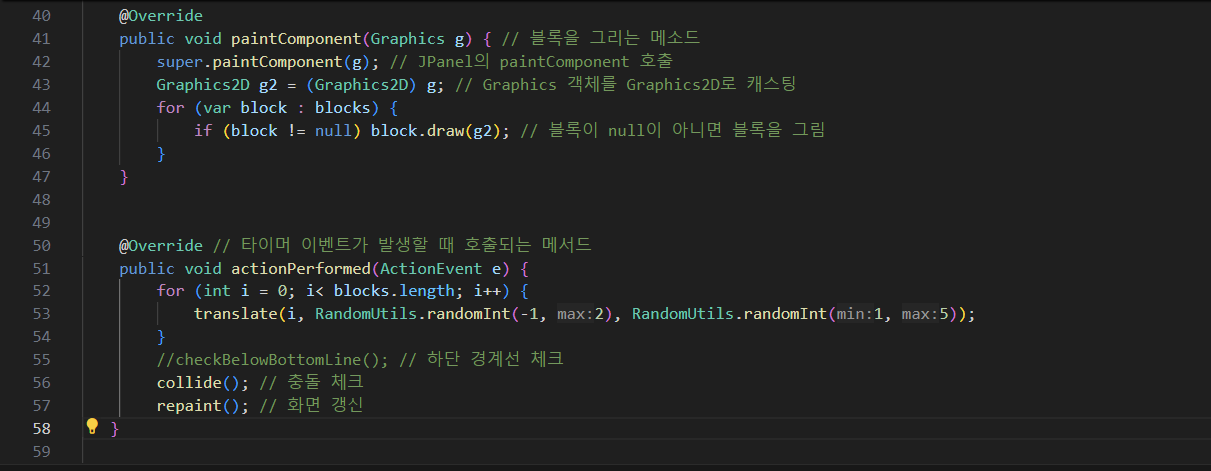
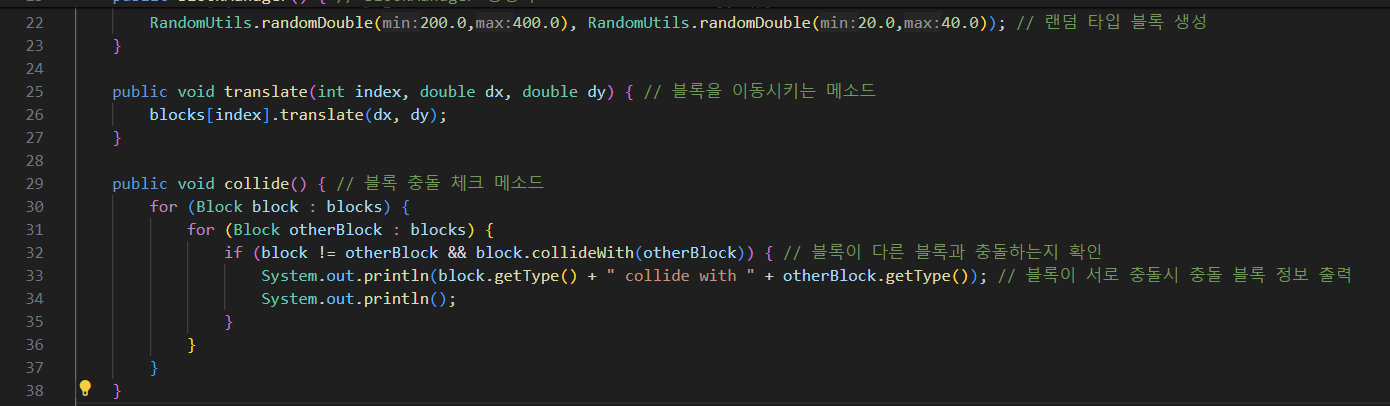
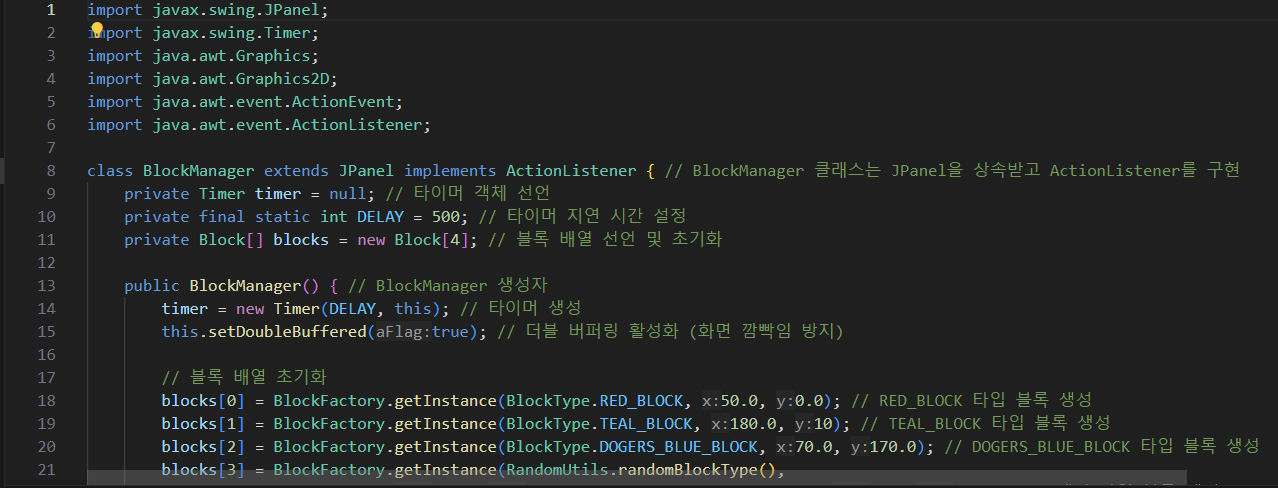
**GREEN BLOCK 클래스)**

