
Mauvaise
Conception
JEOPARDY!

Bad design for $200

ActivityModel.java

```
 * <em>Framework</em><br>
 * The interfaces and classes listed below define a framework for progress
 * management.<br>
 * Contract: {@link ActivityManager}, {@link ActivityModel},
 * {@link JActivityWindow}, {@link JActivityIndicator}.
 *
 * @author Werner Randelshofer
 * @version $Id$
 */
public interface ActivityModel extends BoundedRangeModel {

    public static final String INDETERMINATE_PROPERTY = "indeterminate";
    public static final String NOTE_PROPERTY = "note";
    public static final String WARNING_PROPERTY = "warning";
    public static final String ERROR_PROPERTY = "error";
    public static final String CANCELABLE_PROPERTY = "cancelable";
    public static final String CANCELED_PROPERTY = "canceled";
    public static final String CLOSED_PROPERTY = "closed";

    /**
     * Gets the owner of the progress model. This is typically a {@link org.
     * or a {@link org.jhotdraw.api.app.Application}.
     */
    public Object getOwner();

    /**
     * Set cancelable to false if the operation can not be canceled.
     */
    public void setCancelable(boolean b);

    /**
     * Returns true if the operation can be canceled.
     */
```

JActivityView.java

```
private void updateProperties(PropertyChangeEvent evt) {
    if (evt == null || evt.getPropertyName() == null) {
        updateNote();
        updateWarning();
        updateError();
        updateCancelable();
        updateCanceled();
        updateClosed();
        updateIndeterminate();
        return;
    }
    String name = evt.getPropertyName();
    if ((name == null && ActivityModel.NOTE_PROPERTY == null) || (name != null && name.equals(ActivityModel.NOTE_PROPERTY))) {
        updateNote();
    } else if ((name == null && ActivityModel.WARNING_PROPERTY == null) || (name != null && name.equals(ActivityModel.WARNING_PROPERTY))) {
        updateWarning();
    } else if ((name == null && ActivityModel.ERROR_PROPERTY == null) || (name != null && name.equals(ActivityModel.ERROR_PROPERTY))) {
        updateError();
    } else if ((name == null && ActivityModel.CANCELABLE_PROPERTY == null) || (name != null && name.equals(ActivityModel.CANCELABLE_PROPERTY))) {
        updateCancelable();
    } else if ((name == null && ActivityModel.CANCELED_PROPERTY == null) || (name != null && name.equals(ActivityModel.CANCELED_PROPERTY))) {
        updateCanceled();
        updateCancelable();
    } else if ((name == null && ActivityModel.INDETERMINATE_PROPERTY == null) || (name != null && name.equals(ActivityModel.INDETERMINATE_PROPERTY))) {
        updateIndeterminate();
    } else if ((name == null && ActivityModel.CLOSED_PROPERTY == null) || (name != null && name.equals(ActivityModel.CLOSED_PROPERTY))) {
        updateCancelable();
        updateCanceled();
        updateClosed();
    }
}
```

Bad design for $200

- La méthode updateProperties() dans la classe JActivityView souffre d'un cas de contrôle de type.
- Le problème dans ce cas est un manque de flexibilité lorsque on ajoute ou on retire des types de propriétés de la class ActivityModel.
  - C'est une violation de OCP (consultez les diapos 14-18 dans le cours de SOLID).
- Le contrôle de type est un antipatron connu, causant les mêmes problèmes. (consultez les diapos 41-43 dans le cours de Mauvaise Conception).
  - L'exemple dans le diapo 43 est très semblable à l'un trouvé dans le système JHotDraw.
- Si un type est ajouté ou retiré de ActivityModel, cela causera des changements nécessaires aux autres endroits dans le code, en incluant, par exemple, dans la classe JActivityView.
  - Cet effet d'entraînement est un autre antipatron connu appelé Shotgun Surgery (diapo 39 à la Mauvaise Conception).
- Le contrôle de type peut être résolu en introduisant une hiérarchie (par des patrons State ou Strategy), si une n'existe pas ou en exploitant le polymorphisme si la hiérarchie existe.

