```
DoubleSpongeBob_2_Scaling => DoubleSpongeBob_3_Scaling_Translation


1. Add fields to DrawingComponent for tracking the current origin (in
world
   coordinates), and initialize them to (0,0) in the constructor

        private int w_originX;
        private int w_originY;

        w_originX = 0;
        w_originY = 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_originX, -w_originY);


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_dragStartOriginX;
        private int w_dragStartOriginY;


        initDrag();


        private void initDrag() {
                dragging = false;
                w_dragStartX = 0;
                w_dragStartY = 0;
                w_dragStartOriginX = 0;
                w_dragStartOriginY = 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_originX, -w_originY);

		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_dragStartOriginX = w_originX;
			w_dragStartOriginY = w_originY;
		}
	}

	@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_dragStartOriginX, 
-w_dragStartOriginY);
```

```
				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_originX = w_dragStartOriginX - w_deltaX;
				w_originY = w_dragStartOriginY - w_deltaY;

				repaint();
			}
		}

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

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