ToDo List Program

Informal specification:

A ToDo list program is a tool for managing tasks that need to be completed. It allows a user to create a list of tasks and mark them as completed as they are finished. In this implementation of a ToDo list, each user has a set of tasks that they need to complete. Each task has a unique ID and a description. The system ensures that each user has exactly three tasks assigned to them. The system also enforces that each task has a unique ID and that no two tasks have the same ID. Users can mark their tasks as completed as they finish them, allowing them to track their progress and keep track of what they have left to do.

Formal specification:

1. **Alloy Model:**

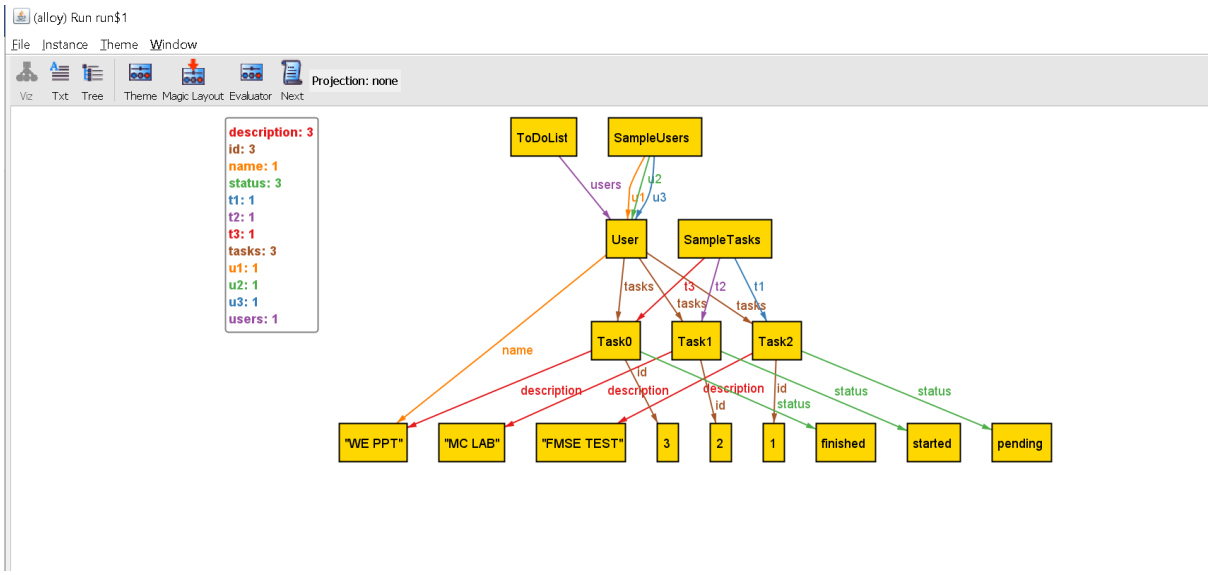
```
-- Defining constant values for status of a task
enum Status{
        pending, started, finished
}
-- Defining Task
sig Task{
        id: one Int,
        description: one String,
        status: one Status
}
-- Defining User as a set of tasks
sig User{
        name: one String,
        tasks: set Task
}

-- Defining todo list with a set of users each having exactly 3 tasks
one sig ToDoList {
   users: set User
} {
   all u: users | #u.tasks = 3
}

-- Defininig some users
one sig SampleTasks {
   t1, t2, t3: Task
}

-- Assigning values to the sample tasks
run {
```

```
    SampleTasks.t1.id = 1
    SampleTasks.t1.description = "FMSE TEST"
    SampleTasks.t1.status = pending
    SampleTasks.t2.id = 2
    SampleTasks.t2.description = "MC LAB"
    SampleTasks.t2.status = started
    SampleTasks.t3.id= 3
    SampleTasks.t3.description = "WE PPT"
    SampleTasks.t3.status = finished
}

-- Defininig some users
one sig SampleUsers {
    u1, u2, u3: User
}

-- Assigning tasks to the sample users
run {
    SampleUsers.u1.tasks = SampleTasks.t1 + SampleTasks.t2
    SampleUsers.u2.tasks = SampleTasks.t2 + SampleTasks.t3
    SampleUsers.u3.tasks = SampleTasks.t1 + SampleTasks.t3
}

-- Defining some constraints on the to-do list
assert ToDoListConstraints {
    -- Each task must have a unique task ID
    all t1, t2: Task | t1 != t2 implies t1.id!= t2.id

    -- No two task should have same id within user's tasks
    all u: User | no t1, t2: u.tasks | t1.id = t2.id
}

--checking assersion for any counter examples
check  ToDoListConstraints

-- Runing the model
run {} for 3
```