Bad design for $400

MDIApplication.java

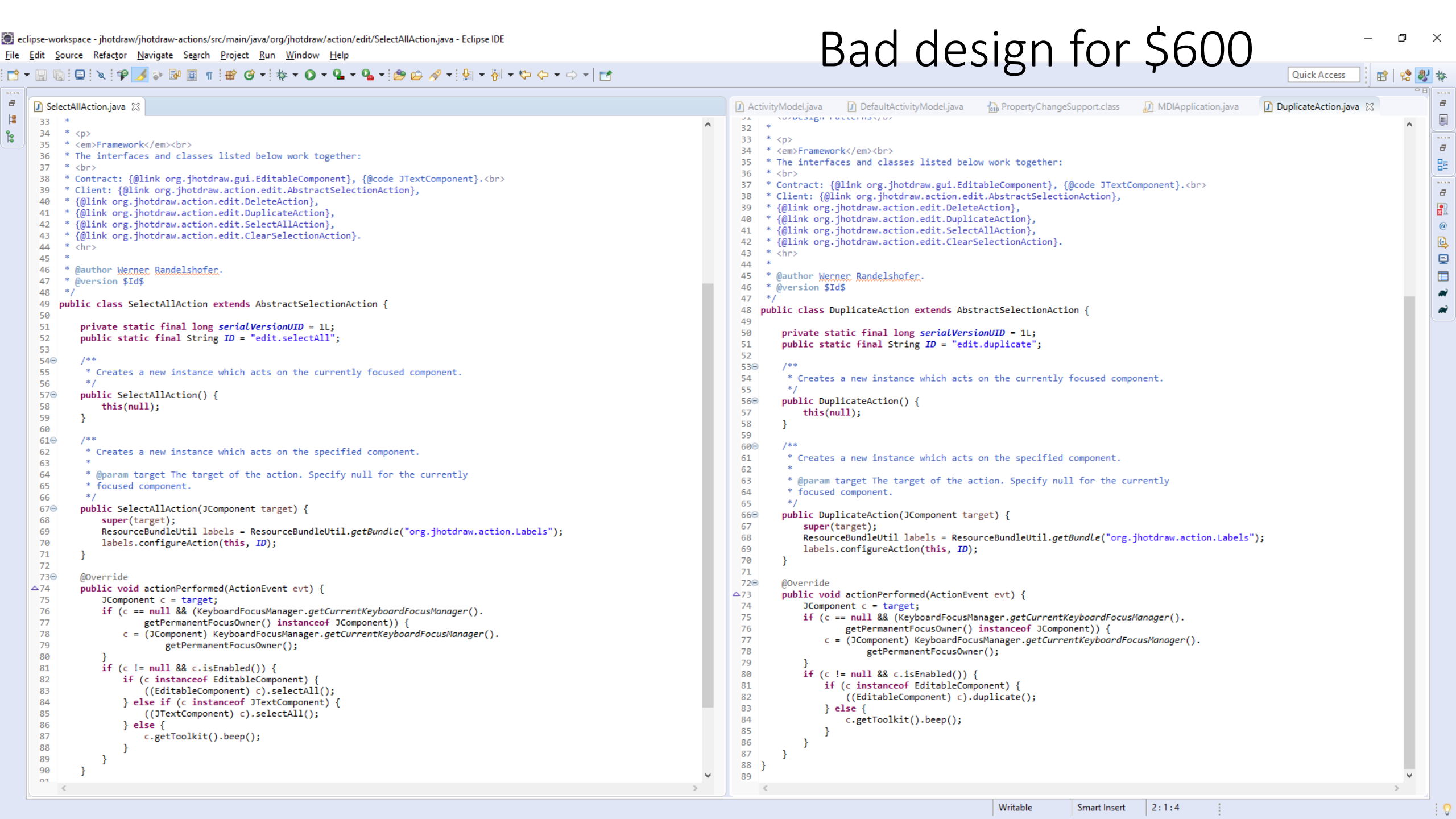
```
@Override
public void show(final View v) {
    if (!v.isShowing()) {
        v.setShowing(true);
        final JInternalFrame f = new JInternalFrame();
        f.setDefaultCloseOperation(JInternalFrame.DO_NOTHING_ON_CLOSE);
        f.setClosable(getAction(v, CloseFileAction.ID) != null);
        f.setMaximizable(true);
        f.setResizable(true);
        f.setIconifiable(false);
        f.setSize(new Dimension(400, 400));
        updateViewTitle(v, f);
        PreferencesUtil.installInternalFramePrefsHandler(prefs, "view", f, desktopPane);
        Point loc = new Point(desktopPane.getInsets().left, desktopPane.getInsets().top);
        boolean moved;
        do {
            moved = false;
            for (View aView : views()) {
                if (aView != v && aView.isShowing()
                        && SwingUtilities.getRootPane(aView.getComponent()).getParent().
                                getLocation().equals(loc)) {
                    Point offset = SwingUtilities.convertPoint(SwingUtilities.getRootPane(aView.getC
                    loc.x += Math.max(offset.x, offset.y);
                    loc.y += Math.max(offset.x, offset.y);
                    moved = true;
                    break;
                }
            }
        } while (moved);
        f.setLocation(loc);
        //paletteHandler.add(f, v);
        f.addInternalFrameListener(new InternalFrameAdapter() {
            @Override
            public void internalFrameClosing(final InternalFrameEvent evt) {
                getAction(v, CloseFileAction.ID).actionPerformed(
                        new ActionEvent(f, ActionEvent.ACTION_PERFORMED,
                                "windowClosing"));
            }

            @Override
            public void internalFrameClosed(final InternalFrameEvent evt) {
                v.stop();
            }
        });
        v.addPropertyChangeListener(new PropertyChangeListener() {
            @Override
            public void propertyChange(PropertyChangeEvent evt) {
                String name = evt.getPropertyName();
                if (((name == null && View.HAS_UNSAVED_CHANGES_PROPERTY == null) || (name != null &&
                        || ((name == null && View.URI_PROPERTY == null) || (name != null && name.equ
```

