EECS 3311 Project Report: Tracker

Amanda D’Errico (213007554, aderrico)

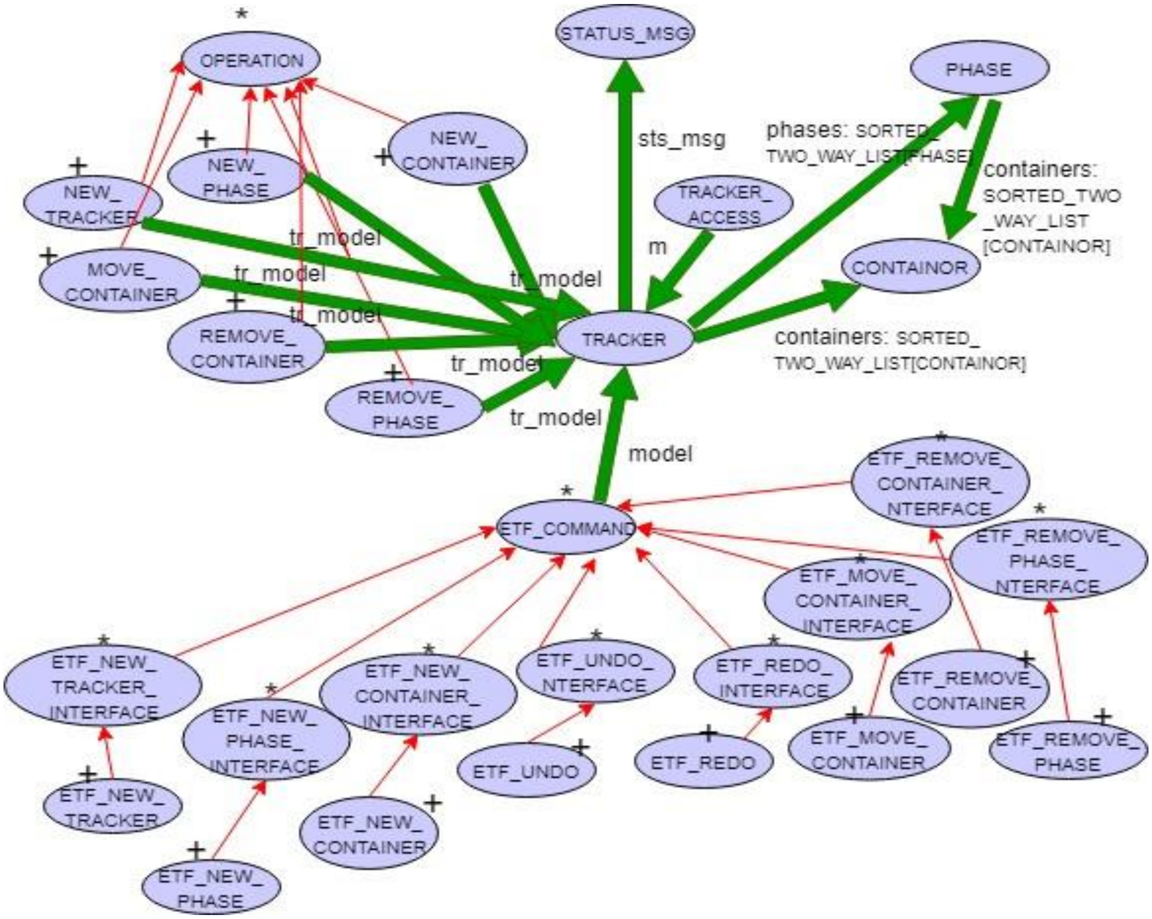
Table of Contents

Requirements for Project Tracker

The tracker project's main objectives are to effectively handle any input given for the main tracker class to accumulate phases and containers, and to ensure that their collective radiations don't exceed the maximum. The customer requirements are provided in the Tracker project document: Containers of material go through various stages of processing in phases before they can handle radioactive materials. There is an initial unpacking phase, an assay phase which measures recoverable material and a compacting phase. Each container has a unique identifier and contains one type of material, and also measures radiation count in MSv. An operator can add or remove phases and add or remove containers. It is also possible to move containers between phases. The objective is to ensure that containers are added, moved or removed to avoid dangerous situations where capacity or radioactivity could exceed.

