2016-06-20

COMPUTER PROGRAMMING

Project Report

기계항공공학부

2010-11616

사세영

**Programming Environment**

* OS : window 10 64bit
* Environment : Eclipse Java EE IDE for Web Developers.

Version: Mars.2 Release (4.5.2)

* System properties:

applicationXMI=org.eclipse.ui.workbench/LegacyIDE.e4xmi

awt.toolkit=sun.awt.windows.WToolkit

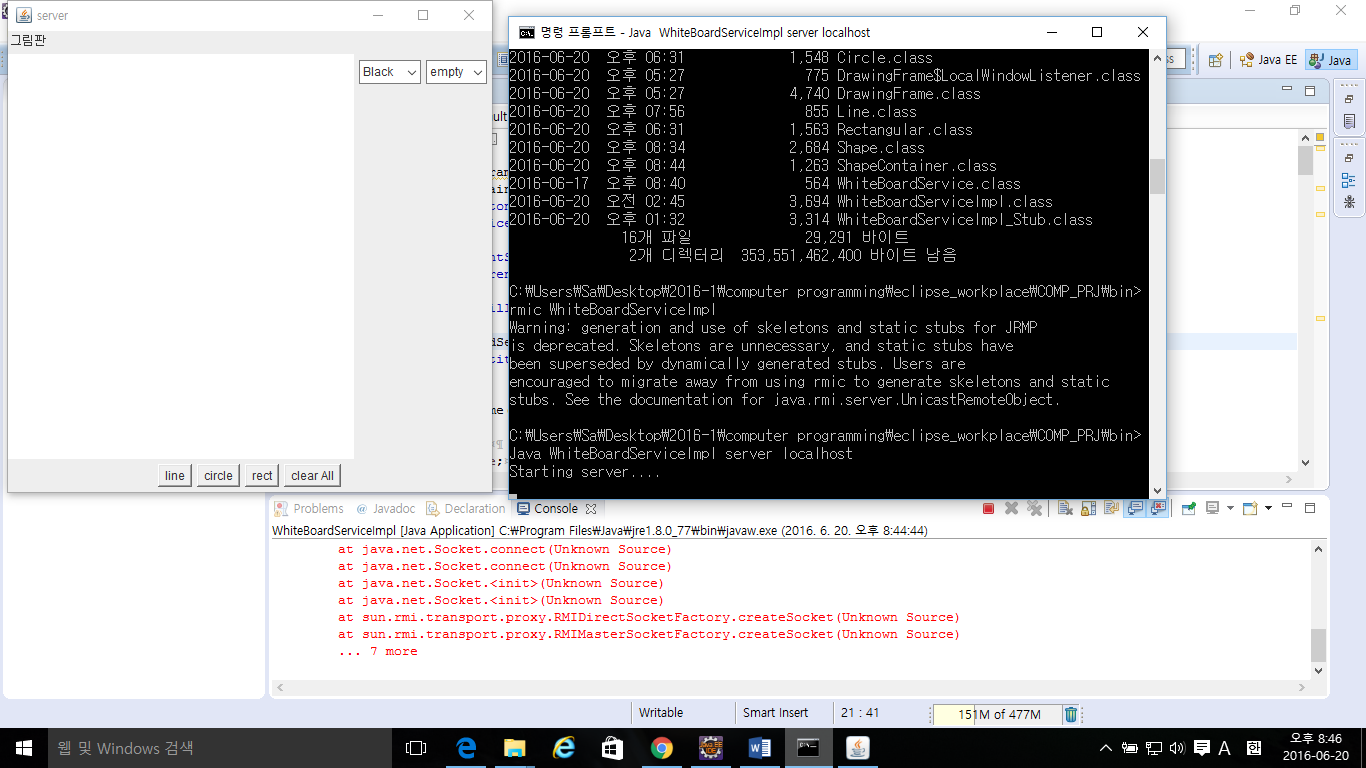
eclipse.application=org.eclipse.ui.ide.workbench

