Introduction: 1

Section A: 1

**1.1 Data Type Objects:** 1

**1.2 Operations:** 1

Section B: 4

**2.1 Data Type Objects:** 4

**2.2 Operations:** 4

**2.2.1 DENTAL\_OFFICE** 4

**2.2.2 PATIENT** 5

**2.2.3 TOOTH** 5

**2.2.4 OPERATION** 5

**2.5 Design Issue** 6

**2.7 ADT Diagram** 6

# **Introduction:**

This is the document for TOOl\_RENTAL ADT and DENTAL\_OFFICE ADT. There will be some BON diagrams for reader to easily understand the logic behind the code, and this document will focus on the design idea for each class involved in both ADTs.

# **Section A:**

1. **TOOL\_RENTAL ADT:**

### **1.1 Data Type Objects:**

inventory : LINKED\_LIST[TOOL\_DESC]

rented : LINKED\_LIST[RENTAL\_DESC]

under\_repair : LINKED\_LIST[REPAIR\_DESC]

to\_rent : LINKED\_LIST[TOOL\_DESC]

### **1.2 Operations:**

>> new\_shop(new\_tools: LINKED\_LIST[TOOL\_DESC]; min\_tool\_count: INTEGER)

-- Set up a tool rental shop, there is a minimum of 100 tools initially available for rent.

-- Require: min\_tool\_count >= 100

-- Ensure: inventory.count = min\_tool\_count and

number\_rented\_tools = 0 and

number\_under\_repair\_tools = 0

>> size\_of\_inventory --> INTEGER

--- The number of tools in the inventory; i.e. owned by the shop.

--- Require: NONE

--- Ensure: Result := inventory.count

>> number\_rented\_tools --> INTEGER

--- The number of tools that are rented out

--- Require: NONE

--- Ensure: Result := rented.count

>> number\_under\_repair\_tools --> INTEGER

--- The number of tools under repair.

--- Require: NONE

--- Ensure: Result := under\_repair.count

>> dispose\_of(tool: TOOL\_DESC)

--- Remove a tool from the shop. It is no longer rentable or repairable.

--- It is no longer owned by the shop.

--- Require: valid\_tool: tool /= void

exist\_inventory: in\_inventory(tool)

not\_exist\_rented: not in\_rented(tool)

--- Ensure: not\_exist\_inventory: not in\_inventory(tool)

decreased\_inventory\_size: size\_of\_inventory = (old size\_of\_inventory - 1)

>> buy (new\_tool : TOOL\_DESC)

--- Buy a new tool, which is, upon purchase, immediately available for rent.

--- Require: valid\_tool: new\_tool /= void

not\_exist\_inventory: not in\_inventory(new\_tool)

brand\_new: new\_tool.repair\_count = 0

--- Ensure

exist\_in\_inventory: in\_inventory(new\_tool)

increased\_inventory\_size: size\_of\_inventory = (old size\_of\_inventory + 1)

increased\_to\_rent: to\_rent.count = (old to\_rent.count + 1)

>> start\_repair\_process (tool : TOOL\_DESC)

--- The tool cannot be rented until it is repaired.

--- require

valid\_tool: tool /= void

exist\_in\_inventory: in\_inventory(tool)

exist\_in\_to\_rent: in\_to\_rent(tool)

not\_exist\_in\_rented: not in\_rented(tool)

not\_exist\_in\_repair: not in\_repair(tool)

--- ensure

exist\_in\_inventory: in\_inventory(tool)

not\_exist\_in\_rented: not in\_rented(tool)

exist\_in\_repair: in\_repair(tool)

not\_exist\_to\_rent: not in\_to\_rent(tool)

increased\_under\_repair\_size: number\_under\_repair\_tools = (old number\_under\_repair\_tools + 1)

decreased\_number\_rented: to\_rent.count = (old to\_rent.count - 1)

>> to make this section more meaningful, please refer to listing of code, there are more supported

>> routines and contracts

* 1. **Operation Interactions:**

There are many interactions when writing the contracts, mostly between the contract support routines and actual operations in TOOL\_RENTAL.

For example, when writing:

>> rent(tool: TOOL\_DESC)

--- the preconditions are basically checking the proper location in which the tool should be before the

--- rental occurs. There is a supported routines called in\_rented to make sure this tool to be rented is

--- not in the existing rented list, since the whole system follow the mutual exclusive mechanism,

--- which means, for all tool or rental or repair tools, it is not possible that the tool will be in two or

--- more lists at the same time.

--- Meanwhile, in\_rented uses the quantifier that I wrote in order to traverse the corresponded list.

--- And the predicate we use same\_tool is doing the existence checking.