****

****

**7개의 각 색이 다른 클래스는 전부 추상 클래스인 Block 클래스를 상속받게 선언해주었고 각 블록의 너비를 50의 상수값으로 선언해주었고, 생성자를 생성해 각 블록의 초기좌표 (x,y) 와 블록타입을 지정하여 블록을 초기화 시키도록 선언해주었다. 다음 추상 메소드인 init을 구현해 블록을 초기화 시켜주었고 rectangles 배열의 각 요소를 지정된 블록타입에 따라 시작위치에서 지정된 좌표로 초기화 하도록 선언해주었다.**

**BlockManagere 클래스)**

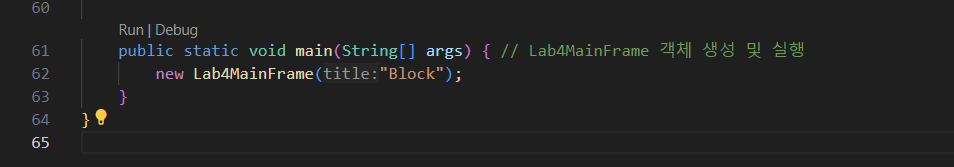
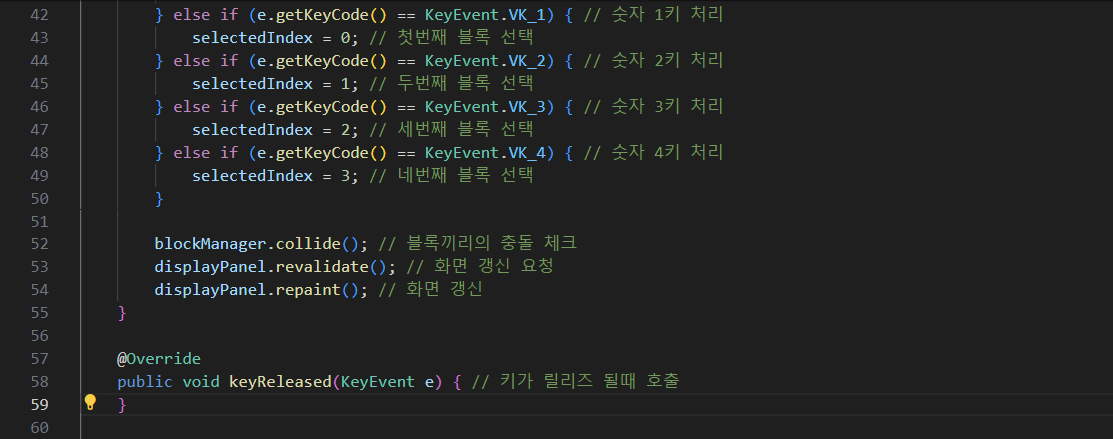
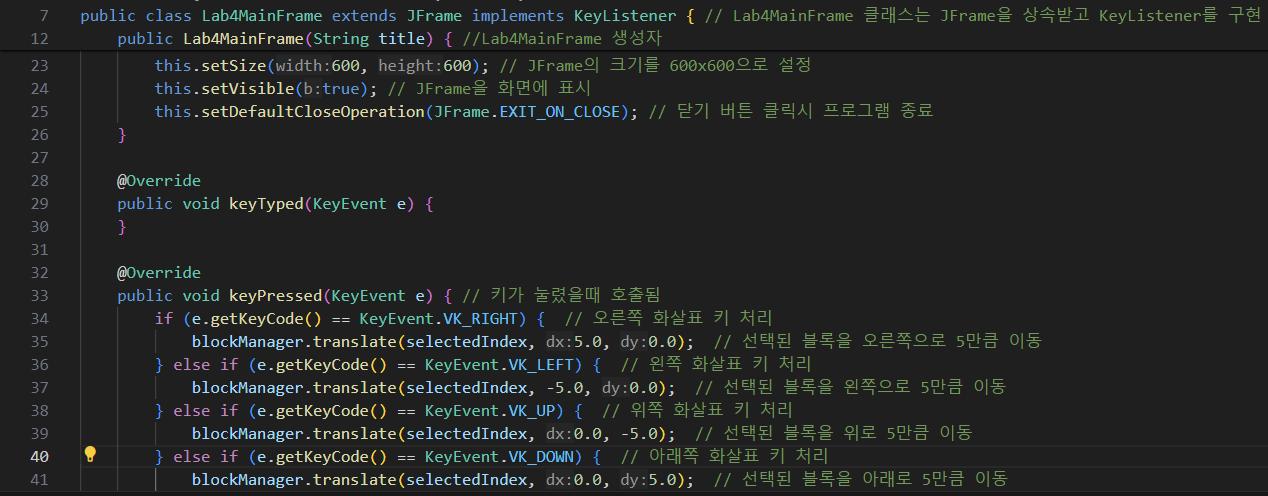
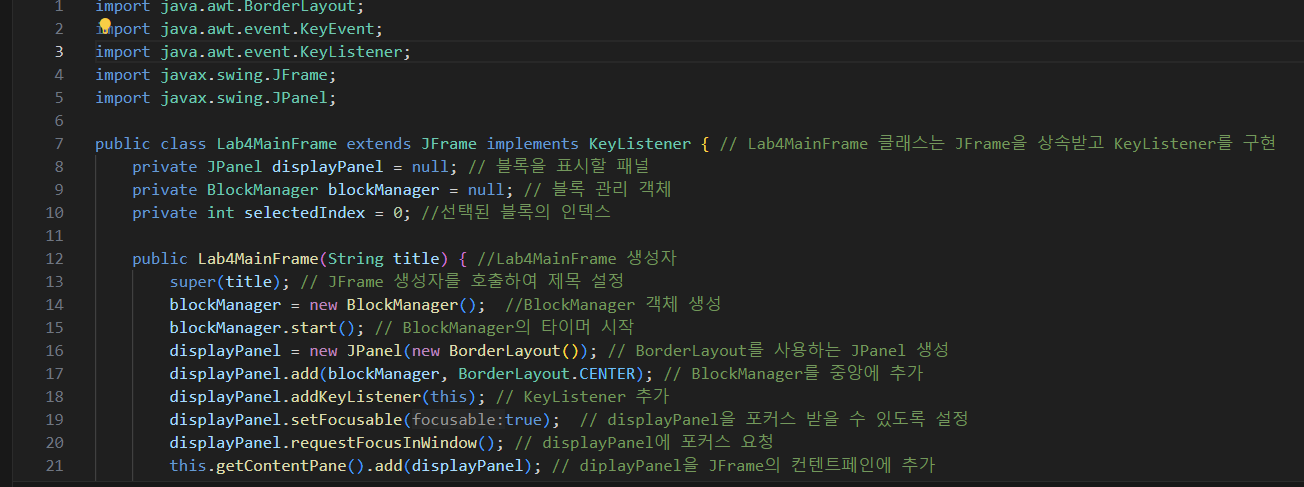
****