eclipse.buildId=4.5.2.M20160212-1500

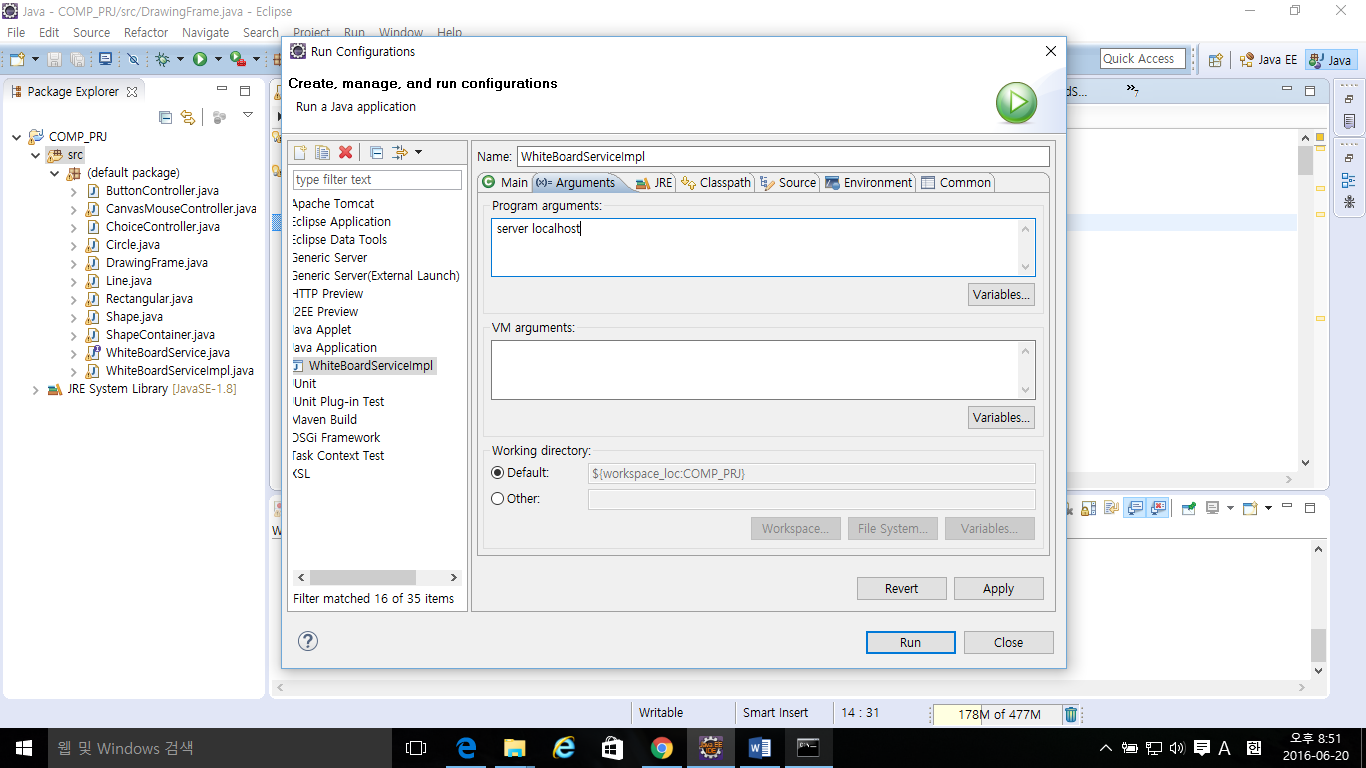
eclipse.commands=-os

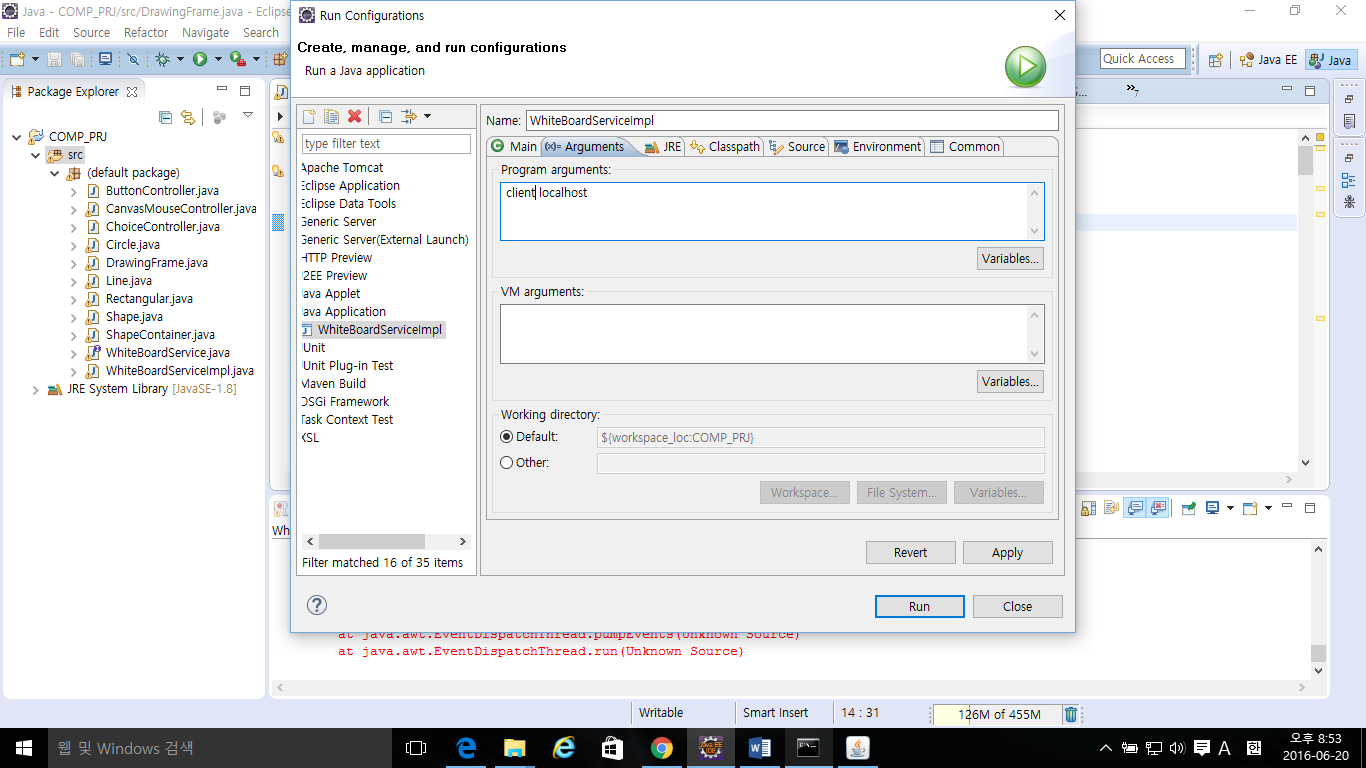
**How to run the program**

1. Move to directory ( folder: ‘bin’ ) and type “rmic WhiteBoardServiceImpl“



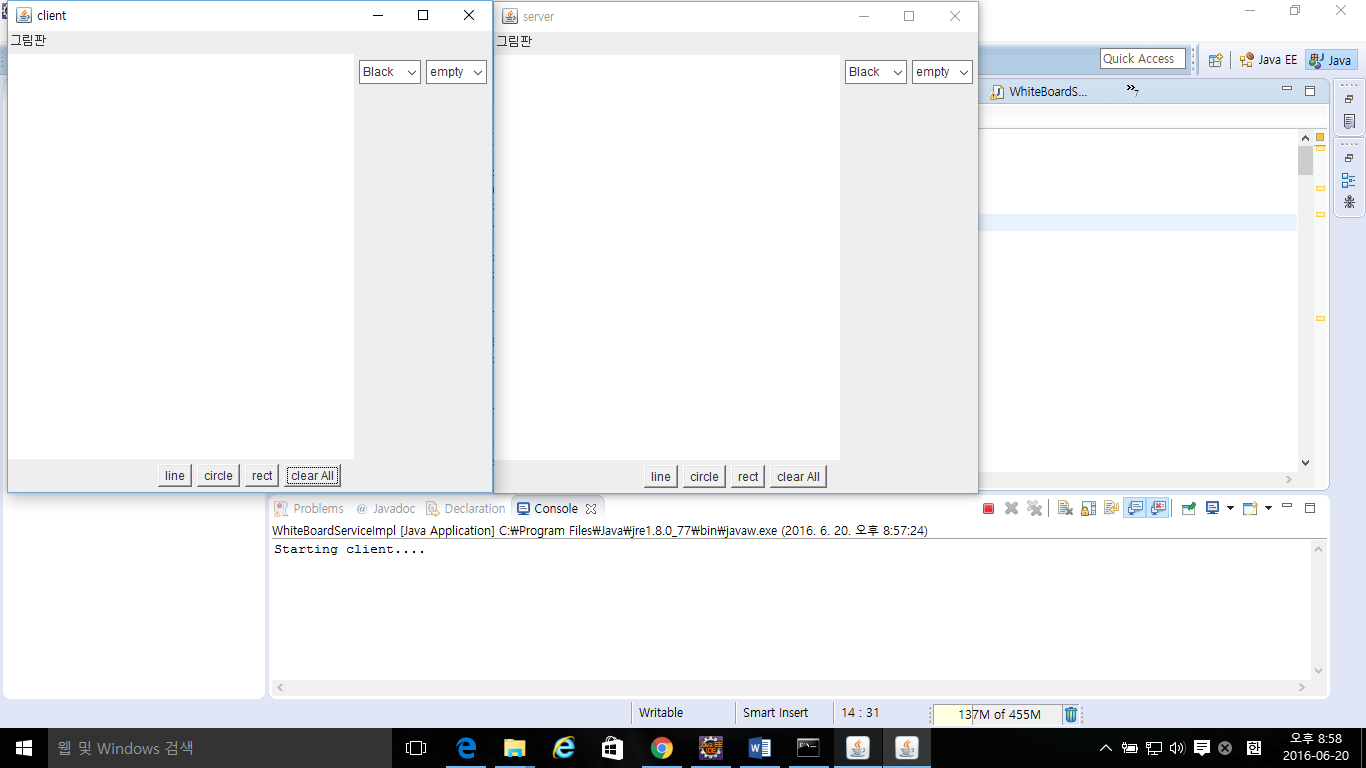
1. Run “WhiteBoardServiceImpl” with arguments both “server localhost” and “client localhost”



