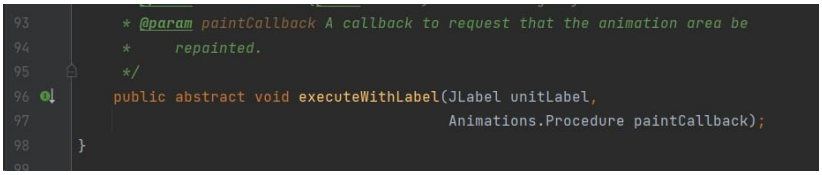
Design Patterns

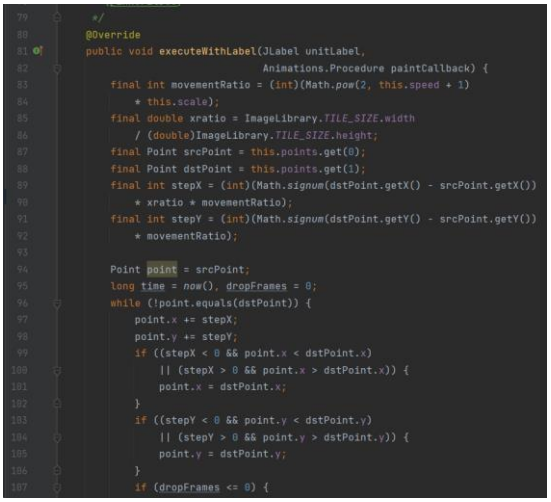
Design Pattern 1 – Template Method Pattern

Este padrão de design permite definir a estrutura de um algoritmo numa classe base, enquanto as subclasses façam as implementações específicas dos métodos dessa interface. Um exemplo no código do freeCol para este design pattern é na classe base Animation(net/sf/freecol/client/gui/animation/Animation) e nas sublclasses (net/sf/freecol/client/gui/animation/UnitImageAnimation e net/sf/freecol/client/gui/animation/UnitMoveAnimation) já que cada uma destas subclasses implementa o método executeWithLabel() de forma distinta nas diferentes subclasses.



```
     * @param paintCallback A callback to request that the animation area be
     *     repainted.
     */
    public abstract void executeWithLabel(JLabel unitLabel,
                                          Animations.Procedure paintCallback);
}
```

Animation Class



```
     */
    @Override
    public void executeWithLabel(JLabel unitLabel,
                                 Animations.Procedure paintCallback) {
        final int movementRatio = (int)(Math.pow(2, this.speed + 1)
            * this.scale);
        final double xratio = ImageLibrary.TILE_SIZE.width
            / (double)ImageLibrary.TILE_SIZE.height;
        final Point srcPoint = this.points.get(0);
        final Point dstPoint = this.points.get(1);
        final int stepX = (int)(Math.signum(dstPoint.getX() - srcPoint.getX())
            * xratio * movementRatio);
        final int stepY = (int)(Math.signum(dstPoint.getY() - srcPoint.getY())
            * movementRatio);

        Point point = srcPoint;
        long time = now(), dropFrames = 0;
        while (!point.equals(dstPoint)) {
            point.x += stepX;
            point.y += stepY;
            if ((stepX < 0 && point.x < dstPoint.x)
                || (stepX > 0 && point.x > dstPoint.x)) {
                point.x = dstPoint.x;
            }
            if ((stepY < 0 && point.y < dstPoint.y)
                || (stepY > 0 && point.y > dstPoint.y)) {
                point.y = dstPoint.y;
            }
            if (dropFrames <= 0) {
```

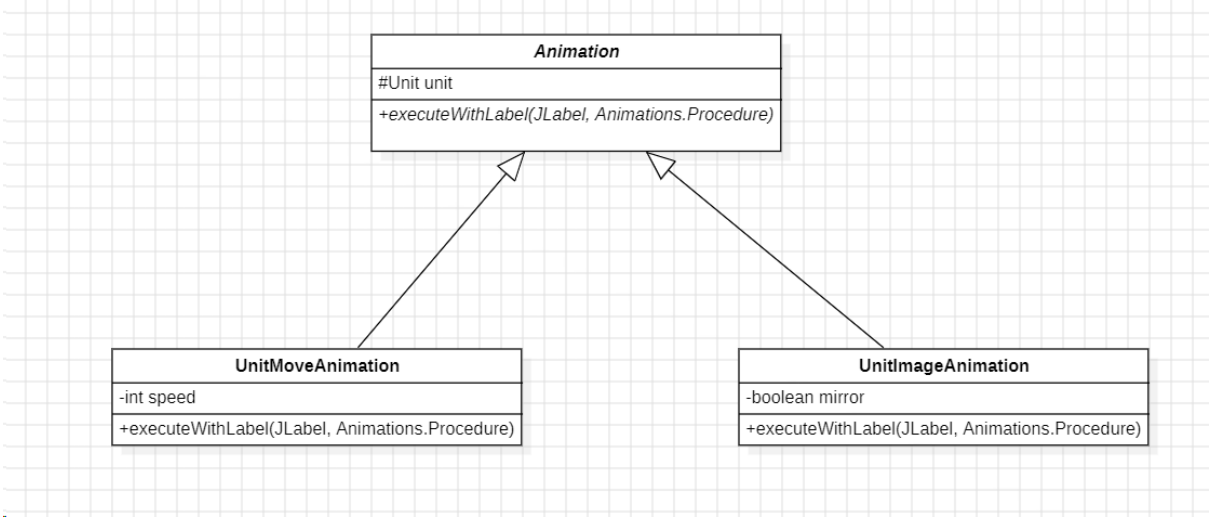
UnitMoveAnimation subclasse

```java
    */
    public void executeWithLabel(JLabel unitLabel,
                                 Animations.Procedure paintCallback) {
        final ImageIcon icon = (ImageIcon)unitLabel.getIcon();

        // Step through the animation, chaning the image
        for (AnimationEvent event : animation) {
            long time = System.nanoTime();
            if (event instanceof ImageAnimationEvent) {
                final ImageAnimationEvent ievent = (ImageAnimationEvent)event;
                Image image = ievent.getImage();
                if (mirror) {
                    // FIXME: Add mirroring functionality to SimpleZippedAnimation
                    image = createMirroredImage(image);
                }
                icon.setImage(image);
                paintCallback.execute(); // paint now

                // Time accounting
                time = ievent.getDurationInMs()
                    - (System.nanoTime() - time) / 1000000;
                if (time > 0) Utils.delay(time, warning: "Animation delayed.");
            }
        }
    }
}
```

