```java
package manager;

import java.time.LocalDate;

import app.Status;
import todo.TodoEntry;

public class TodoManager {

    private TodoListEntry first;
    private int nextFreeId = 0;

    /**
     * Returns a new unique ID (increasing number) that can be used to create a new
     * todoentry
     *
     * @return
     */
    public int getNewId() {
        this.nextFreeId++;
        return this.nextFreeId - 1;

    }

    /**
     * Finds a Todoentry by its id. returns null, if none was found
     *
     * @param id
     * @return
     */
    public TodoEntry findById(int id) {
        TodoListEntry entry = this.first;
        while (entry != null) {
            TodoEntry todoData = (TodoEntry) entry.getData();
            if (todoData.getId() == id) {
                return todoData;
            }
            entry = entry.getNext();
        }
        return null;
    }

    /**
     * Adds a new list entry to the list and returns the list entry
     *
     * @param te
     * @return
     */
    public TodoListEntry add(TodoEntry te) {

        // first find where to add the element based on time
        TodoListEntry entry = this.first;
        TodoListEntry newEntry = new TodoListEntry(null, null, te);
        // if it is the first element, just save it and stop
        if (this.first == null) {
            this.first = newEntry;

            return this.first;
        }
```

```
60
61          while (entry != null) {
62
63              TodoEntry todoData = (TodoEntry) entry.getData();
64              // element is the very first in the list based on due date
65              if (te.getDueTo().isBefore(todoData.getDueTo()) ||
te.getDueTo().isEqual(todoData.getDueTo())) {
66                  // first set the prev and next of the new element
67                  newEntry.setPrevious(entry.getPrevious());
68                  newEntry.setNext(entry);
69                  // then set the next of the previous if there is a previous
70                  if (entry.getPrevious() != null) {
71                      entry.getPrevious().setNext(newEntry);
72                  }
73                  // then set the previous of the current
74                  entry.setPrevious(newEntry);
75
76                  // if it is before the formerly first element we have to
reset the first
77                  if (this.first.equals(entry)) {
78                      this.first = newEntry;
79                  }
80
81                  return newEntry; // end loop and return newEntry
82                  // case that it is the last element in the list
83              } else if (entry.getNext() == null) {
84                  newEntry.setPrevious(entry);
85                  entry.setNext(newEntry);
86                  return newEntry;
87              }
88              entry = entry.getNext();
89          }
90          // fallback, if we do not hit a return clause in the loop, which is
impossible..
91          // but java needs it
92          return newEntry;
93      }
94
95      /**
96       * Returns number of todo entries until a given date, with a given
status. NULL
97       * values means everything is fetched for this criteria
98       *
99       * @param until
100      * @param status
101      * @return
102      */
103     int count(LocalDate until, Status status) {
104         int count = 0;
105         TodoListEntry entry = this.first;
106         while (entry != null) {
107             TodoEntry todoData = (TodoEntry) entry.getData();
108             // first check for date constraint
109             if (until == null || todoData.getDueTo().isBefore(until) ||
todoData.getDueTo().isEqual(until)) {
110                 if (status == null || todoData.getStatus() == status) {
111                     count++;
112                 }
113             }
114             entry = entry.getNext();
```

```
115         }
116         return count;
117     }
118 
119     /**
120      * Returns all TodoEntrys until a given date, with a given status. NULL
    values
121      * means everything is fetched for this criteria
122      *
123      * @param until
124      * @param status
125      * @return
126      */
127     public TodoEntry[] get(LocalDate until, Status status) {
128         TodoEntry[] foundEntries = new TodoEntry[this.count(until, status)];
    // create array of necessary size
129         TodoListEntry entry = this.first;
130         int count = 0; // helper variable because of static array that is
    intialized beforehand ^^
131         while (entry != null) {
132             TodoEntry todoData = (TodoEntry) entry.getData();
133             // first check for date constraint
134             if (until == null || todoData.getDueTo().isBefore(until) ||
    todoData.getDueTo().isEqual(until)) {
135                 if (status == null || todoData.getStatus() == status) {
136                     foundEntries[count] = todoData;
137                     count++;
138                 }
139             }
140             entry = entry.getNext();
141 
142         }
143 
144         return foundEntries;
145     }
146 
147     /**
148      * Removes all entires until a certain date and returns number of deleted
    items
149      *
150      * @param until all items to this date will be deleted
151      * @return number of deleted items
152      */
153     public int removeUnitl(LocalDate until) {
154         TodoListEntry entry = this.first;
155         int count = 0; // helper variable because we return number of removed
    items
156 
157         while (entry != null && (((TodoEntry)
    entry.getData()).getDueTo().isBefore(until)
158                 || ((TodoEntry) entry.getData()).getDueTo().isEqual(until)))
    {
159             entry.getNext().setPrevious(null); // set the previous of item
    after the deleted one to 0
160             this.first = entry.getNext(); // set a new first
161             count++;
162             entry = entry.getNext();
163 
164         }
165         return count;
```

```
166     }
167
168     /**
169      * Returns the last entry in the list
170      *
171      * @return
172      */
173     public TodoListEntry getLast() {
174         TodoListEntry entry = this.first;
175         while (entry.getNext() != null) {
176             entry = entry.getNext();
177         }
178         return entry;
179     }
180 }
```