Bad design for $400

- (Si on avait accès au code entier de la méthode show(),) la méthodes à une Complexité Cyclomatique de 17, un niveau d'imbrication maximum de 6 et 82 lignes de code.
- C'est un cas clair d'une Méthode Longue (consultez la diapo 34 à la Mauvaise Conception).
  - Cela peut être détecté par les métriques (consultez la diapo 50 à la mauvaise conception). Remarque : LOC ne doit pas être trop élevé, mais la complexité accrue peut être considéré comme une raison suffisante.
- La méthode semble d'utiliser beaucoup des classes View et JInternalFrame.
  - Cela peut être considéré comme une indication de Feature Envy (consultez la diapo 40 à la Mauvaise Conception).
- La méthode longue, aussi dans ce cas, peut être résolu en extrayant des parties aux méthodes plus petites et moins complexes ou en utilisant le Compose Method refactoring to pattern (diapo 35 à la Mauvaise Conception).
  - On peut aussi extraire et déplacer les parties qui correspondent aux classes View et JInternalFrame pour résoudre le problème de Feature Envy (consultez la diapo 40 à la Mauvaise Conception).

Bad design for $600




```java
 *
 * <p>
 * <em>Framework</em><br>
 * The interfaces and classes listed below work together:
 * <br>
 * Contract: {@link org.jhotdraw.gui.EditableComponent}, {@code JTextComponent}.<br>
 * Client: {@link org.jhotdraw.action.edit.AbstractSelectionAction},
 * {@link org.jhotdraw.action.edit.DeleteAction},
 * {@link org.jhotdraw.action.edit.DuplicateAction},
 * {@link org.jhotdraw.action.edit.SelectAllAction},
 * {@link org.jhotdraw.action.edit.ClearSelectionAction}.
 * <hr>
 *
 * @author Werner Randelshofer.
 * @version $Id$
 */
public class SelectAllAction extends AbstractSelectionAction {

    private static final long serialVersionUID = 1L;
    public static final String ID = "edit.selectAll";

    /**
     * Creates a new instance which acts on the currently focused component.
     */
    public SelectAllAction() {
        this(null);
    }

    /**
     * Creates a new instance which acts on the specified component.
     *
     * @param target The target of the action. Specify null for the currently
     * focused component.
     */
    public SelectAllAction(JComponent target) {
        super(target);
        ResourceBundleUtil labels = ResourceBundleUtil.getBundle("org.jhotdraw.action.Labels");
        labels.configureAction(this, ID);
    }

    @Override
    public void actionPerformed(ActionEvent evt) {
        JComponent c = target;
        if (c == null && (KeyboardFocusManager.getCurrentKeyboardFocusManager().
                getPermanentFocusOwner() instanceof JComponent)) {
            c = (JComponent) KeyboardFocusManager.getCurrentKeyboardFocusManager().
                    getPermanentFocusOwner();
        }
        if (c != null && c.isEnabled()) {
            if (c instanceof EditableComponent) {
                ((EditableComponent) c).selectAll();
            } else if (c instanceof JTextComponent) {
                ((JTextComponent) c).selectAll();
            } else {
                c.getToolkit().beep();
            }
        }
    }
```

```java
 *
 * <p>
 * <em>Framework</em><br>
 * The interfaces and classes listed below work together:
 * <br>
 * Contract: {@link org.jhotdraw.gui.EditableComponent}, {@code JTextComponent}.<br>
 * Client: {@link org.jhotdraw.action.edit.AbstractSelectionAction},
 * {@link org.jhotdraw.action.edit.DeleteAction},
 * {@link org.jhotdraw.action.edit.DuplicateAction},
 * {@link org.jhotdraw.action.edit.SelectAllAction},
 * {@link org.jhotdraw.action.edit.ClearSelectionAction}.
 * <hr>
 *
 * @author Werner Randelshofer.
 * @version $Id$
 */
public class DuplicateAction extends AbstractSelectionAction {

    private static final long serialVersionUID = 1L;
    public static final String ID = "edit.duplicate";

    /**
     * Creates a new instance which acts on the currently focused component.
     */
    public DuplicateAction() {
        this(null);
    }

    /**
     * Creates a new instance which acts on the specified component.
     *
     * @param target The target of the action. Specify null for the currently
     * focused component.
     */
    public DuplicateAction(JComponent target) {
        super(target);
        ResourceBundleUtil labels = ResourceBundleUtil.getBundle("org.jhotdraw.action.Labels");
        labels.configureAction(this, ID);
    }

    @Override
    public void actionPerformed(ActionEvent evt) {
        JComponent c = target;
        if (c == null && (KeyboardFocusManager.getCurrentKeyboardFocusManager().
                getPermanentFocusOwner() instanceof JComponent)) {
            c = (JComponent) KeyboardFocusManager.getCurrentKeyboardFocusManager().
                    getPermanentFocusOwner();
        }
        if (c != null && c.isEnabled()) {
            if (c instanceof EditableComponent) {
                ((EditableComponent) c).duplicate();
            } else {
                c.getToolkit().beep();
            }
        }
    }
}
```

