DoubleSpongeBob_2_Scaling => DoubleSpongeBob_3_Scaling_Translation

1. Add fields to DrawingComponent for tracking how far the user has
   translated the drawing (in world coordinates), and initialize them
   to (0,0) in the constructor

```
    private int w_translateX;
    private int w_translateY;

    w_translateX = 0;
    w_translateY = 0;
```

2. In DrawingComponent.repaint, call translate on the Graphics2D to account
   for translation in the conversion from world to device coordinates

```
    g2.translate(w_translateX, w_translateY);
```

3. Add a MouseAdapter

```
    this.addMouseListener(mouseAdapter);
    this.addMouseMotionListener(mouseAdapter);
```

4. Add fields and methods for implementing mouse-based translation of shapes

```
    private boolean dragging;
    private int w_dragStartX;
    private int w_dragStartY;
    private int w_dragStartTranslateX;
    private int w_dragStartTranslateY;


    initDrag();


    private void initDrag() {
        dragging = false;
        w_dragStartX = 0;
        w_dragStartY = 0;
        w_dragStartTranslateX = 0;
        w_dragStartTranslateY = 0;
    }


    private MouseAdapter mouseAdapter = new MouseAdapter() {

    @Override
    public void mousePressed(MouseEvent e) {
        int d_X = e.getX();
        int d_Y = e.getY();
```

```
        AffineTransform transform = new AffineTransform();
        transform.scale(scale, scale);
        transform.translate(w_translateX, w_translateY);

        Point2D d_Pt = new Point2D.Double(d_X, d_Y);
        Point2D w_Pt = new Point2D.Double();
        try
        {
            transform.inverseTransform(d_Pt, w_Pt);
        }
        catch (NoninvertibleTransformException ex) {
            return;
        }
        int w_X = (int)w_Pt.getX();
        int w_Y = (int)w_Pt.getY();

        boolean hitShape = false;

        Graphics2D g2 = (Graphics2D)getGraphics();
        for (DrawingShape shape : shapes) {
            if (shape.contains(g2, w_X, w_Y)) {
                hitShape = true;
                break;
            }
        }

        if (hitShape) {
            dragging = true;
            w_dragStartX = w_X;
            w_dragStartY = w_Y;
            w_dragStartTranslateX = w_translateX;
            w_dragStartTranslateY = w_translateY;
        }
    }

    @Override
    public void mouseDragged(MouseEvent e) {
        if (dragging) {
            int d_X = e.getX();
            int d_Y = e.getY();

            AffineTransform transform = new AffineTransform();
            transform.scale(scale, scale);
            transform.translate(w_dragStartTranslateX, w_dragStartTranslateY);

            Point2D d_Pt = new Point2D.Double(d_X, d_Y);
            Point2D w_Pt = new Point2D.Double();
            try
            {
                transform.inverseTransform(d_Pt, w_Pt);
            }
            catch (NoninvertibleTransformException ex) {
                return;
```

```
            }
            int w_X = (int)w_Pt.getX();
            int w_Y = (int)w_Pt.getY();

            int w_deltaX = w_X - w_dragStartX;
            int w_deltaY = w_Y - w_dragStartY;

            w_translateX = w_dragStartTranslateX + w_deltaX;
            w_translateY = w_dragStartTranslateY + w_deltaY;

            repaint();
        }
    }

    @Override
    public void mouseReleased(MouseEvent e) {
        initDrag();
    }

    @Override
    public void mouseWheelMoved(MouseWheelEvent e) {
        return;
    }
};
```