See tracker-defns.txt for the grammar used in the user interface. The acceptance tests given in the *tests/instructor* directory contain the files *at1.expected.txt*, *at2.expected.txt* and *at3.expected.txt*, which provide a clearer understanding of the behavior for the tracker project.

BON class diagram Overview (Architecture)



The program is being manipulated through the ETF Command and ETF Redo/Undo classes which all call the model class (Tracker). Tracker has a singleton instance so the class is only created once for the operator to record to container and phases in the tracker plant. Tracker is the class used as the supplier to all the classes that use it as an instance. In the bon diagram above, the Operation class doesn’t use a Tracker instance, but its effective classes do. Effective classes New_Tracker, New_Phase, New_Container, Remove_Container, Remove_Phase, and Move_Container use a tracker model instance to manipulate the state of the tracker by updating the messages stored in strings. When the Tracker class runs, the new string messages are updated in the out function. Operation is used for the undo and redo implementation in Tracker to preserve polymorphism. Additional classes include Tracker_Plant, Phase, Containor and Status_Msg classes which are explained in the following page.

Table of Modules – Responsibilities and Information Hiding

Class	Description	Design
Containor	**Responsibility:** The Containor for Tracker project. It defines the container id, the single material, the phase id of the phase it's located in and the radioactivity. **Secret:** Only modules Containor has are essentially its attributes so that information hiding occurs and other classes are unable to manipulate its functionality.	All attributes must be defined to represent a container, and encapsulation and information hiding were used for implementation.
Phase	**Responsibility:** The Phase for Tracker project. It defines the container id, the single material, the phase id of the phase it's located in and the radioactivity. **Secret:** Container attribute of Phase does not affect the containers list in the Tracker class, therefore	All attributes must be defined to represent a phase, and encapsulation and information hiding were used for implementation with the container attribute.

	fulfillng the single responsibility principle.	
Tracker_Plant	**Responsibility:** Tracker sets the max_phase_radiation and max_container_radiation to be used in New_Tracker. The comparable function is_less is also redefined for Tracker since I am comparing containers in a sorted two way list in the Tracker class. THis class is used primarily for introducing the string out for a Tracker_Plant class. **Secret:** Only responsible for ensuring max radiations are not exceeded (SRP).	This class has only the responsibility of defining the max phase and max container radiation.
Status_Msg	**Responsibility:** This class stores all game status messages. Some examples include default message “ok”, “e1: current tracker is in use”, “e5: identifiers/names must start with A-Z, a-z or 0..9”, etc. **Secret:** None	Initializes various status messages. If messages are not default they are checked in the ETF classes for operations and redo undo.

Operation	**Responsibility:** The deferred class for all the actions in the tracker project. This includes New_Tracker, New_Phase, New_Container, Remove_Container, Remove_Phase, and Move_Container. All classes have execute, undo and redo as deferred functions. Additional deferred classes that are implemented in effective classes are set_new_track, set_old_state and set_old_msg. The attributes old_msg and old_state are used to retrieve the message of the action for undo and redo function calls. New_track is used for ETF_Undo to ensure that New_Tracker is called so the appropriate messages for undo can be outputted. All other effective classes set boolean new_track to be false. **Secret:** Uses inheritance to define all operations to	Used inheritance for all child classes to use, so that they share similar functionality and follow an is-a relationship.