* 1. **Axioms / ADT Invariant:**

total\_inventory\_equation:

to\_rent.count = (size\_of\_inventory - number\_rented\_tools - number\_under\_repair\_tools)

rented list intersect repair list = void ^ rented list intersect to rent = void

repair list intersect rented list = void ^ repair list intersect to rent = void

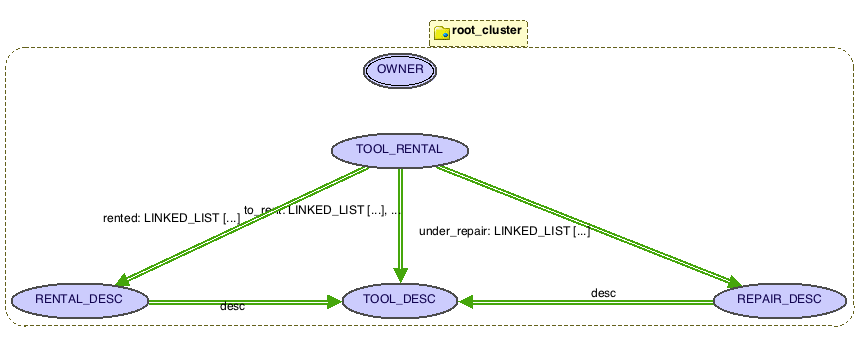
to rent list intersect repair list = void ^ to rent list intersect rented list = void

all list elements are in the inventory

* 1. **Design Issue:**

Since the whole system follow the mutual exclusive mechanism, which means, for all tool or rental or repair tools, it is not possible that the tool will be in two or more lists at the same time. Hence, the contract design idea is basically focusing on the list elements checking to follow the rules.

* 1. **ADT Diagram for TOOL\_RENTAL**

****

# **Section B:**

1. **DENTAL\_OFFICE ADT:**

### **2.1 Data Type Objects:**

Imported: NONE

Exported: DENTAL\_OFFICE

Hidden: PATIENT, TOOTH, OPERATION

### **2.2 Operations:**

#### **2.2.1 DENTAL\_OFFICE**

>> make

--- initialize the dental office which contains a list of patients

>> get\_root\_canal\_patients\_list --> LINKED\_LIST[PATIENT]

--- Query the existing patient list to get the patients who have had root canal

--- Require: patient\_list /= void

--- Ensure: for all patients in the Result list, they all have had root canal operation

>> number\_of\_teeth\_extraction(start\_Date, end\_Date: DATE): INTEGER

--- return the number of extractions done in the period of time

--- Require: start\_Date is less than end\_Date

--- Ensure: Result := #{t: TOOTH | t in Patients\_LIST :: t.operations.has(extraction) ^ t.operation.date is between start\_Date and end\_Date }

>> get\_operations\_from\_all(start\_Date, end\_Date: DATE): LINKED\_LIST[OPERATION]

--- all operations performed on male/female patients

--- Require: start\_Date is less than end\_Date

--- Ensure: Result := list of operations that perform on any of tooth of each patients in ---- the dental\_office

>> some support routines and agent to traverse the patient list, please refer to source >> code for detail

#### **2.2.2 PATIENT**

>> make(address: STRING; bod: DATE)

--- Initialize the patient with address, birth of date and the list of his/her teeth

>> had\_root\_canal : BOOLEAN

--- routine to determine if the patient has had root canal

--- require teeth /= void

--- ensure Result = there\_exists\_teeth(agent has\_root\_canal\_tooth\_in\_p(?))

>> had\_extraction: BOOLEAN

--- require teeth /= void

--- ensure Result = there\_exists\_teeth(agent has\_extraction\_tooth\_in\_p(?))

#### **2.2.3 TOOTH**

>> has\_root\_canal\_tooth: BOOLEAN

--- Return True if this tooth has had root canal

--- Require: True

--- Ensure: Result := there\_exists1\_operation(agent is\_root\_canal(?))

>> has\_extraction\_tooth: BOOLEAN

--- Return True if this tooth has been extracted

--- Require: True

--- Ensure: Result := there\_exists1\_operation(agent is\_extraction(?))

>> get\_extraction\_op: OPERATION

--- Return the operation object for query in dental\_office to get the date of that

--- operation

--- Require: True

--- Ensure: Result := find\_operation(agent is\_extraction(?))

>> Some other support routines please refer to the source code in the end of report

#### **2.2.4 OPERATION**

>> make(date: DATE; type: INTEGER)

--- Initialize the operation object that has date and operation type

--- for operation type, 0 is filling, 2 is extraction, and 1 for root canal

* 1. **Operation Interactions:**

In this DENTAL\_OFFICE implementation, the operation interactions among classes are basically the support routines interaction, that is, traversing every element among patient list or tooth list with a patient class.

* 1. **Axioms / ADT Invariant**

-- If this tooth got extracted then current is set to void, this idea

-- will be applied to patient class to check if any one of the tooth

-- is extracted, if yes, then the tooth is set to void and no more

-- operations can be done

extracted: there\_exists1\_operation(agent is\_extraction(?)) implies Current = void

-- make sure that the count for root canal operation is at most one

at\_most\_1\_root\_canal: count\_exists\_operation(agent is\_root\_canal(?)) <= 1

### **2.5 Design Issue**

Design the DENTAL\_OFFFICE require a good understanding of classes relationships. Since this system is mainly dealing with the problem of list traversal and query the list elements, hence, in the design process, we need some support routines to accomplish these tasks. In Eiffel, this can be done with the idea of agent and tuple features.

* 1. **Sample Query Usage**

In DENTAL\_OFFICE, for example, we want to instantiate it within the START class.

E.g:

office: DENTAL\_OFFICE

office. get\_root\_canal\_patients\_list

--- this will return a list of patients who have had root\_canal

### **2.7 ADT Diagram**