**BlockManager클래스는 JPanel을 상속받고 ActionListenerd을 구현해주었고, 아래에 private Timer timer = null; 을 사용해 타이머 객체를 선언해주었고 private final static int DELAY = 500; 을 사용해 타이머 지연 시간 설정을 500초로 설정 했고 private Block[] blocks = new Block[4]; 을 사용해 블록 배열 선언 및 초기화를 시켜주었다. 그 다음 public BlockManager() 를 사용해 Blockmanager 생성자를 생성시켰고 그 안에 timer= new Timer(DELAY, this) 를 선언해 타이머를 생성해주고 this.setDoubleBuffered(true);를 선언해 화면 깜빡임 방지로 더블 버퍼링을 활성화 시켰다. 그 다음 블록 배열을 초기화 시켜주었고 RED\_BLOCK, TEAL\_BLOCK, DOGER\_BLUE\_BLOCK, 그리고 랜덤타입 블록을 생성해주었다. 다음 public void translate(int index, double dx, double dy) {blocks[index].translate(dx,dy); } 를 사용해 블록일 이동시키는 메소드를 선언해주었다. public void collide()를 선언해 블록끼리의 충돌을 확인하는 메소드를 선언했고 if(block != otherBlock && block.collideWith(otherBlock) 을 사용해 블록이 다른블록과 충돌하는지 확인하고 System.out.println(block.getType() + “collide with “ + otherBlock.getType()); 을 사용해 블록이 서로 충돌시 충돌 하는 블록의 정보가 출력되도록 선언해주었다. 그 다음 public void paintComponent(Graphics g)를 사용해 블록을 그리는 메소드를 선언해주었고 super.paintComponent(g) 를 사용해 JPanel 의 paintComponent를 호출했고, Graphics2D g2 = (Graphics2D) g; 를 사용해 Graphics 객체를 Graphics2D로 캐스팅 해주었다. 다음 if(block != null) block.draw(g2); 를 사용해 블록의 값이 null이 아니라면 블록을 그리도록 선언했다. 다음 public void actionPerformed(ActionEvent e)를 사용해 타이버 이벤트가 발생될때 호출되는 메소드를 선언했고 for문을 통해 각 블록을 x축과 y축에 대해 랜덤으로 이동하게 하였고 collide(); , repaint(); 를 선언해줌으로써 블록끼리의 충돌을 체크해주고 화면을 갱신하도록 설정해주었다. 다음 public void start() 를 사용해 타이머 시작 메소드를 호출하였고 if-else 문을 사용해 타이머가 실행중이라면 타이머를 재시작하게하고 타이머가 실행중이 아니라면 타이머를 시작하게 하도록 만들었다.**

