Reactive GWT

This library supports reactive programming in a way that integrates nicely with Google Web Toolkit.

AbstractHasValue<T>

The standard GWT way to implement HasValue is to extend Widget (usually Composite), which provides `fireEvent(GwtEvent<?>):void` (required by HasHandlers, a supertype of HasValue). If you want to design a widget that has more than one piece of observable state, you're out of luck, because your widget can only implement HasValue once.

```
// Sorry, no can do.
class StatefulWidget extends Composite
    implements HasValue<Integer>,
               HasValue<String> {
```

An alternative is to let the HasValue implementations be components of your widget:

```
class StatefulWidget extends Composite {
  HasValue<Integer> getIntegerHasValue() { ... }
  HasValue<String> getStringHasValue() { ... }
```

Reactive GWT provides **AbstractHasValue** to use for these HasValue implementations. AbstractHasValue also implements Signal.

Signal<T>

A **Signal** is a variable observable value. Signal<T> extends:

- Source<T> (for retrieving the signal's current value)
- HasValueChangeHandlers<T> (for observing changes to the signal's value)

This is very similar to HasValue. The difference is that the Signal interface does *not* not have methods that mutate its state. This allows us to derive signals from other signals. The main purpose of Reactive GWT is to support Signal composition, because it is an incredibly useful and intuitive concept,

Example 1: Boolean signal composition

Suppose you have a CheckBox, three TextBox widgets, and a Label. The label needs to be visible only when the checkbox is checked and at least one of the text boxes is nonempty. This can be expressed naturally by declaring the label's visibility change as a reaction to the composite state of the other widgets:

```
import static Signals.*;
...
and(
  valueOf(checkBox),
  or(
    not(emptyString(valueOf(textBox1))),
    not(emptyString(valueOf(textBox2))),
    not(emptyString(valueOf(textBox3)))
  )
).addValueChangeHandler(ValueChangeHandlers.setVisible(label));
```

Example 2: String signal composition

Example 1 demonstrated some of the commonly-used composition operations supported by Reactive GWT, but you can also compose values using your own functions. Suppose you want the label to always display the contents of the nonempty text boxes, separated by commas:

```
Signals.compose(
  ImmutableList.of(
    Signals.valueOf(textBox1),
    Signals.valueOf(textBox2),
    Signals.valueOf(textBox3)
  ),
  new Function<Iterable<? extends String>, String>() {
    public String apply(Iterable<? extends String> values) {
      return Joiner.on(", ").join(removeEmptyStrings(values));
    }
    Iterable<? extends String> removeEmptyStrings(Iterable<? extends String> values) {
      return Iterables.filter(values, new Predicate<String>() {
        public boolean apply(String value) {
          return value.length() != 0;
        }
      });
    }
  }
).addValueChangeHandler(ValueChangeHandlers.setText(label));
```