Some output screenshots:

```
-- Assigning tasks to the sample users
run {
   SampleUsers.u1.tasks = SampleTasks.t1 + SampleTasks.t2
   SampleUsers.u2.tasks = SampleTasks.t2 + SampleTasks.t3
   SampleUsers.u3.tasks = SampleTasks.t1 + SampleTasks.t3
}

-- Defining some constraints on the to-do list
assert ToDoListConstraints {
   -- Each task must have a unique task ID
   all t1, t2: Task | t1 != t2 implies t1.id!= t2.id

   -- No two task should have same id within user's tasks
   all u: User | no t1, t2: u.tasks | t1.id = t2.id
}

--checking assersion for any counter examples
check  ToDoListConstraints

-- Runing the model
run {} for 3
```

```
Executing "Run run$1"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   875 vars. 102 primary vars. 1482 clauses. 23ms.
   Instance found. Predicate is consistent. 34ms.

Executing "Run run$1"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   949 vars. 111 primary vars. 1601 clauses. 47ms.
   Instance found. Predicate is consistent. 16ms.

Executing "Run run$1"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   949 vars. 111 primary vars. 1601 clauses. 32ms.
   Instance found. Predicate is consistent. 23ms.
```



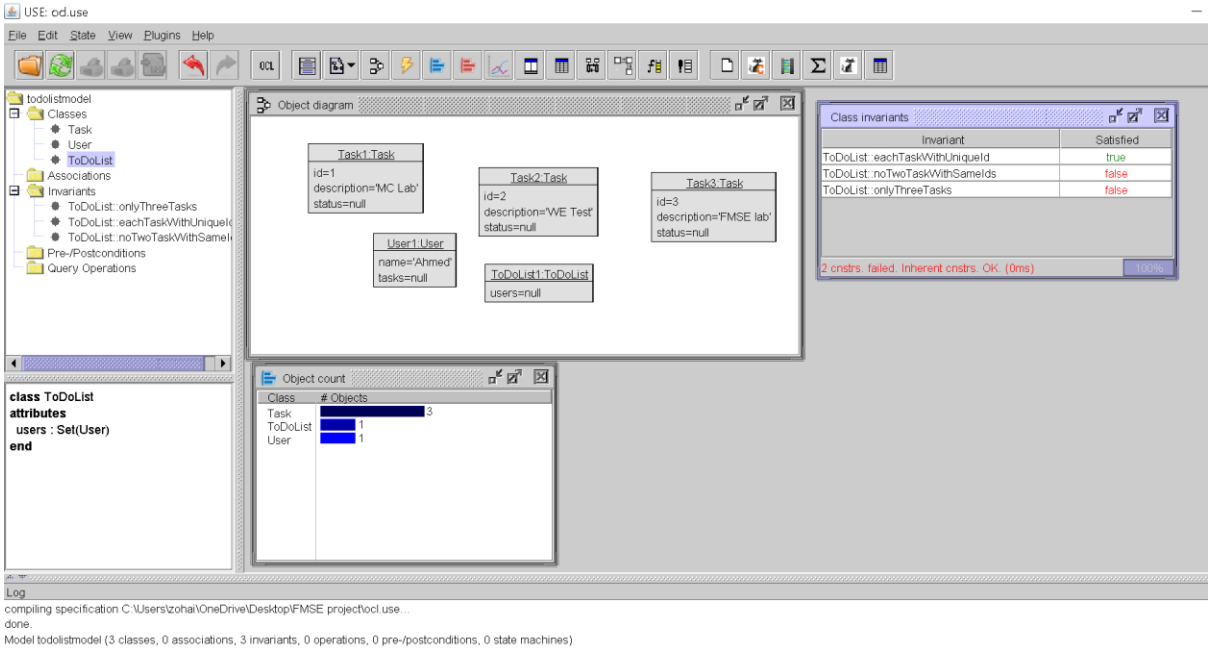
2. OCL Model:

```
model todolistmodel

-- Defining constant values for status of a task
-- Status can be either "pending", "started", or "finished"
enum Status{
  pending,
  started,
  finished
};

-- Defining Task class with three properties:
```

```
-- id: an integer representing the unique identifier for a task
-- description: a string describing the task
-- status: constant value from Status enum to represent the current
status of the task
class Task
  attributes
   id: Integer
   description: String
   status: Status
end

-- Defining User class with two properties:
-- name: a string representing the user's name
-- tasks: a collection of Tasks representing the tasks assigned to the user
class User
  attributes
   name: String
   tasks: Set(Task)
end


-- Defining ToDoList class with one property:
-- users: a collection of User instances representing the users in the to-do
list
class ToDoList
  attributes
   users: Set(User)
end

-- Defining some constraints on the to-do list
-- Each user must have exactly three tasks assigned to them
-- Each task must have a unique task ID
-- No two task should have the same id within a user's tasks
constraints

context ToDoList
  inv onlyThreeTasks: self.users->forAll(u | u.tasks->size() = 3)
```