**마지막으로 public void stop() 를 사용해 타이머 정지 메소드를 선언했고**

**timer.stop() 을 사용해 타이머를 정지 시키도록 만들었다.**

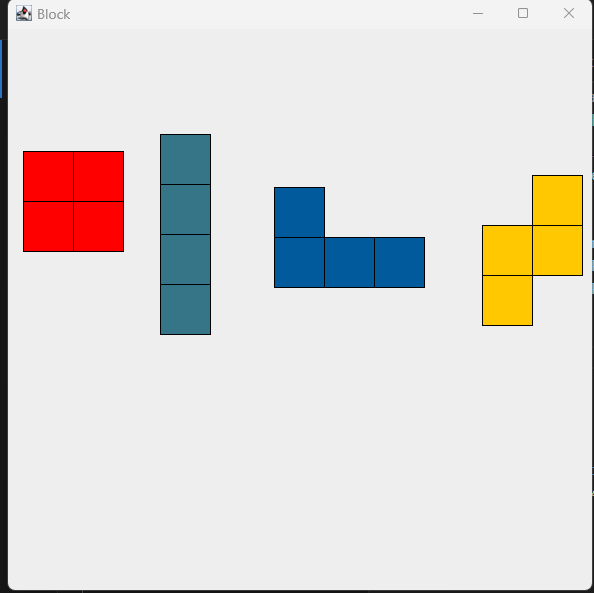
**Lab4MainFrame 클래스)**

****

**Lab4MainFrame 클래스는 JFrame을 상속받고 KeyLisetener를 구현했다. 우선 처음에 블록을 표시할 패널, 블록 관리 객체, 선택된 블록의 인덱스를 선언해주었고, 그 다음 Lab4MainFrame의 생성자를 생성해주었고 그 안에 super(title)를 통해 JFrame생성자를 호출해 제목을 설정해주었고 blockManger = new BlockManager()를 통해 BlockManager의 객체를 생성해주었고 다음 blockManger.start()를 통해 blockManager의 타이머를 시작시켜주었고 displayPanel = new JPanel(new BorderLayout()) 를 통해 BorderLayout를 사용하는 JPanel를 생성시켜주었고 displayPanel.add(blockManager, BorderLayout.CENTER) 를 통해 BlockManager를 중앙에 추가시켜주었고 displayPanel.addKeyListener(this)를 통해 KeyListener를 추가시켜주었다. 그 후 this.setSize(600,600) 를 통해 JFrame 의 크기를 600x600 의 크기로 설정해주었고, this.setVisible(true) 를 통해 JFrame을 화면에 표시할 수 있도록 했고 this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE)를 통해 닫기 버튼 클릭시 프로그램이 종료되도록 만들었다. 그 다음 public void keyPressed(keyEvent e) 를 통해 키가 눌리게 되면 호출되는 메소드를 호출해주었고, if-else 문을 통해서 방향키 위,아래,왼쪽,오른쪽 키를 누르게 되면 선택된 블록들이 각각 5씩 이동하게 해주었고 초기에 선택된 블록은 1번인 첫번째 블록이고 2번을 누를시 두번째 블록에 대해 이동시킬 수 있고 3번을 누를시 세번째 블록에 대해 이동, 4번을 누를시 네번째 블록에 대한 이동을 시킬 수 있도록 설정했다. 그 다음 blockManager.collide() 를 통해 블록끼리의 충돌을 체크하도록 설정했고 displayPanel.revaliate() 를 통해 화면에 대해 갱신을 하도록 요청하게 설정했고 displayPanel.repaint() 를 통해 화면을 갱신하게 설정했다. 마지막으로 public static void main(String[] args) { new Lab4MainFrame(”Block”) } 을 통해 Lab4MainFrame에 대한 객체 생성 및 실행이 되도록 설정했다.**

**실행창) ) **

**현재 1,2,3번 블록들이 서로 충돌해서 멈추게 되었고 실행창아래에 RED\_BLOCK, DOGER\_BLUE\_BLOCK 이 충돌했다 라고 뜨고 있고, TEAL\_BLOCK 과 DOGER\_BLUE\_BLOCK이 충돌했다 라고 뜨고있다. 그리고 4번 블록인 초록색 블럭이 하단라인에 도착하자 멈춘 모습이다.**

****

**4번째 블록은 7가지 블록타입 중 항상 랜덤으로 생성되고 이번에는 직접 1번키, 2번키, 3번키, 4번키를 눌러서 서로 부딪치지 않게 이동시켜서 내려가도록 설정했다.**