src\test\java\shelf\ShelfTest.java

```java
package shelf;

import org.junit.*;
import static org.junit.Assert.*;

import java.util.Iterator;
import java.util.NoSuchElementException;

public class ShelfTest {

  private Book[] books;
  private Shelf<Book> bookShelf;

  @Before
  public void initializeBooks(){
    books = new Book[4];
    books[0] = new Book("Ullenbloom", "Java ist auch eine Insel", 1246);
    books[1] = new Book("Schuld", "Ferdinand von Schirach", 208);
    books[2] = new Book("Bibi und Tina: Pferdeabenteuer am Meer", "Matthias von Bornstädt"
, 34);
    books[3] = new Book("Alice im Wunderland", "Lews Carroll", 1246);
    bookShelf = new Shelf<>();

    bookShelf.setUpperLeft(books[0]);
    bookShelf.setUpperRight(books[1]);
    bookShelf.setLowerLeft(books[2]);
    bookShelf.setLowerRight(books[3]);
  }

  @Test
  public void shelfConstructorInitialize() {
    Shelf<ShelfItem> shelf = new Shelf<ShelfItem>();
    assertNull(shelf.getUpperLeft());
    assertNull(shelf.getUpperRight());
    assertNull(shelf.getLowerLeft());
    assertNull(shelf.getLowerRight());
  }

  @Test
  public void shelfSetIndexPosition() {
    Shelf<Book> shelf = new Shelf<>();

    for (int i = 0; i < 4; i++){
      shelf.set(i, books[i]);
    }

    assertSame(books[0], shelf.getUpperLeft());
    assertSame(books[1], shelf.getUpperRight());
    assertSame(books[2], shelf.getLowerLeft());
    assertSame(books[3], shelf.getLowerRight());

  }

  @Test(expected=IllegalArgumentException.class)
  public void shelfSetIndexTooLargeIndex(){
    Shelf<Book> shelf = new Shelf<>();
    shelf.set(4, books[0]);
```

```
57    }
58
59    @Test(expected=IllegalArgumentException.class)
60    public void shelfSetIndexNegativeIndex(){
61      Shelf<Book> shelf = new Shelf<>();
62      shelf.set(-1, books[0]);
63    }
64
65
66    @Test
67    public void shelfGetIndexPosition() {
68      Shelf<Book> shelf = new Shelf<>();
69
70      shelf.setUpperLeft(books[0]);
71      shelf.setUpperRight(books[1]);
72      shelf.setLowerLeft(books[2]);
73      shelf.setLowerRight(books[3]);
74
75      for (int i = 0; i < 4; i++){
76        assertSame(books[i], shelf.get(i));
77      }
78    }
79
80    @Test(expected=IllegalArgumentException.class)
81    public void shelfGetIndexTooLargeIndex(){
82      Shelf<Book> shelf = new Shelf<>();
83      shelf.get(4);
84    }
85
86    @Test(expected=IllegalArgumentException.class)
87    public void shelfGetIndexNegativeIndex(){
88      Shelf<Book> shelf = new Shelf<>();
89      shelf.get(-1);
90    }
91
92    @Test
93    public void shelfIterator(){
94      Shelf<Book> shelf = new Shelf<>();
95
96      books[2] = null;
97
98      for (int i = 0; i < 4; i++){
99        shelf.set(i, books[i]);
100     }
101
102     int i = 0;
103     for (Book book : shelf){
104       assertSame(books[i++], book);
105     }
106
107   }
108
109   @Test
110   public void shelfIteratorBeyondLast() {
111     Shelf<Book> shelf = new Shelf<>();
112     Iterator<Book> it = shelf.iterator();
113     for (int i = 0; i < 4; i++){
114       assertTrue(it.hasNext());
115       it.next();
116     }
```

```
117
118       assertFalse(it.hasNext());
119     }
120
121     @Test(expected=NoSuchElementException.class)
122     public void shelfIteratorBeyondLastException() {
123       Shelf<Book> shelf = new Shelf<>();
124       Iterator<Book> it = shelf.iterator();
125       for (int i = 0; i < 4; i++){
126           it.next();
127       }
128
129       it.next();
130     }
131
132     @Test
133     public void shelfTakeFrom(){
134       Shelf<Book> shelf = new Shelf<Book>();
135       shelf.takeFrom(bookShelf);
136
137       for (int i = 0; i < 4; i++)
138         assertSame(books[i], shelf.get(i));
139
140       for (int i = 0; i < 4; i++)
141         assertNull(bookShelf.get(i));
142     }
143
144     @Test(expected=IllegalArgumentException.class)
145     public void shelfTakeFromNull() {
146       bookShelf.takeFrom(null);
147     }
148
149     @Test
150     public void shelfMax() {
151       bookShelf.set(2,null);
152
153       Book max = bookShelf.max(
154           (b1,b2) -> Integer.compare(b1.getPages(),b2.getPages()));
155
156       assertTrue(max == books[0] || max == books[3]);
157     }
158
159     @Test(expected=IllegalArgumentException.class)
160     public void shelfMaxNull() {
161       bookShelf.max(null);
162     }
163
164     @Test
165     public void shelfTransferAndTrim() {
166       bookShelf.set(1,null);
167       bookShelf.set(2,null);
168
169       Shelf<Book> shelf = new Shelf<>();
170
171       Shelf.transferAndTrim(bookShelf, shelf);
172
173       assertSame(books[0], shelf.get(0));
174       assertSame(books[3], shelf.get(1));
175       assertSame(null, shelf.get(2));
176       assertSame(null, shelf.get(3));
```

```
177
178       assertNull(bookShelf.get(0));
179       assertNull(bookShelf.get(1));
180       assertNull(bookShelf.get(2));
181       assertNull(bookShelf.get(3));
182     }
183
184     @Test
185     public void shelfTransferAndTrimNonEmpty() {
186
187       Shelf<Book> shelf = new Shelf<>();
188
189       shelf.set(2, books[1]);
190       shelf.set(3, books[2]);
191
192       bookShelf.set(1,null);
193       bookShelf.set(2,null);
194
195       Shelf.transferAndTrim(bookShelf, shelf);
196
197       assertSame(books[0], shelf.get(0));
198       assertSame(books[3], shelf.get(1));
199       assertSame(null, shelf.get(2));
200       assertSame(null, shelf.get(3));
201     }
202
203     @Test(expected=IllegalArgumentException.class)
204     public void shelfTransferAndTrimFromNull() {
205       Shelf.transferAndTrim(null, bookShelf);
206     }
207
208     @Test(expected=IllegalArgumentException.class)
209     public void shelfTransferAndTrimToNull() {
210       Shelf.transferAndTrim(bookShelf, null);
211     }
212
213 }
214
215
```