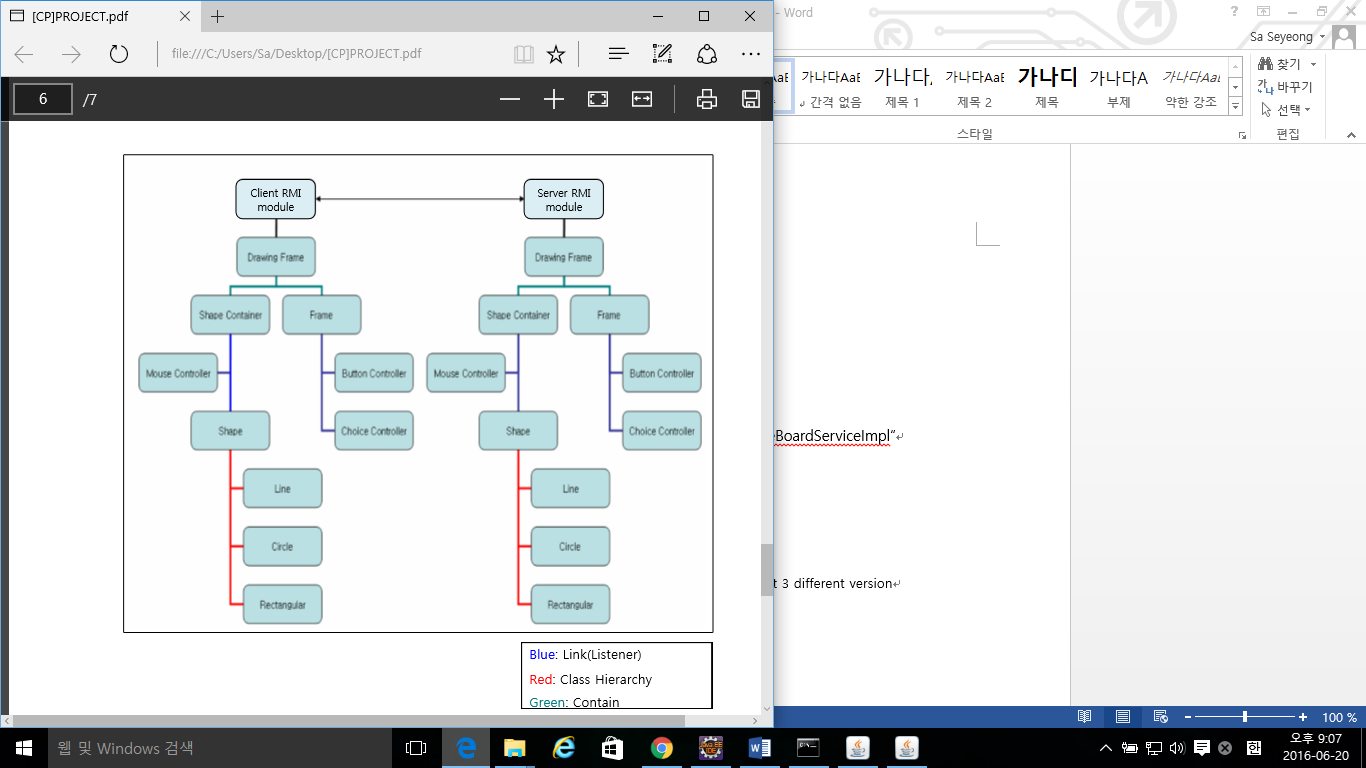


There is some problem to execute the program twice in same time by CMD of Window 10, so I run the program by eclipse.

1. There are two windows of program executed



**Detailed description of code**



1. **Drawing Frame**
2. **Abstract**

In DrawingFrame constructor, There are initialization codes of Frame, ShapeContainer, Controller, etc…

And there are previously implemented codes of context, major member functions.

1. **Code**

* **Constructor**

Initialization of Button Controller

|  |
| --- |
| // Make a Panel instance in Constructor of DrawingFrame  buttonPanel = **new** Panel();    // Make a button controller and make buttons in buttonController  ButtonController buttonController = **new** ButtonController(**this**);  // align panel at South  **this**.getContentPane().add("South", buttonPanel); |

Initialization of Choice Controller

|  |
| --- |
| choicePanel = **new** Panel();    ChoiceController choiceController = **new** ChoiceController(**this**);    // align panel at East  **this**.getContentPane().add("East", choicePanel); |

Implementation and initialization of name Panel

|  |
| --- |
| Panel namePanel = **new** Panel(); // Make Panel    Label drawingLabel = **new** Label(); // Make Label  drawingLabel.setText("그림판 ");    // set layout of Panel to the label locates left upside.  namePanel.setLayout(**new** BorderLayout());  namePanel.add("West",drawingLabel);    **this**.getContentPane().add("North", namePanel); |

Initialization of Shape Container

|  |
| --- |
| shapeContainer = **new** ShapeContainer();    **this**.getContentPane().add("Center", shapeContainer); |

Initialization of Mouse Controller

|  |
| --- |
| CanvasMouseController CMouseC = **new** CanvasMouseController(**this**);    shapeContainer.addMouseListener(CMouseC); // to listen mouse clicked  shapeContainer.addMouseMotionListener(CMouseC); // to listen mouse dragging |

Intialization of other members. ( previously implemented )

**Member Functions**

* **addShape() :** make new shape by two points and add it to shapeContainer and another container of rmi module.
* **removeFrontShape() :** delete recently added shape. Mouse dragged event is using this. Also does in rmi module
* **removeAll :** remove all shape in shapeContainer. Also does in rmi module
* **other get / set functions**

1. **Shape Container**
2. **Abstract**

There are a few changes in Shape Container, implemented overriding member function ‘paintComponents()’ and added statement to ‘paint()’

1. **Code**

* **paintComponents()**

Overriding member function, this call paint function of all components, by order of time. First made component is painted first, so the newest component is come front when it is overlapped.

|  |
| --- |
| **public** **void** paint(Graphics g) {    g.setColor(getBackground());  g.fillRect(0, 0, getWidth(), getHeight());  g.setColor (Color.***black***);  **super**.paint(g);    paintComponents(g); // added this statement    }  // Override paintComponets to paint shapes older to newer, exactly reverse order  **public** **void** paintComponents(Graphics g){  **for**(**int** i = 0; i < **this**.getComponentCount() ; i++ ){  **this**.getComponent(i).paint(g);  }  } |

1. **Button Controller and Choice Controller**
2. **Abstract**

In Button Controller, constructor make 4 buttons and set action listeners. If Each Button clicked, it converts ‘currentShape’ of Drawing Frame.