Bad design for $600

- Code dupliqué entre les classes SelectAllAction et DuplicateAction.
  - consultez la diapo 32 à la Mauvaise Conception.
- La duplication peut compliquer la maintenance; si un clone change, toutes les autres instances doit être mises à jour. Donc, on doit minimiser les points du changement.
- En extrayant les éléments communs afin qu'ils peuvent être réutilisés, c'est la solution pour le code dupliqué.
  - Lorsque deux classes sont liées, on peut extraire les éléments dans une super classe qui sera entendue par les classes des clones.
- Étant donné que dans ce cas la hiérarchie existe, on peut aussi parler d'une hiérarchie non-factorisée (consultez la diapo 28 à la Mauvaise Conception).
  - Une autre réponse acceptable, mais pas nécessairement aussi bonne, est l'Abstraction Dupliquée (consultez la diapo 22 à la Mauvaise Conception).

Bad design for $800

SVGUtil.java/toPathData()

```
public static String toPathData(BezierPath path) {
    StringBuilder buf = new StringBuilder();

    if (path.size() == 0) {
        // nothing to do
    } else if (path.size() == 1) {
        BezierPath.Node current = path.get(0);
        buf.append("M ");
        buf.append(current.x[0]);
        buf.append(' ');
        buf.append(current.y[0]);
        buf.append(" L ");
        buf.append(current.x[0]);
        buf.append(' ');
        buf.append(current.y[0] + 1);
    } else {
        BezierPath.Node previous;
        BezierPath.Node current;

        previous = current = path.get(0);
        buf.append("M ");
        buf.append(current.x[0]);
        buf.append(' ');
        buf.append(current.y[0]);
        for (int i=1, n = path.size(); i < n; i++) {
            previous = current;
            current = path.get(i);

            if ((previous.mask & BezierPath.C2_MASK) =
                if ((current.mask & BezierPath.C1_MASK
                    buf.append(" L ");
                    buf.append(current.x[0]);
                    buf.append(' ');
                    buf.append(current.y[0]);
                } else {
                    buf.append(" Q ");
                    buf.append(current.x[1]);
                    buf.append(' ');
                    buf.append(current.y[1]);
                    buf.append(' ');
                    buf.append(current.x[0]);
                    buf.append(' ');
                    buf.append(current.y[0]);
                }
            } else {
                if ((current.mask & BezierPath.C1_MASK
                    buf.append(" Q ");
                    buf.append(current.x[2]);
                    buf.append(' ');
                    buf.append(current.y[2]);
                    buf.append(' ');
                    buf.append(current.x[1]);
                    buf.append(' ');
                    buf.append(current.y[1]);
                    buf.append(' ');
                    buf.append(current.x[0]);
                    buf.append(' ');
                    buf.append(current.y[0]);
                }
            }
        }
        if (path.isClosed()) {
            if (path.size() > 1) {
                previous = path.get(path.size() - 1);
                current = path.get(0);

                if ((previous.mask & BezierPath.C2_MASK) == 0) {
                    if ((current.mask & BezierPath.C1_MASK) == 0) {
                        buf.append(" L ");
                        buf.append(current.x[0]);
                        buf.append(' ');
                        buf.append(current.y[0]);
                    } else {
                        buf.append(" Q ");
                        buf.append(current.x[1]);
                        buf.append(' ');
                        buf.append(current.y[1]);
                        buf.append(' ');
                        buf.append(current.x[0]);
                        buf.append(' ');
                        buf.append(current.y[0]);
                    }
                } else {
                    if ((current.mask & BezierPath.C1_MASK) == 0) {
                        buf.append(" Q ");
                        buf.append(previous.x[2]);
                        buf.append(' ');
                        buf.append(previous.y[2]);
                        buf.append(' ');
                        buf.append(current.x[0]);
                        buf.append(' ');
                        buf.append(current.y[0]);
                    } else {
                        buf.append(" C ");
                        buf.append(previous.x[2]);
                        buf.append(' ');
                        buf.append(previous.y[2]);
                        buf.append(' ');
                        buf.append(current.x[1]);
                        buf.append(' ');
                        buf.append(current.y[1]);
                        buf.append(' ');
                        buf.append(current.x[0]);
                        buf.append(' ');
                        buf.append(current.y[0]);
                    }
                }
                buf.append(" Z");
            }
```

