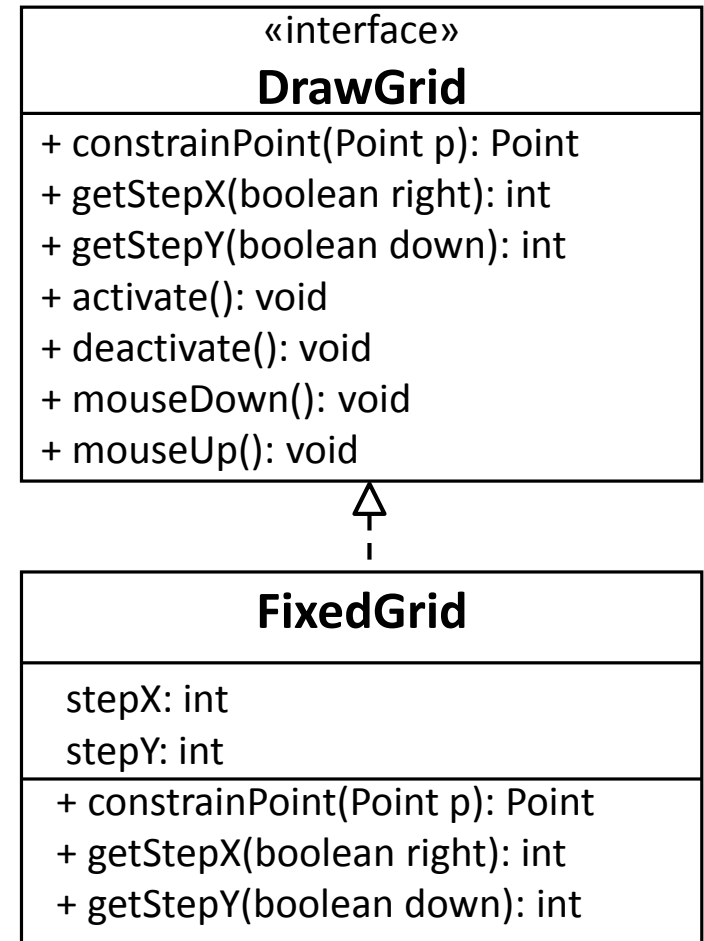


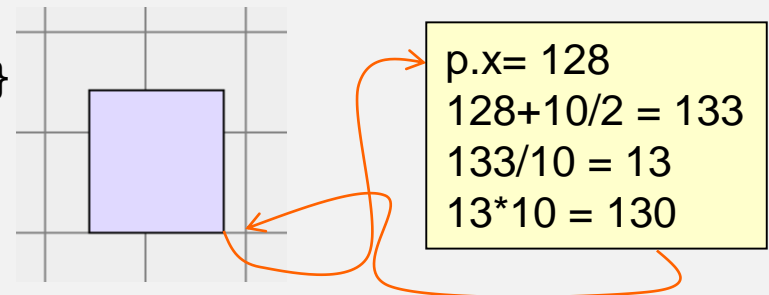
Assignment 5 : Snap to Grid

- Snap To Grid
- Snap To Nearest



Snap to Grid

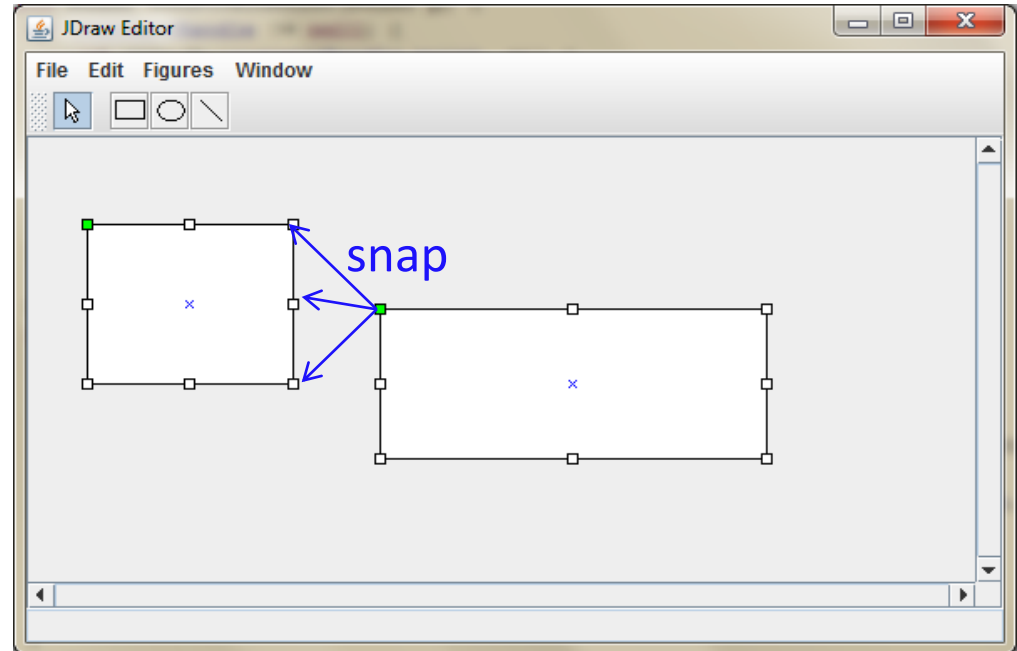
```
public class FixedGrid implements DrawGrid {  
    private final int stepX, stepY;  
    public FixedGrid(int sx, int sy) {  
        this.stepX = Math.max(1, sx);  
        this.stepY = Math.max(1, sy);  
    }  
    @Override public Point constrainPoint(Point p) {  
        int x = ((p.x + stepX/2) / stepX) * stepX;  
        int y = ((p.y + stepY/2) / stepY) * stepY;  
        return new Point(x, y);  
    }  
    @Override public int getStepX(boolean right) { return stepX; }  
    @Override public int getStepY(boolean down) { return stepY; }  
    @Override public void activate() {}  
    @Override public void deactivate() {}  
    @Override public void mouseDown() {}  
    @Override public void mouseUp() {}  
}
```



Snap to Nearest Object

- **Idea**

- Snap to the positions of near-by handles instead of to fixed grid points



- **3 Cases**

- Existing figure is changed over the handles: `Handle.dragInteraction()`
- New figure is defined: `Tool.dragInteraction()`
- Figure (or whole selection) is moved: `move(...)`

Case 1: Existing figure is changed over handles

```
public class SnapGrid implements DrawGrid {  
    private static final int SNAP = 15;  
    private final DrawView view;  
    public SnapGrid(DrawView view) { this.view = view; }  
  
    public Point constrainPoint(Point p) {  
        for (Figure f : view.getModel().getFigures()) {  
            if (!view.getSelection().contains(f))  
                for (FigureHandle h : f.getHandles()) {  
                    Point pos = h.getLocation();  
                    if (nearBy(p, pos)) { return pos; }  
                }  
        }  
        return p;  
    }  
  
    private boolean nearBy(Point p, Point q) {  
        return p.distance(q) < SNAP;  
    }  
}
```

All figures in the model, except the figures in the selection

Check handles of each figure. If a handle is near-by, then its position is returned

Case 2: New figure is created over tool

- **Problem**

- New figure is not yet in the selection, so it "snaps" to itself
=> We have to exclude the created figure
=> Register an observer in the model (in ctor, activate or mouseDown)

```
private final DrawModelListener listener;  
private Figure figure;  
  
public SnapGrid(DrawView view) {  
    this.view = view;  
    listener = e -> {  
        if (e.getType() == DrawModelEvent.Type.FIGURE_ADDED) {  
            figure = e.getFigure();  
        }  
    };  
}
```

figure reference
must be cleared in
method mouseUp()

