



28.5 Workshop: Refactoring with Eclipse






Consider the source code shown below provides of a poorly structured software application, written in . The software has been placed on Moodle in a zip archive under 'Lectures'.  versions are available. One version is the unformatted, obfuscated original. The other is the end product of comprehension and refactoring.


Recovering structure –
example (879)


```
import java.awt.*;import java.util.*;public class F
extends Frame{Date D=new Date();void T(Date d){D=d;
repaint();}double P=Math.PI,A=P/2,a,c,U=.05;int W,H,m,R;
double E(int a,int u){return(3*P/2+2*P*a/u)%(2*P);}void N
(Graphics g,double q,double s){g.fillPolygon(new int[]{H(
s,q),H(U,q+A),H(U,q+3*A)},new int[]{J(s,q),J(U,q+A),J(U,q
+3*A)},3);}public void paint(Graphics g){Color C=
SystemColor.control;g.setColor(C);g.fillRect(0,0,W=size()
.width,H=size().height);W-=52;H-=52; R=Math.min(W/2,H/2);
g.translate(W/2+25,H/2+36);g.setColor(C.darker());for(m
=0;m<12;++m){a=E(m,12);g.drawLine(H(.8),J(.8),H(.9),J(.9)
);}m=D.getMinutes();N(g,E(D.getHours()*60+m,720),.5);N(g,
E(m,60),.8);N(g,E(D.getSeconds(),60),.9);}int H(double y)
{return(int)(R*y*Math.cos(a));}int H(double y,double q){a
=q;return H(y);}int J(double y){return(int)(R*y*Math.sin(
a));}int J(double y,double q){a=q;return J(y);}public
static void main(String[]_)throws Exception{F c=new F();c
.resize(200,200);c.show();for(;;){c.T(new Date());Thread
sleep(200);}}}
```


Fortunately the code may be amenable to comprehension and restructuring, so we may be able to recover its functionality relatively easily.

Contextualised refactoring tools are integrated into the Eclipse IDE. You can access any refactoring for a fragment of source code by highlighting the fragment, right clicking the mouse button. This will cause a context menu to appear. From there, navigate to 'Refactor' and then choose the refactoring you wish to apply.


For example, to rename the  class to  select the  identifier in the class declaration. Then right click the mouse button, choose 'Refactor', then 'Rename...'. A tooltip will appear in the editor next to the  identifier with the description 'Enter new name, press Enter to refactor'. Type the new name  directly into the editor and press return.

You may get a warning dialogue explaining that changing the name of the class may break any external dependencies such as scripts. You can ignore this warning for now and click 'Continue' (What type of problem associated with refactoring is the dialogue warning you about?). All references to the  class are updated to the new name in the source file.

Eclipse can also be used to automatically format source code. To do this, open a source file  and choose 'Source → Format' from the window menu.

Alternatively, use the keyboard shortcut 'Control, Shift and F' 

For the tutorial follow the steps taken below to refactor the F.java source file into the Clock.java file, using the automated refactoring tools wherever possible.

The first step is to apply some formatting to the source code structure, using the features of an automated tool such as Eclipse . The code shown below shows the result of this step.

```
import java.awt.*;
import java.util.*;

public class F extends Frame {
    Date D = new Date();

    void T(Date d) {
        D = d;
        repaint();
    }

    double P = Math.PI, A = P / 2, a, c, U = .05;
    int W, H, m, R;

    double E(int a, int u) {
        return (3 * P / 2 + 2 * P * a / u) % (2 * P);
    }

    void N(Graphics g, double q, double s) {
        g.fillPolygon(new int[] { H(s, q), H(U, q + A), H(U, q + 3 * A) },
            new int[] { J(s, q), J(U, q + A), J(U, q + 3 * A) }, 3);
    }

    public void paint(Graphics g) {
        Color C = SystemColor.control;
        g.setColor(C);
        g.fillRect(0, 0, W = size().width, H = size().height);
        W -= 52;
        H -= 52;
        R = Math.min(W / 2, H / 2);
        g.translate(W / 2 + 25, H / 2 + 36);
        g.setColor(C.darker());
        for (m = 0; m < 12; ++m) {
            a = E(m, 12);
            g.drawLine(H(.8), J(.8), H(.9), J(.9));
        }
        m = D.getMinutes();
        N(g, E(D.getHours() * 60 + m, 720), .5);
        N(g, E(m, 60), .8);
        N(g, E(D.getSeconds(), 60), .9);
    }

    int H(double y) {
        return (int) (R * y * Math.cos(a));
    }

    int H(double y, double q) {
        a = q;
        return H(y);
    }

    int J(double y) {
```

```

    return (int) (R * y * Math.sin(a));
}

int J(double y, double q) {
    a = q;
    return J(y);
}

public static void main(String[] _) throws Exception {
    F c = new F();
    c.resize(200, 200);
    c.show();
    for (;;) {
        c.T(new Date());
        Thread.sleep(200);
    }
}
}

```

This step reveals immediately that:

- the program has a graphical user interface because it extends the class `Frame` from the `java.awt` package;
- there is an unused attribute `c` in the `F` class that can be removed; and
- the program has a main method that constructs a new instance of type `F` and then passes the instance a new `Date` instance every 200 milliseconds, using the `T(Date)` method.

Following the method `can`, we can see that the `Date` instance is assigned to an instance attribute (`a`), so we could rename this method to `setDate()` and the attribute and parameter to `date`. Automatic refactoring tools in IDEs such as Eclipse can be helpful here.