Bad design for $800

- (Si on avait accès au code entier de la méthode toPathData(),) la méthode a une Complexité Cyclomatique de 12, un niveau d'imbrication maximum de 6, et 115 lignes de code.
- C'est un cas clair d'une Méthode Longue (consultez la diapo 34 à la Mauvaise Conception).
  - Cela peut être détecté par les métriques (consultez la diapo 50 à la mauvaise conception). Remarque : LOC ne doit pas être trop élevé, mais la complexité accrue peut être considéré comme une raison suffisante.
- La méthode semble d'utiliser beaucoup des classes StringBuilder et BezierPath.
  - Cela peut être considéré comme une indication de Feature Envy (consultez la diapo 40 à la Mauvaise Conception).
- La méthode longue, aussi dans ce cas, peut être résolu en extrayant des parties aux méthodes plus petites et moins complexes ou en utilisant le Compose Method refactoring to pattern (diapo 35 à la Mauvaise Conception).
  - On peut aussi extraire et déplacer les parties qui correspondent aux classes StringBuilder et BezierPath pour résoudre le problème de Feature Envy (consultez la diapo 40 à la Mauvaise Conception).
  - On peut aussi minimiser le longueur de la méthode en mergeant tous les appels à la méthode append().

Bad design for $1000

```java
/**
 * Creates a new legend item.
 *
 * @param label  the label ({@code null} not permitted).
 * @param description  the description (not currently used,
 *         {@code null} permitted).
 * @param toolTipText  the tool tip text ({@code null} permitted).
 * @param urlText  the URL text ({@code null} permitted).
 * @param shapeVisible  a flag that controls whether or not the shape is
 *                      displayed.
 * @param shape  the shape ({@code null} permitted).
 * @param shapeFilled  a flag that controls whether or not the shape is
 *                     filled.
 * @param fillPaint  the fill paint ({@code null} not permitted).
 * @param shapeOutlineVisible  a flag that controls whether or not the
 *                             shape is outlined.
 * @param outlinePaint  the outline paint ({@code null} not permitted).
 * @param outlineStroke  the outline stroke ({@code null} not
 *                       permitted).
 * @param lineVisible  a flag that controls whether or not the line is
 *                     visible.
 * @param line  the line ({@code null} not permitted).
 * @param lineStroke  the stroke ({@code null} not permitted).
 * @param linePaint  the line paint ({@code null} not permitted).
 */
public LegendItem(AttributedString label, String description,
                  String toolTipText, String urlText,
                  boolean shapeVisible, Shape shape,
                  boolean shapeFilled, Paint fillPaint,
                  boolean shapeOutlineVisible, Paint outlinePaint,
                  Stroke outlineStroke,
                  boolean lineVisible, Shape line, Stroke lineStroke,
                  Paint linePaint) {

    Args.nullNotPermitted(label, "label");
    Args.nullNotPermitted(fillPaint, "fillPaint");
    Args.nullNotPermitted(lineStroke, "lineStroke");
    Args.nullNotPermitted(line, "line");
    Args.nullNotPermitted(linePaint, "linePaint");
    Args.nullNotPermitted(outlinePaint, "outlinePaint");
    Args.nullNotPermitted(outlineStroke, "outlineStroke");
    this.label = characterIteratorToString(label.getIterator());
    this.attributedLabel = label;
    this.description = description;
    this.shapeVisible = shapeVisible;
    this.shape = shape;
    this.shapeFilled = shapeFilled;
    this.fillPaint = fillPaint;
    this.fillPaintTransformer = new StandardGradientPaintTransformer();
    this.shapeOutlineVisible = shapeOutlineVisible;
    this.outlinePaint = outlinePaint;
    this.outlineStroke = outlineStroke;
    this.lineVisible = lineVisible;
    this.line = line;
    this.lineStroke = lineStroke;
    this.linePaint = linePaint;
    this.toolTipText = toolTipText;
    this.urlText = urlText;
}
```

```java
@Override
public boolean equals(Object obj) {
    if (obj == this) {
        return true;
    }
    if (!(obj instanceof LegendItem)) {
        return false;
    }
    LegendItem that = (LegendItem) obj;
    if (this.datasetIndex != that.datasetIndex) {
        return false;
    }
    if (this.series != that.series) {
        return false;
    }
    if (!this.label.equals(that.label)) {
        return false;
    }
    if (!AttributedStringUtils.equal(this.attributedLabel,
            that.attributedLabel)) {
        return false;
    }
    if (!ObjectUtils.equal(this.description, that.description)) {
        return false;
    }
    if (this.shapeVisible != that.shapeVisible) {
        return false;
    }
    if (!ShapeUtils.equal(this.shape, that.shape)) {
        return false;
    }
    if (this.shapeFilled != that.shapeFilled) {
        return false;
    }
    if (!PaintUtils.equal(this.fillPaint, that.fillPaint)) {
        return false;
    }
    if (!ObjectUtils.equal(this.fillPaintTransformer,
            that.fillPaintTransformer)) {
        return false;
    }
    if (this.shapeOutlineVisible != that.shapeOutlineVisible) {
        return false;
    }
    if (!this.outlineStroke.equals(that.outlineStroke)) {
        return false;
    }
    if (!PaintUtils.equal(this.outlinePaint, that.outlinePaint)) {
        return false;
    }
    if (!this.lineVisible == that.lineVisible) {
        return false;
    }
    if (!ShapeUtils.equal(this.line, that.line)) {
        return false;
    }
    if (!this.lineStroke.equals(that.lineStroke)) {
        return false;
    }
```