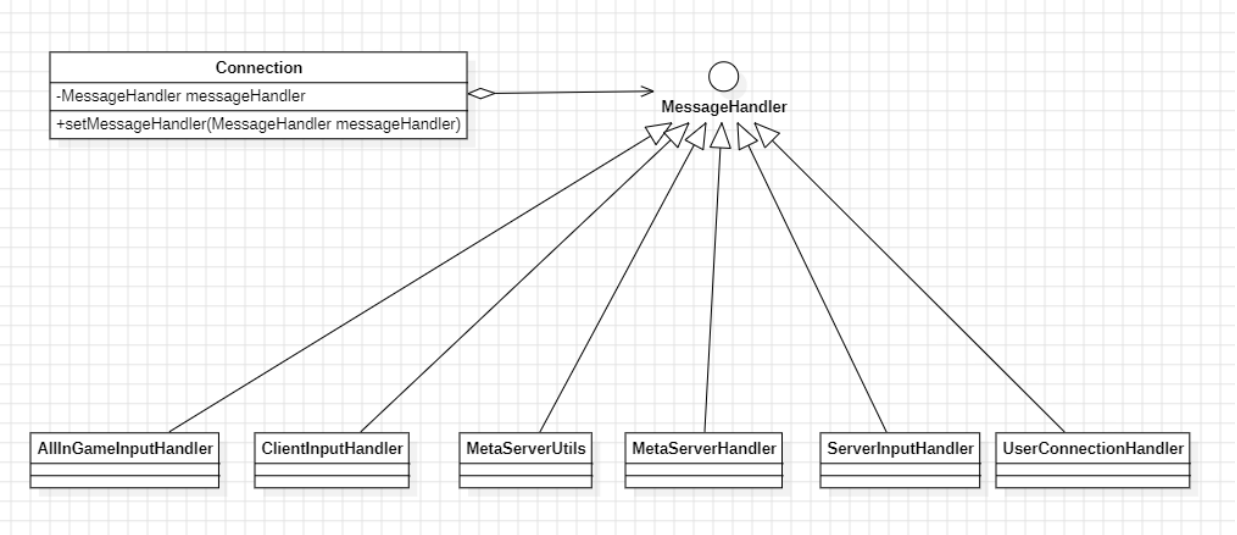
UnitImageAnimation subclasse



Design Pattern 2 – State Pattern

Este padrão de design é usado para modelar objetos que podem ter diferentes estados e permite que que esses objetos mudem de estado durante a execução do programa. Um exemplo deste padrão é a interface MessageHandler (net/sf/freecol/common/networking/MessageHandler) e os diferentes estados, nomeadamente, AllInGameInputHandler(net/sf/freecol/server/ai/AIInGameInputHandler), ClientInputHandler(net/sf/freecol/client/control/ClientInputHandler), MetaServerUtils(net/sf/freecol/common/metaserver/MetaServerUtils), MetaServerHandler(net/sf/freecol/metaserver/MetaServerHandler), ServerInputHandler(net/sf/freecol/server/control/ServerInputHandler) e UserConnectionHandler(net/sf/freecol/server/control/UserConnectionHandler).

Sendo este estado messageHandler modificado em Connection (net/sf/freecol/common/networking/Connection).



Design Pattern 3 – Façade pattern

Este padrão de design fornece uma interface unitária para um conjunto de classes, simplificando a interação com o subsistema, ocultando a complexidade interna e fornecendo uma única entrada para o cliente. O exemplo deste padrão é a interface Selector(net/sf/freecol/common/i18n/Selector), que é implementada pelas classes: DefaultNumberRule (net/sf/freecol/common/i18n/DefaultNumberRule), DualNumberRule(net/sf/freecol/common/i18n/DualNumberRule) e OtherNumberRule(net/sf/freecol/common/i18n/OtherNumberRule), por exemplo. Depois na classe Messages(net/sf/freecol/common/i18n/Messages) tem 0 ou mais Selectors.