The `main(String[] _)` method (880)

The next step is to investigate the `paint(Graphics g)` method, which is called when `repaint()` is called. Looking at the attributes `w` and `h`, we can see that they are used to store the width and height of the application frame, so we can also rename these. Also notice that the renamed attributes are only used in the method where they are declared, so it is probably better to declare them as method variables.

The `T(Date d)` method (881)

The same applies to the `m` attribute, which is being used to store a minutes attribute from the `date` instance, and the `R` attribute which is being used to store a maximum radius, if a circle were to be drawn in the frame. There is a slight problem here, that `m` is used for a different purpose, as an index in the for loop. This index needs to be renamed, by convention to `i`.

The `paint(Graphics g)` method (882)

Next let's consider the `E` method, which, given the modulo operation, must return a proportion of 2π , so an angle in radians. The first thing that can be done is to rename the attribute `a`, to make it clear that it represents an angle. The method can also be renamed to make its purpose clear.

The `E(int a, int u)` method (883)

Given that `Math.PI` is a standard constant in the Java language, it is a probably unnecessary to define a separate attribute to store this value in the `F` class. If

the `Math`. part of the static variable is too cumbersome, we could use a static import instead. Although this does couple the class to the definition of π in the Java SDK, this probably isn't going to change anytime soon.

Now consider the four methods `H(double)`, `H(double, double)`, `J(double)`, and `J(double, double)`, which have similar structures. Each method is used to calculate the length of an opposite or adjacent side of a triangle when given a length for hypotenuse proportionate to the radius of the circle, based on the `angle` attribute. Another purpose for this calculation is to obtain the x and y coordinates of a 2D vector represented as a length and an angle. So, we can rename these two pairs of methods appropriately.

The

`H(double y, double q)`
methods (884)

In addition, note that all the methods are coupled to the value of `angle`, which can be set externally to the method, or from within the method. This is unfortunate, because the flow of control is confused, and the value of `angle` may change outwith the methods that depend on it. It would be better to pass the `angle` attribute to each method and perform the calculation directly. The same is true of the `radius` attribute, however, this is used in combination with a constant, so must be left alone for the time being. This now makes the function of the for loop in the main `paint()` method much clearer. The loop is used to draw twelve lines on the `Graphics` object for the frame at twelve different angles.

The final method to examine is the `N()` method, whose only purpose is to draw a filled polygon. Initially, we can rename the input parameters, since we know how they will be used in the `calculateX()` and `calculateY()` methods.

The `N(Graphics g, double q, double s)`
method (885)

Next, we can see that the polygon has three points, one at the end of the specified vector and two very close to the origin (parameterised by the constant `U` and `angle` offset by `A`, so `N` is used to draw triangles of different lengths and different angles. Regarding `A`, it can be observed that it is set to $\pi/2$, or a quarter of circle. So, the offset expression for the final coordinate can be simplified to `angle - A`.

Returning to the `paint()` method reveals the purpose of the `N()` method. The method is used to draw three triangles at the appropriate angles for seconds, minutes and hours of the current date. The `N()` method and class can now both be renamed.

Finally, the method order can be reorganised and most made private. Some of the method calls used in the original application have been deprecated, so these are replaced as well. The revised version of the class is shown below.

The `paint(Graphics g)`
method (887)

```
import java.awt.*;
import java.util.*;

import static java.lang.Math.PI;

public class Clock extends Frame {

    private double offset = PI / 2, U = .05;

    private int radius;

    private Date date = new Date();
```

```

public void setDate(Date date) {
    this.date = date;
    repaint();
}

public void paint(Graphics g) {
    Color C = SystemColor.control;
    g.setColor(C);

    int width = getSize().width;
    int height = getSize().height;

    g.fillRect(0, 0, width, height);
    width -= 52;
    height -= 52;
    radius = Math.min(width / 2, height / 2);

    g.translate(width / 2 + 25, height / 2 + 36);
    g.setColor(C.darker());

    for (int i = 0; i < 12; ++i) {
        double angle = calculateAngle(i, 12);
        g.drawLine(
            calculateX(.8, angle),
            calculateY(.8, angle),
            calculateX(.9, angle),
            calculateY(.9, angle));
    }

    Calendar cal = Calendar.getInstance();
    cal.setTime(date);

    int hours = cal.get(Calendar.HOUR);
    int minutes = cal.get(Calendar.MINUTE);
    int seconds = cal.get(Calendar.SECOND);

    drawHand(g, calculateAngle(hours*60 + minutes, 720), .5);
    drawHand(g, calculateAngle(minutes, 60), .8);
    drawHand(g, calculateAngle(seconds, 60), .9);
}

private double calculateAngle(int numer, int denom) {
    return (3 * PI / 2 + 2 * PI * numer / denom) % (2 * PI);
}

private void drawHand(
    Graphics g, double angle, double length) {

    g.fillPolygon(
        new int[] {
            calculateX(length, angle),
            calculateX(U, angle + offset),
            calculateX(U, angle - offset)},
        new int[] {
            calculateY(length, angle),
            calculateY(U, angle + offset),
            calculateY(U, angle - offset)},
        3
    );
}

```

```

}

private int calculateX(double length, double angle) {
    return (int) (radius * length * Math.cos(angle));
}

private int calculateY(double length, double angle) {
    return (int) (radius * length * Math.sin(angle));
}

public static void main(String[] _) throws Exception {
    Clock c = new Clock();
    c.resize(200, 200);
    c.show();
    while(true) {
        c.setDate(new Date());
        Thread.sleep(200);
    }
}
}

```



Following the comprehension and refactoring process, we discover that the application is a clock! When you have finished, consider if any other refactorings might be appropriate. Prepare a refactoring plan for these and apply them to the file.