# Submission Notes for Homework 4

### Julia Schwarz

Implemented one-way constraints using my own algorithm.

**Test Programs:**

1. TestHomework4: the TestHomework4 class with a few more basic additions, showing how centering of objects can be implemented.
2. TestNotifyPropertyChanged: Tests the property changed listener system (see below) I implemented as part of constraints.
3. NodeAndEdgeEditor: A basic node and edge editor, as per the required document.
4. MyTestValueRect: Implements and tests a “ValueRect” (as mentioned in the assignment) which has a value it displays as a string. Changing the slider value changes the ValueRect’s value as well as line thickness. The slider is also implemented using constraints.
5. MyTestConstraints: A set of sliders that modify a rectangle. Also shows how a rectangle in the center always remains centered. The centering, slider modification, and actual sliders are all implemented via constraints.

**Other Notes:**

1. All code related to constraints is in the homework4.android.constraints namespace.
2. A significant portion of the code relies on the assumption that all GraphicalObject properties are set via a “setPropertyName” method, and accessed via a “getPropertyName” method. This allows me to easily reference getter and setter methods, and listen to property change values, using only the property name.
3. The constraint classes are relatively well commented.
4. There are three main components to my constraints architecture. My aim in designing the constraint system was to keep the architecture as simple as possible to implement but also make it generic, so that constraints could operate not only on properties of graphical objects but on arbitrary values:
   1. An **architecture for listening to changes on arbitrary object properties**, inspired by the DependencyProperty architecture in WPF. Any object can listen to changes on property values by setting up a PropertyChangedListener, specifying the name of the property to listen to. Any property that can be listened to must call “NotifyPropertyChanged(oldValue, newValue)”, which is defined in GraphicalObjectBase. A basic test of the architecture is in **MyTestNotifyPropertyChanged.**
   2. A notion of a **generic** **Variable**. Constraints operate on variables. These variables can be of arbitrary type and represent arbitrary information, for example a timer value (for animation), a HTTP status, or a mouse position. Variables must implement getters and seters. Additionally, I implemented a generic GraphicalObjectProperty variable, which uses Reflection to tie any property of a graphical object to a variable merely by using its name. **Each variable has a list of constraints that depend on it.** That is, a list of constraints that must be updated when the variable value changes.
   3. A set of **Constraints** which operate on Variables. Constraints change variable values, which in turn change values of the underlying data (GraphicalObject properties or otherwise). When a variable values changes, all constraints that depend on that variable value get re-evaluated. It is up to the Constraint class to properly set up this dependency. Constraints may arbitrarily specify dependencies. Additionally, constraints may implement arbitrary code in the evaluate() function that will set the properties of variables when the constraint is evaluated. Additionally, constraints are type safe, they specify the type of variables they can work on via generics. Example constraints are implemented in the code, the simplest/recommended constraints to look at are **EqualConstraint**, and **IntSumConstraint** (as well as **IntProductConstraints**). Constraints can be active and inactive. When constructed, consraints are inactive. A constraint is only in effect when it is active. Activate() and deactivate() change active state.
5. ValueRect (in homework4.android.graphicalobject) shows an example of how to create constraints in new classes.