inv eachTaskWithUniqueId: Task.allInstances()->forAll(t1, t2 | t1 <> t2 implies t1.id <> t2.id)
inv noTwoTaskWithSameIds: User.allInstances()->forAll(u | u.tasks->forAll(t1, t2 | t1 <> t2 implies t1.id <> t2.id))

Screenshot:



3. JML Specification:

```
//Todo list program JML Specifiation
import java.util.*;

enum Status {
   pending, started, finished
}
class Task {
   private int id;
   private String description;
   private Status status;

   /* @requires s != null;
  @ requires i >= 0;
  @ requires d!= null;
  @ ensures status == s;
  @ ensures id == i;
  @ ensures description == d;
  @*/
  public Task(int i, String d, Status s) {
```

```
        id = i;
        description = d;
        status = s;
    }
    public int getId() {
        return id;
    }

    public String getDescription() {
        return description;
    }

    public Status getStatus() {
        return status;
    }
}

class User {
    private String name;
    private HashSet<Task> tasks;

//@ public invariant tasks.size() == 3;

//@ requires n != null;
//@ ensures name == n;
//@ ensures tasks.size() == 3;
    public User(String n, Task t1, Task t2, Task t3) {
        name = n;
        tasks = new HashSet<Task>();
        tasks.add(t1);
        tasks.add(t2);
        tasks.add(t3);
    }
//@ requires s == Status.pending || s == Status.started || s == Status.finished;
//@ requires t != null;
//@ ensures tasks.contains(t);
//@ ensures tasks.size() == \old(tasks.size()) + 1;
public void addTask(Task t, Status s) {
        t.getStatus() = s;
        tasks.add(t);
    }
public String getName() {
        return name;
    }
```

```
    public HashSet<Task> getTasks() {
        return tasks;
    }
}


public class ToDoList {
    private HashSet<User> users;

/*
-- Defining some constraints
-- Each user must have exactly three tasks assigned to them
-- Each task must have a unique task ID
-- No two task should have the same id within a user's tasks
*/
//@ public invariant(\forall Task t1,t2; t1 != t2 && t1.getId() != t2.getId);
//@ public invariant (\forall User u; u in users; u.getTasks().size() == 3);
//@ public invariant (\forall Task t1,t2; t1 in u1.getTasks() ; t2  in
u2.getTasks());t1.getId() != t2.getId());
    public ToDoList() {
        users = new HashSet<User>();
    }

    public HashSet<User> getUsers() {
        return users;
    }

    /*@ requires u != null;
    @ ensures users.contains(u);
    @ ensures users.size() == \old(users.size()) + 1;
        @*/
    public void addUser(User u) {
        users.add(u);
    }


     /*@ requires t != null;
    @ ensures (\exists User u; u in users; t in u.getTasks());
     @*/
    public void addTaskToUser(Task t) {
        for (User u : users) {
            u.addTask(t);
        }
```

```
  }
}
enum Status {
    pending, started, finished
}
class Task {
    private int id;
    private String description;
    private Status status;

    /* @requires s != null;
   @ requires i >= 0;
   @ requires d!= null;
   @ ensures status == s;
   @ ensures id == i;
   @ ensures description == d;
   @*/
   public Task(int i, String d, Status s) {
        id = i;
        description = d;
        status = s;
    }
    public int getId() {
        return id;
    }

    public String getDescription() {
```

```
//@ public invariant tasks.size() == 3;

//@ requires n != null;
//@ ensures name == n;
//@ ensures tasks.size() == 3;
    public User(String n, Task t1, Task t2, Task t3) {
        name = n;
        tasks = new HashSet<Task>();
        tasks.add(t1);
        tasks.add(t2);
        tasks.add(t3);
    }
//@ requires s == Status.pending || s == Status.started || s == Status.finished;
//@ requires t != null;
//@ ensures tasks.contains(t);
//@ ensures tasks.size() == \old(tasks.size()) + 1;
public void addTask(Task t, Status s) {
        t.getStatus() = s;
        tasks.add(t);
    }
public String getName() {
        return name;
    }

    public HashSet<Task> getTasks() {
        return tasks;
    }
```