Choice Controller is similar to Button Controller, make two choices, and one choice affects ‘color’, and the other choice affects ‘fill state’.

1. **Code**

* **Button Controller**

In constructor, there are initialization and adding 4 buttons ( “line”, “circle”, “rect”, “clear” ). And each button is linked action listener called “LocalButtonHandler”.

Local Button Handler converts state of ‘Drawing Frame’ by using set functions.

|  |
| --- |
| **public** ButtonController(DrawingFrame jF)  {  bT = jF;    // Make Button instances  Button lineButton = **new** Button("line");  Button circleButton = **new** Button("circle");  Button rectButton = **new** Button("rect");  Button clearButton = **new** Button("clear All");    // add buttons to buttonPanel  jF.getButtonPanel().add(lineButton);  jF.getButtonPanel().add(circleButton);  jF.getButtonPanel().add(rectButton);  jF.getButtonPanel().add(clearButton);      LocalButtonHandler LBH = **new** LocalButtonHandler();      // Set a action when button is clicked  lineButton.addActionListener(LBH);  circleButton.addActionListener(LBH);  rectButton.addActionListener(LBH);  clearButton.addActionListener(LBH);    }    **private** **class** LocalButtonHandler **implements** ActionListener {  **public** **void** actionPerformed(ActionEvent ae) {    **if**(ae.getActionCommand().equals("line"))  {  bT.setCurrShape(**new** Shape().LINE);  }  **else** **if**(ae.getActionCommand().equals("circle"))  {  bT.setCurrShape(**new** Shape().CIRCLE);  }  **else** **if**(ae.getActionCommand().equals("rect"))  {  bT.setCurrShape(**new** Shape().RECT);  }  **else** **if**(ae.getActionCommand().equals("clear All"))  {  bT.removeAll();  }  }  } } |

* **Choice Controller**

Choice controller is implemented same way, but there are two action listeners to control each choice.

1. **Mouse Controller**
2. **Abstract**

Mouse controller call ‘addShape()’ and ‘removeFrontShape()’ in DrawingFrame when mouse clicked or dragged.

1. **Code**

* **Constructor**

Initialize Point1 and Point2.

|  |
| --- |
| **public** CanvasMouseController(DrawingFrame jF){    bT = jF;  p1 = **new** Point();  p2 = **new** Point();    } |

* **Mouse event Listener**

When mouse Pressed, current location of mouse cursor is to Point1 and Point2, and execute ‘addShape’ to add shape to containers.

When mouse Dragged, only Point2 update and value of Point1 remains. Then execute ‘removeFrontShape’ and ‘addShape’ to convert previous shape to new shape that converts through mouse cursor.

When mouse Released, execute same as when dragged, and finally there is a shape remain until ‘remove All’ button clicked.

|  |
| --- |
| **public** **void** mousePressed(MouseEvent arg0) {    p1 = arg0.getPoint();  p2 = arg0.getPoint();  bT.addShape(p1, p2);  }  **public** **void** mouseDragged(MouseEvent arg0) {  p2 = arg0.getPoint();  bT.removeFrontShape();  bT.addShape(p1, p2);  }  **public** **void** mouseReleased(MouseEvent arg0) {    p2 = arg0.getPoint();  bT.removeFrontShape();  bT.addShape(p1, p2);  } |

1. **Shape**
2. **Abstract**

There is no change in class Shape, just be some experiment.

1. **Content**

* **Two Points**
* **Color and Fill state**
* **Get / Set member function**
* **Constants represent Line, Circle, Rectangle**

1. **Line , Circle, Rectangle**
2. **Abstract**

These classes extend Shape, and there are changes in overriding member function paint(). It paint shape on container

1. **Code**

* **Line**

Draw line that has color of the class member, start Point1 and ends Point2.

|  |
| --- |
| **public** **void** paint (Graphics g) {    g.setColor(**this**.getColor());  g.drawLine( p1.x, p1.y, p2.x, p2.y);    // for overlapping with heavywheight shape  **super**.paint(g);  } |

* **Circle**

Draw Circle that has color of the class member, start Point1 and ends Point2. There are 2 modes fill and empty.

There is some problem when Point2 comes left or upper side of Point1, the shape draw reversely, if the fill state is empty, then shape draws like fill state is filled.

In order to size of Circle is always positive, there are ‘if’, ‘else’ statement.