Bad design for $1000

- Le constructeur de la classe LegentItem a 15 paramètres.
  - Cela peut être classifié comme un antipatron de Long Parameter List (consultez la diapo 37 à la Mauvaise Conception).
- Cela peut causer une complexité accrue comme dans le cas de la méthode equals().
  - Le résultat peut être une Méthode Longue ou des Switch Statements (consultez les diapos 34 et 41 à la Mauvaise Conception).
- Une solution ici est d'extraire des groups des paramètres en tant que Parameter Objects.
  - Par exemples, les paramètres pertinentes à la peinture, au forme ou aux lignes.
- Si les Parameter Objects peuvent être extraits, on peut aussi penser au cas de l'Abstraction Manquante (consultez la diapo 20 à la Mauvaise Conception).

Bad design for $2000

LogarithmicAxis.java/
refreshTicksVertical()

```
/**
 * Calculates the positions of the tick labels for the axis, storing the
 * results in the tick label list (ready for drawing).
 *
 * @param g2  the graphics device.
 * @param dataArea  the area in which the plot should be drawn.
 * @param edge  the location of the axis.
 *
 * @return A list of ticks.
 */
@Override
protected List refreshTicksVertical(Graphics2D g2, Rectangle2D dataArea,
        RectangleEdge edge) {

    List ticks = new java.util.ArrayList();

    //get lower bound value:
    double lowerBoundVal = getRange().getLowerBound();
    //if small log values and lower bound value too small
    // then set to a small value (don't allow <= 0):
    if (this.smallLogFlag && lowerBoundVal < SMALL_LOG_VALUE) {
        lowerBoundVal = SMALL_LOG_VALUE;
    }
    //get upper bound value
    double upperBoundVal = getRange().getUpperBound();

    //get log10 version of lower bound and round to integer:
    int iBegCount = (int) Math.rint(switchedLog10(lowerBoundVal));
    //get log10 version of upper bound and round to integer:
    int iEndCount = (int) Math.rint(switchedLog10(upperBoundVal));

    if (iBegCount == iEndCount && iBegCount > 0
            && Math.pow(10, iBegCount) > lowerBoundVal) {
          //only 1 power of 10 value, it's > 0 and its resulting
          // tick value will be larger than lower bound of data
        --iBegCount;       //decrement to generate more ticks
    }

    double tickVal;
    String tickLabel;
    boolean zeroTickFlag = false;
    for (int i = iBegCount; i <= iEndCount; i++) {
        //for each tick with a label to be displayed
        int jEndCount = 10;
        if (i == iEndCount) {
            jEndCount = 1;
        }

        for (int j = 0; j < jEndCount; j++) {
            //for each tick to be displayed
            if (this.smallLogFlag) {
                //small log values in use
                tickVal = Math.pow(10, i) + (Math.pow(10, i) * j);
                if (j == 0) {
                    //first tick of group; create label text
                    if (this.log10TickLabelsFlag) {
                        //if flag then
                        tickLabel = "10^" + i;   //create "log10"-type label
                    }
                    else {    //not "log10"-type label
                        if (this.expTickLabelsFlag) {
                            //if flag then
                            tickLabel = "1e" + i;  //create "1e#"-type label
                        }
                        else {    //not "1e#"-type label
                            if (i >= 0) {   // if positive exponent then
                                            // make integer
