```java
package hyperDap.base.types.dataSet;

import java.util.ArrayList;

/**
 * An abstract subclass of {@link DataSet} that allows marking entries as valid ({@code true} or
 * invalid ({@code false}.
 *
 * @author soenk
 *
 * @param <T>
 */

public abstract class ValidityDataSet<T> extends DataSet<T> {

  protected ArrayList<Boolean> valids;

  public ValidityDataSet(Number base, Number step) {
    super(base, step);
    this.valids = new ArrayList<Boolean>();
  }

  // helpers
  //
  ********************************************************************************
  *********************************

  /**
   * Ensures that internally the validites and values align correctly. As the two properties are
   * stored in separate {@link ArrayList ArrayLists} this method will ensure they are of the same
   * length. If the validity list must be extended it is extended with {@code false}.
   *
   * @category helper
   * @return {@code true} if changes had to be made, {@code false} otherwise.
   */
  public boolean cleanLength() {
    int a = this.values.size();
    int b = this.valids.size();
    if (a == b) {
      return false;
    }
    if (a > b) {
      for (int i = b; i < a; i++) {
        this.valids.add(false);
      }
      return true;
    }
    for (int i = a; i < b; i++) {
      this.valids.remove(a);
    }
    return true;
  }

  // add
  //
  ********************************************************************************
  ************************************

  /**
   * {@inheritDoc}
   * <p>
   * New entries are marked as valid.
   */
  @Override
  public void add(int index, T value) {
    int i = this.values.size();
    super.add(index, value);
    if (i <= index) {
      while (i <= index) {
        this.valids.add(true);
```

```
67            i++;
68          }
69        } else {
70          this.valids.set(index, true);
71        }
72      }
73
74      /**
75       * {@inheritDoc}
76       * <p>
77       * The new entry is marked as valid.
78       */
79      @Override
80      public boolean add(T value) {
81        if (super.add(value) == true) {
82          if (this.valids.add(true) == true) {
83            return true;
84          }
85          this.values.remove(this.values.size() - 1);
86        }
87        return false;
88      }
89
90      /**
91       * {@inheritDoc}
92       * <p>
93       * The new entry iis marked as valid.
94       */
95      @Override
96      public double addValue(T value) {
97        this.valids.add(true);
98        return super.addValue(value);
99      }
100
101     // get
102     //
        ************************************************************************************
        ************************************
103
104     /**
105      * Check whether an value is considered valid or not.
106      *
107      * @param index The index of the value to be checked
108      * @return The validity of the entry at position {@code index}
109      * @throws IndeOutOfBoundsException When there is no such value
110      */
111     public boolean getValidByIndex(int index) throws IndexOutOfBoundsException {
112       if (index < 0 || index >= this.size()) {
113         throw new IndexOutOfBoundsException();
114       }
115       try {
116         return this.valids.get(index);
117       } catch (IndexOutOfBoundsException e) {
118         this.cleanLength();
119       }
120       return this.valids.get(index);
121     }
122
123     /**
124      * Check whether the value corresponding to this xValue is valid.
125      *
126      * @param independentValue The {@code xValue}
127      * @return The validity of the value stored under this xValue
128      * @throws IndeOutOfBoundsException When there is no such value
129      */
130     public boolean getValid(double independentValue) throws IndexOutOfBoundsException {
131       return this.getValidByIndex(this.getIndex(independentValue));
132     }
133
134     /**
135      * {@link Number} encapsulation of {@link #getValid(double)}.
136      *
137      * @param independentValue The {@code xValue}
```

```
138       * @return The validity of the value stored under this {@code xValue}
139       * @throws IndeOutOfBoundsException When there is no such value
140       */
141      public boolean getValid(Number independentValue) throws IndexOutOfBoundsException {
142        return this.getValid(independentValue.doubleValue());
143      }
144
145      // edit validity
146      //
         ********************************************************************************
         ************************************
147
148      /**
149       * Edit whether a value is considered valid or not.
150       * <p>
151       * If needed {@link #cleanLength()} is called.
152       *
153       * @param index The index of the value
154       * @param validity Whether the value should be valid ({@code true}) or invalid
          ({@code false})
155       * @return If this Set was altered as a result of this operation ({@code true}) or
          not
156       *         ({@code false})
157       * @throws IndexOutOfBoundsException if there is no corresponding value, after
          calling
158       *         {@link #cleanLength()} first.
159       */
160      public boolean editValidityByIndex(int index, boolean validity) throws
         IndexOutOfBoundsException {
161        if (index < 0 || index >= this.size()) {
162          throw new IndexOutOfBoundsException();
163        }
164        this.cleanLength();
165        return this.valids.set(index, validity);
166      }
167
168      /**
169       * Edit whether a value is considered valid or not.
170       * <p>
171       * If needed {@link #cleanLength()} is called.
172       *
173       * @param index The {@code xValue} this value is stored under
174       * @param validity Whether the value should be valid ({@code true}) or invalid
          ({@code false})
175       * @return If this Set was altered as a result of this operation ({@code true}) or
          not
176       *         ({@code false})
177       * @throws IndexOutOfBoundsException if there is no corresponding value, after
          calling
178       *         {@link #cleanLength()} first.
179       */
180      public boolean editValidity(double xValue, boolean validity) throws
         IndexOutOfBoundsException {
181        return this.editValidityByIndex(this.getIndex(xValue), validity);
182      }
183
184      /**
185       * A {@link Number} encapsulation of {@link #editValidity(double, boolean)}.
186       * <p>
187       * If needed {@link #cleanLength()} is called.
188       *
189       * @param index The {@code xValue} this value is stored under
190       * @param validity Whether the value should be valid ({@code true}) or invalid
          ({@code false})
191       * @return If this Set was altered as a result of this operation ({@code true}) or
          not
192       *         ({@code false})
193       * @throws IndexOutOfBoundsException if there is no corresponding value, after
          calling
194       *         {@link #cleanLength()} first.
195       */
196      public boolean editValidity(Number xValue, boolean validity) throws
         IndexOutOfBoundsException {
```

```
      return this.editValidity(xValue.doubleValue(), validity);
    }

    // other
    //
    ************************************************************************************
    ************************************

    /**
     * {@inheritDoc}
     */
    @Override
    public void clear() {
      super.clear();
      this.valids.clear();
    }

    /**
     * {@inheritDoc}
     */
    @Override
    public void ensureCapacity(int capacity) {
      super.ensureCapacity(capacity);
      this.valids.ensureCapacity(capacity);
    }

  }

```