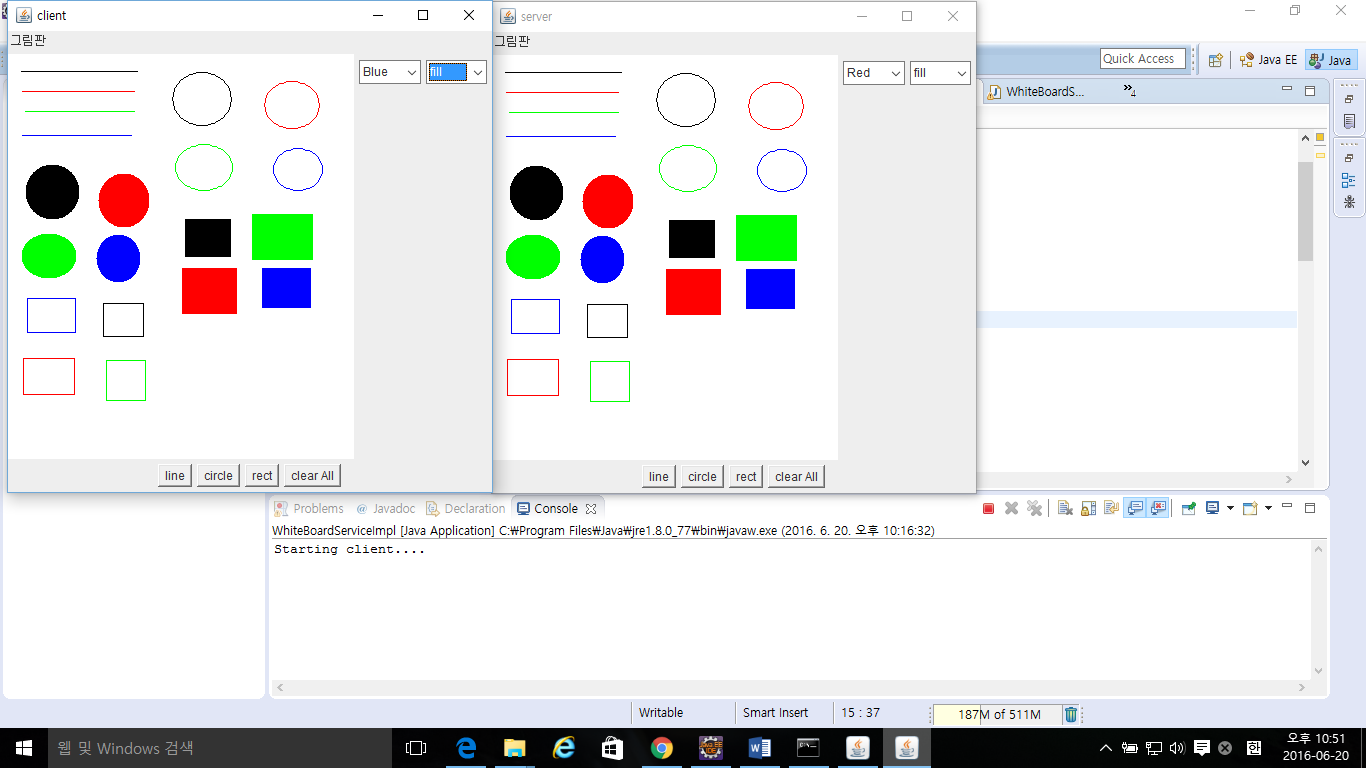
|  |
| --- |
| **public** **void** paint (Graphics g) {    g.setColor(**this**.getColor());    **if**(**this**.fill == **true**){  **if**(p1.x < p2.x)  {  **if**(p1.y < p2.y){  g.fillRect( p1.x, p1.y, (p2.x - p1.x), p2.y - p1.y);  }  **else**{  g.fillRect( p1.x, p2.y, (p2.x - p1.x), p1.y - p2.y);  }  }  **else**  {  **if**(p1.y < p2.y){  g.fillRect( p2.x, p1.y, (p1.x - p2.x), p2.y - p1.y);  }  **else**{  g.fillRect( p2.x, p2.y, (p1.x - p2.x), p1.y - p2.y);  }  }    }  **else**{  **if**(p1.x < p2.x)  {  **if**(p1.y < p2.y){  g.drawRect( p1.x, p1.y, (p2.x - p1.x), p2.y - p1.y);  }  **else**{  g.drawRect( p1.x, p2.y, (p2.x - p1.x), p1.y - p2.y);  }  }  **else**  {  **if**(p1.y < p2.y){  g.drawRect( p2.x, p1.y, (p1.x - p2.x), p2.y - p1.y);  }  **else**{  g.drawRect( p2.x, p2.y, (p1.x - p2.x), p1.y - p2.y);  }  }  }    // for overlapping with heavywheight shape  **super**.paint(g);  } |