                                NumberFormat format
                                    = getNumberFormatOverride();
                                if (format != null) {
                                    tickLabel = format.format(tickVal);
                                }
                                else {
                                    tickLabel = Long.toString((long)
                                            Math.rint(tickVal));
                                }
                            }
                            else {
                                //negative exponent; create fractional value
                                //set exact number of fractional digits to
                                // be shown:
                                this.numberFormatterObj
                                    .setMaximumFractionDigits(-i);
                                //create tick label:
                                tickLabel = this.numberFormatterObj.format(
                                        tickVal);
                            }
                        }
                    }
                }
                else {   //not first tick to be displayed
                    tickLabel = "";     //no tick label
                }
            }
            else { //not small log values in use; allow for values <= 0
                if (zeroTickFlag) {      //if did zero tick last iter then
                    --j;
                }               //decrement to do 1.0 tick now
                tickVal = (i >= 0) ? Math.pow(10, i) + (Math.pow(10, i) * j)
                        : -(Math.pow(10, -i) - (Math.pow(10, -i - 1) * j));
                if (j == 0) {  //first tick of group
                    if (!zeroTickFlag) {     // did not do zero tick last
                                             // iteration
                        if (i > iBegCount && i < iEndCount
                                && Math.abs(tickVal - 1.0) < 0.0001) {
                            // not first or last tick on graph and value
                            // is 1.0
                            tickVal = 0.0;        //change value to 0.0
                            zeroTickFlag = true;  //indicate zero tick
                            tickLabel = "0";      //create label for tick
                        }
                        else {
                            //first or last tick on graph or value is 1.0
                            //create label for tick:
                            if (this.log10TickLabelsFlag) {
                                //create "log10"-type label
                                tickLabel = (((i < 0) ? "-" : "")
                                        + "10^" + Math.abs(i));
```

Bad design for $2000

- (Si on avait accès au code entier de la méthode refreshTicksVertical(),) la méthode a une Complexité Cyclomatique de 29, un niveau d'imbrication maximum de 10, et 139 lignes de code.
- C'est un cas clair d'une Méthode Longue (consultez la diapo 34 à la Mauvaise Conception).
  - Cela peut être détecté par les métriques (consultez la diapo 50 à la mauvaise conception). Remarque : LOC ne doit pas être trop élevé, mais la complexité accrue peut être considéré comme une raison suffisante.
- La méthode longue, aussi dans ce cas, peut être résolu en extrayant des parties aux méthodes plus petites et moins complexes ou en utilisant le Compose Method refactoring to pattern (diapo 35 à la Mauvaise Conception).