Case 2: New figure is created over tool

- **Problem**

- New figure is not yet in the selection, so it "snaps" to itself
=> We have to exclude the created figure
=> Register an observer in the model (in ctor, activate or mouseDown)

```
public void activate() {  
    view.getModel().addModelChangeListener(listener);  
}  
  
public void deactivate() {  
    view.getModel().removeModelChangeListener(listener);  
}  
  
public Point constrainPoint(Point p) {  
    for (Figure f : view.getModel().getFigures()) {  
        if (!view.getSelection().contains(f) && f != figure) {  
            ...  
        }  
    }  
}
```

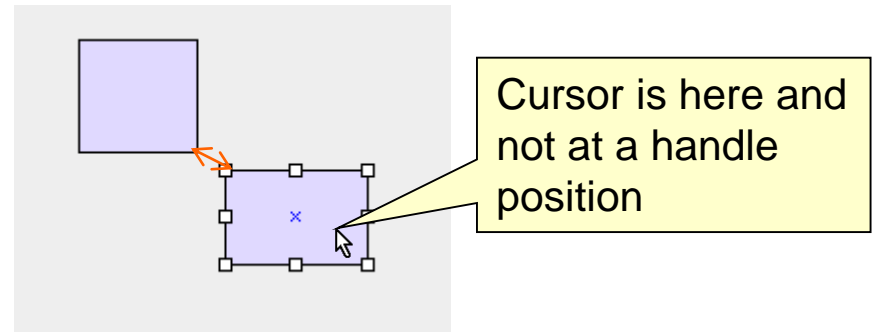
Only snap to
figures which are
not selected

Only snap to
figures which are
not being created

Case 3: Figures are moved with the mouse

- **Problem**

- The coordinates of the mouse are not at a handle position but anywhere inside the figure



- Idea: compare all handles of the selected figures with the handles of the other figures
 - if such a delta is small enough, then add this delta to the current mouse position
 - Problem: on the next call of constrainPoint, the delta is zero and the mouse coordinates are not changed, i.e. the figure jumps back

Case 3: findNearHandleOf

```
private FigureHandle findNearHandleOf(FigureHandle h) {  
    for (Figure f : view.getModel().getFigures()) {  
        if (!view.getSelection().contains(f)) {  
            for (FigureHandle nh : f.getHandles()) {  
                if (nearBy(h.getLocation(), nh.getLocation())) {  
                    return nh;  
                }  
            }  
        }  
    }  
    return null;  
}
```

for loop checks
whether any handle
nh of figure **f** is near-
by the given handle **h**

Iterates over all
figures which
are not selected

- Looks for a handle which is near-by the handle **h** which is passed as an argument to method `findNearHandleOf`
- If no such handle is found, null is returned

Case 3: constrainPoint

```
private Point p0; // snapped/modified mouse coordinates
private boolean snapped = false;
public Point constrainPoint(Point p) {
    if (snapped) {
        if (nearBy(p0, p)) { return p0; }
        else { snapped = false; return p; }
    }
    for (Figure s : view.getSelection()) {
        for (FigureHandle h : s.getHandles()) {
            FigureHandle nearHandle = findNearHandleOf(h);
            if (nearHandle != null) { snapped = true;
                int dx = nearHandle.getLocation().x-h.getLocation().x;
                int dy = nearHandle.getLocation().y-h.getLocation().y;
                return p0 = new Point(p.x + dx, p.y + dy);
            }
        }
    }
    return p;
}
```

Prevents jumping
back after a snap
to a handle

Iterates over all
figures which are
moved (does not
work for new figures)

If a near-by handle is found, then p0 is
moved by the corresponding delta and
the new point is stored in p0

Remarks

- **Is interface DrawGrid powerful enough?**
 - No, access to the model and to the selection is missing
 - Missing references can be passed to a concrete grid over its constructor
 - No, kind of operation is not known in the grid
 - If a new figure is created, it is not known which is this figure, a possible workaround is to register a listener
- **Is DrawGrid an application of State or Strategy?**