* **Rectangle**

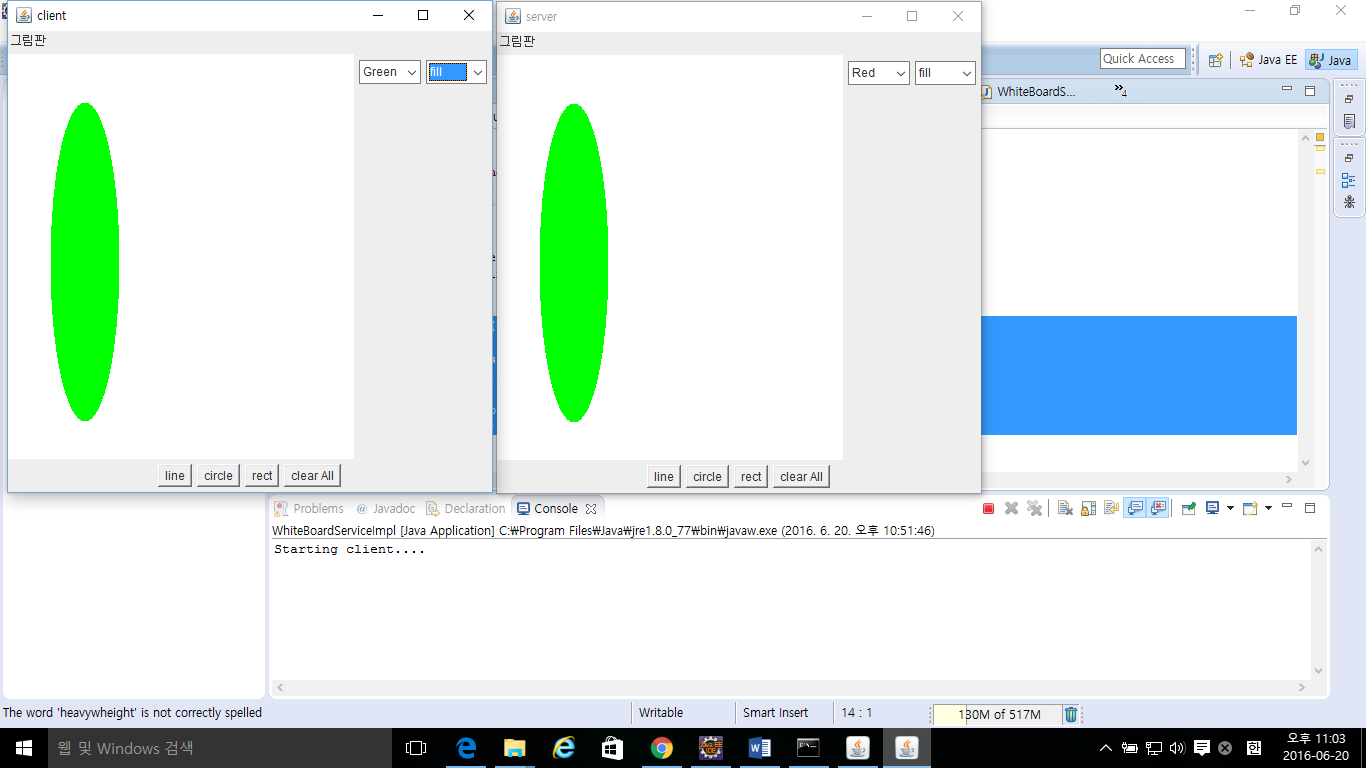
Rectangle is implemented same as Circle.

**Outputs of the Program**

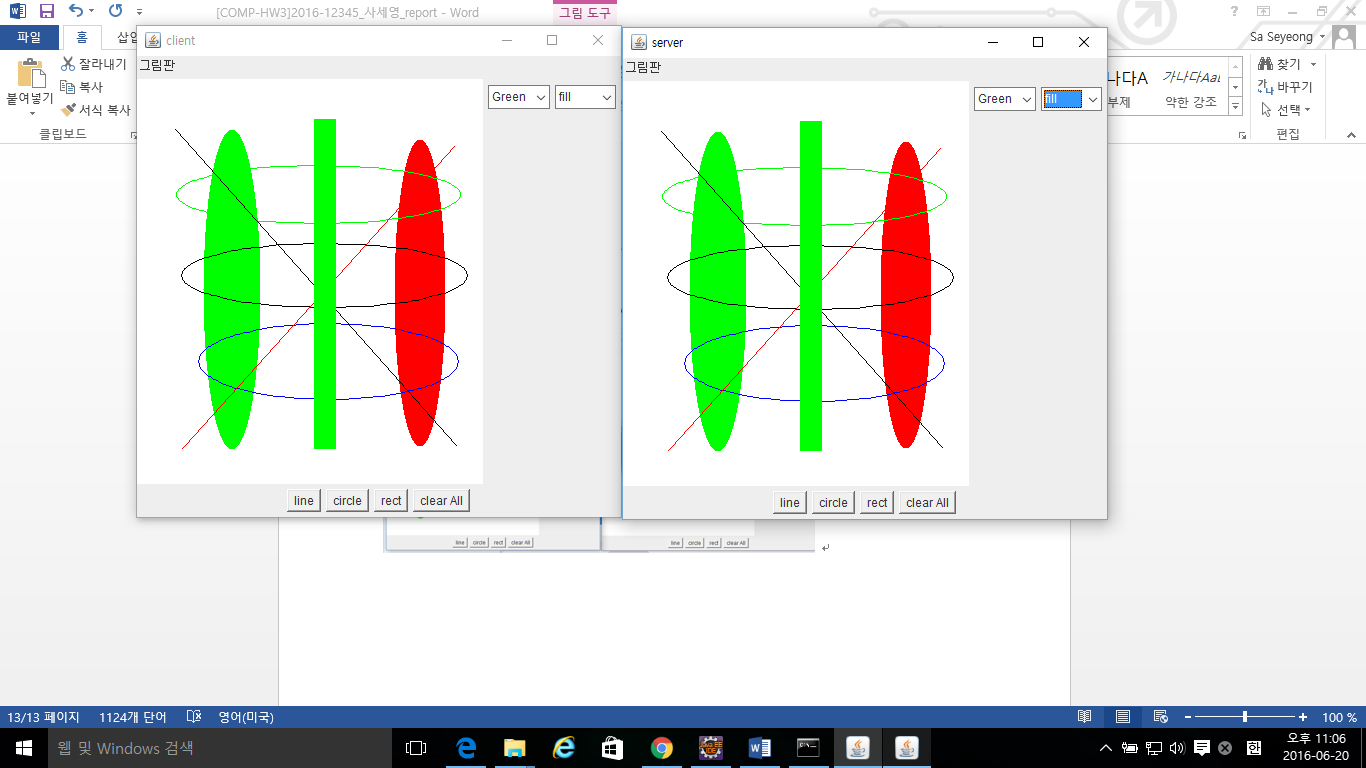
All shape and color doing well.



