```
package app;

import java.time.LocalDate;

import inout.In;
import inout.Out;
import manager.TodoManager;
import todo.TodoEntry;

public class App {

    public static void main(String[] args) throws Exception {
        TodoManager manager = new TodoManager();
        In.open("todos.txt");
        if (!In.done()) {
            Out.println("Cannot open file todos.txt");
            return;
        }

        int year = In.readInt();
        while (In.done()) {
            int month = In.readInt();
            int day = In.readInt();
            String descr = In.readString();
            System.out.println(LocalDate.of(year, month, day));
            TodoEntry newEntry = new TodoEntry(manager.getNewId(),
descr, LocalDate.of(year, month, day), Status.OPEN);
            manager.add(newEntry);

            year = In.readInt();

        }

        In.close();

        Out.println();
        Out.println("All Todos:");
        Out.println("===========");
        TodoEntry[] allEntries;
        allEntries = manager.get(null, null);
        for (int i = 0; i < allEntries.length; i++) {
            System.out.println(allEntries[i]);
        }

        Out.println();
        Out.println("Until March 9:");
        Out.println("==============");

        allEntries = manager.get(LocalDate.of(2020, 3, 9), null);
        for (int i = 0; i < allEntries.length; i++) {
            System.out.println(allEntries[i]);
        }

        manager.findById(0).complete();
```

```
        manager.findById(1).complete();
        manager.findById(2).complete();

        Out.println();
        Out.println("Done:");
        Out.println("===========");
        allEntries = manager.get(null, Status.DONE);
        for (int i = 0; i < allEntries.length; i++) {
            System.out.println(allEntries[i]);
        }

        Out.println();
        Out.println("Still open:");
        Out.println("===========");
        allEntries = manager.get(null, Status.OPEN);
        for (int i = 0; i < allEntries.length; i++) {
            System.out.println(allEntries[i]);
        }

        Out.println();
        Out.println("Still open until Until March 9:");
        Out.println("==============================");
        allEntries = manager.get(LocalDate.of(2020, 3, 9),
Status.OPEN);
        for (int i = 0; i < allEntries.length; i++) {
            System.out.println(allEntries[i]);
        }
    }
}
```

```
package app;

public enum Status {
    OPEN, DONE,
}
```

```
package manager;

public class TodoListEntry {
    private TodoListEntry next;
    private TodoListEntry previous;
    private Object data;

    public TodoListEntry getNext() {
        return next;
    }

    public void setNext(TodoListEntry next) {
        this.next = next;
    }

    public TodoListEntry getPrevious() {
        return previous;
    }
```

```
    public void setPrevious(TodoListEntry previous) {
        this.previous = previous;
    }

    public Object getData() {
        return data;
    }

    public void setData(Object data) {
        this.data = data;
    }

    public TodoListEntry(TodoListEntry next, TodoListEntry previous,
Object data) {
        this.next = next;
        this.previous = previous;
        this.data = data;
    }

}

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;
        }

        while (entry != null) {

            TodoEntry todoData = (TodoEntry) entry.getData();
            // element is the very first in the list based on due
date
            if (te.getDueTo().isBefore(todoData.getDueTo()) ||
te.getDueTo().isEqual(todoData.getDueTo())) {
                // first set the prev and next of the new element
                newEntry.setPrevious(entry.getPrevious());
                newEntry.setNext(entry);
                // then set the next of the previous if there is a
previous
                if (entry.getPrevious() != null) {
                    entry.getPrevious().setNext(newEntry);
                }
                // then set the previous of the current
                entry.setPrevious(newEntry);

                // if it is before the formerly first element we
have to reset the first
                if (this.first.equals(entry)) {
                    this.first = newEntry;
                }

                return newEntry; // end loop and return newEntry
```

```
                // case that it is the last element in the list
            } else if (entry.getNext() == null) {
                newEntry.setPrevious(entry);
                entry.setNext(newEntry);
                return newEntry;
            }
            entry = entry.getNext();
        }
        // fallback, if we do not hit a return clause in the loop,
which is impossible..
        // but java needs it
        return newEntry;
    }

    /**
     * Returns number of todo entries until a given date, with a
given status. NULL
     * values means everything is fetched for this criteria
     *
     * @param until
     * @param status
     * @return
     */
    int count(LocalDate until, Status status) {
        int count = 0;
        TodoListEntry entry = this.first;
        while (entry != null) {
            TodoEntry todoData = (TodoEntry) entry.getData();
            // first check for date constraint
            if (until == null || todoData.getDueTo().isBefore(until)
|| todoData.getDueTo().isEqual(until)) {
                if (status == null || todoData.getStatus() ==
status) {
                    count++;
                }
            }
            entry = entry.getNext();
        }
        return count;
    }

    /**
     * Returns all TodoEntrys until a given date, with a given
status. NULL values
     * means everything is fetched for this criteria
     *
     * @param until
     * @param status
     * @return
     */
    public TodoEntry[] get(LocalDate until, Status status) {
        TodoEntry[] foundEntries = new TodoEntry[this.count(until,
status)]; // create array of necessary size
        TodoListEntry entry = this.first;
```

```
        int count = 0; // helper variable because of static array
that is intialized beforehand ^^
        while (entry != null) {
            TodoEntry todoData = (TodoEntry) entry.getData();
            // first check for date constraint
            if (until == null || todoData.getDueTo().isBefore(until)
|| todoData.getDueTo().isEqual(until)) {
                if (status == null || todoData.getStatus() ==
status) {
                    foundEntries[count] = todoData;
                    count++;
                }
            }
            entry = entry.getNext();

        }

        return foundEntries;
    }

    /**
     * Returns the last entry in the list
     *
     * @return
     */
    public TodoListEntry getLast() {
        TodoListEntry entry = this.first;
        while (entry.getNext() != null) {
            entry = entry.getNext();
        }
        return entry;
    }
}
```

```
package todo;

import java.time.LocalDate;
import app.Status;

public class TodoEntry {
    private final int id;
    private final String description; // see comment underneath
    private final LocalDate dueTo; // maybe dueTo FINAL is not
ideal, as in the future, one should edit this
    private Status status;

    public TodoEntry(final int id, final String description, final
LocalDate dueTo, final Status status) {
        this.id = id;
        this.description = description;
        this.dueTo = dueTo;
        this.status = status;
    }
```

```
    /**
     * Completes a todo entry by setting the status to DONE
     */
    public void complete() {
        this.status = Status.DONE;
    }

    public LocalDate getDueTo() {
        return dueTo;
    }

    public Status getStatus() {
        return status;
    }

    public int getId() {
        return id;
    }

    @Override
    public String toString() {
        return "ID: " + id + "\t Date: " + dueTo + "\t Description:
" + description + "\t\t\t Status: " + status;
    }

}
```