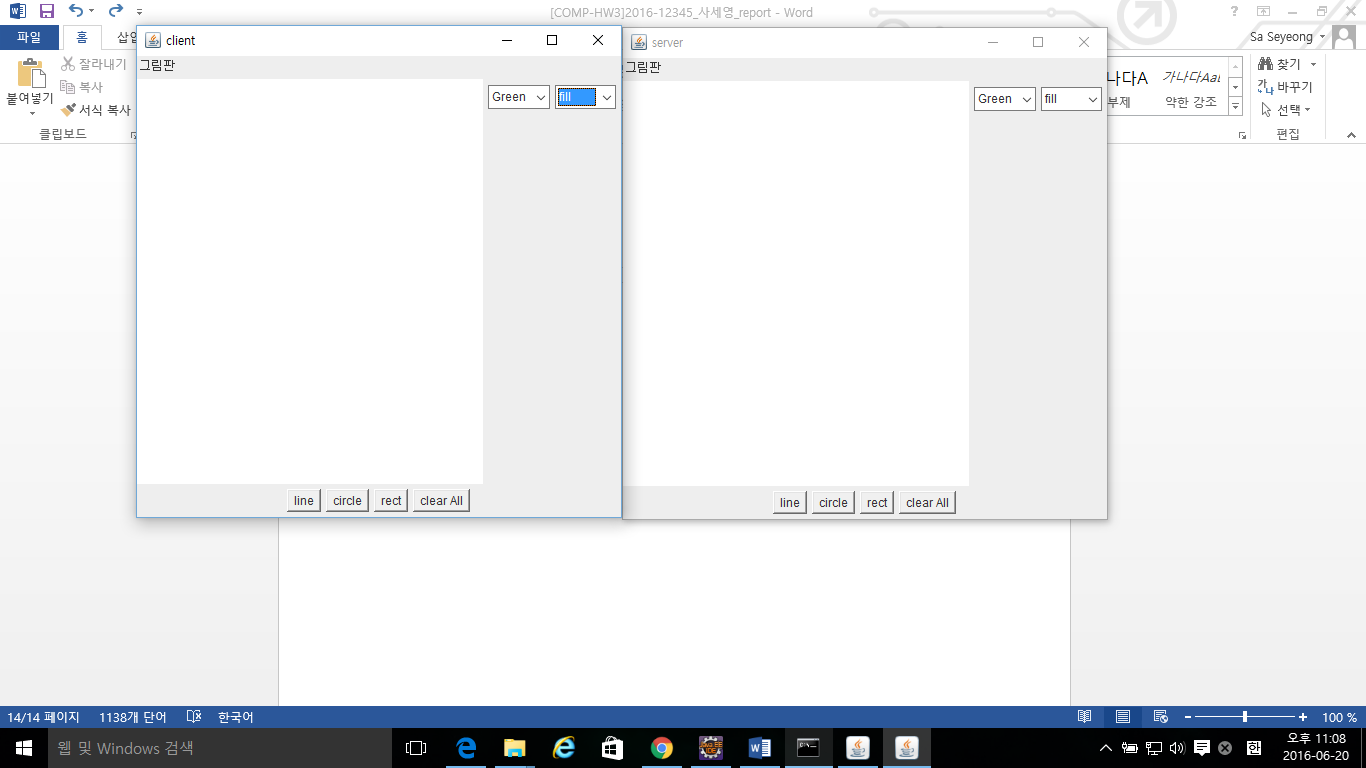
When mouse dragged, the shape follow the mouse cursor.



The newest shape come front and older shape come backside.



Then click clear All.



**Discussion**

In this project, JAVA AWT and Swing are used for implementing Graphic User Interface program. First made frame and panel, made each button, then linked each button and choices to action listener.

The most interesting things I learned in this project is about graphic class methods and the way of executing function ‘paint()’ in Container. The paint method calls its ‘paintComponent()’, then calls other paint method of child class or parent class. The order of calling child class method is reversed, so if there is no other implementation, the recently add shape paint first, so is overlapped previously drawn shapes. I made ‘paintComponents()’ and forced executing the method. But I think it is not good ways, because paint() method may be called several times. In this project, the unlucky event didn’t happen.