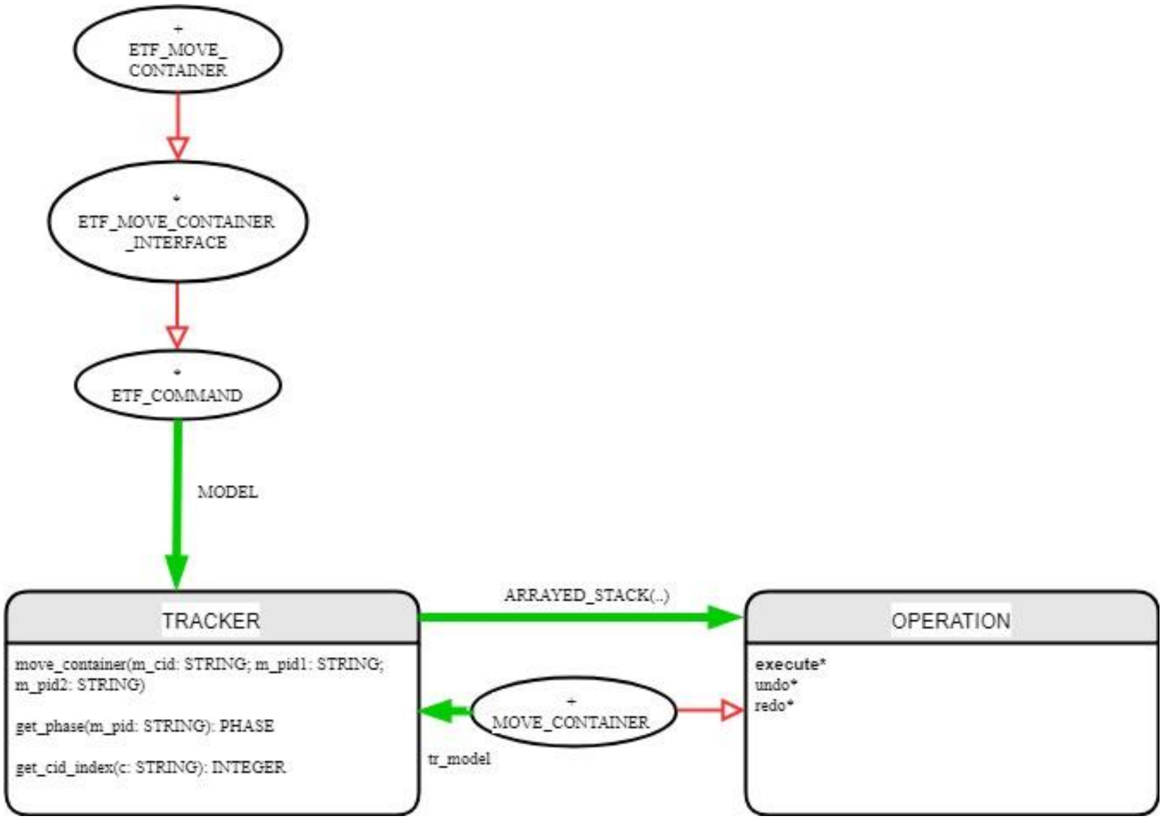
	have specific features.	
New_Tracker	**Responsibility:** This class sets and defines the max phase and max container radiation in the system. Any amount for container or phase classes that exceeds gets rejected in the ETF classes. Also stores the message associated with calling the instance. **Secret:** None	Allowed ETF New_Tracker to incorporate defensive programming and catch all errors and then defined each attribute and converted array of integers back to array of strings to access in New_Tracker.
New_Phase	**Responsibility:** This class stores name of phase, the capacity of the amount of containers it can hold and the materials that the phase expects. The materials is of type ARRAY[STRING] so the ETF class can convert the expected materials input from an array of integers of the set of materials to an array of strings that represent each material. Also stores the message associated with calling the instance.	Allowed ETF New_Phase to incorporate defensive programming and catch all errors and then defined each attribute and converted array of integers back to array of strings to access in New_Phase.

	Secret: None	
New_Container	**Responsibility:** This class identifies a new container to put into a phase. Its attributes include its container id, the expected material and its radioactivity and the phase id to determine which phase the container is being placed. In the ETF class, there is a function that converts the index of the material given into the actual material to call in this class. Also stores the message associated with calling the instance. **Secret:** None	Allowed ETF New_Container to incorporate defensive programming and catch all errors and then defined each attribute and converted index of material back to the material string to access in New_Container.
Remove_Container	**Responsibility:** This class identifies the container to remove and the phase it’s located in and removes it. If there is no container that exists then when placing the instance in the undo stack arbitrary phases and containers are defined. Also stores the message associated with calling the	Allowed ETF Remove_Container to incorporate defensive programming and catch all errors and then defined each attribute and accessed pids and cids through the Phase and Containor classes.

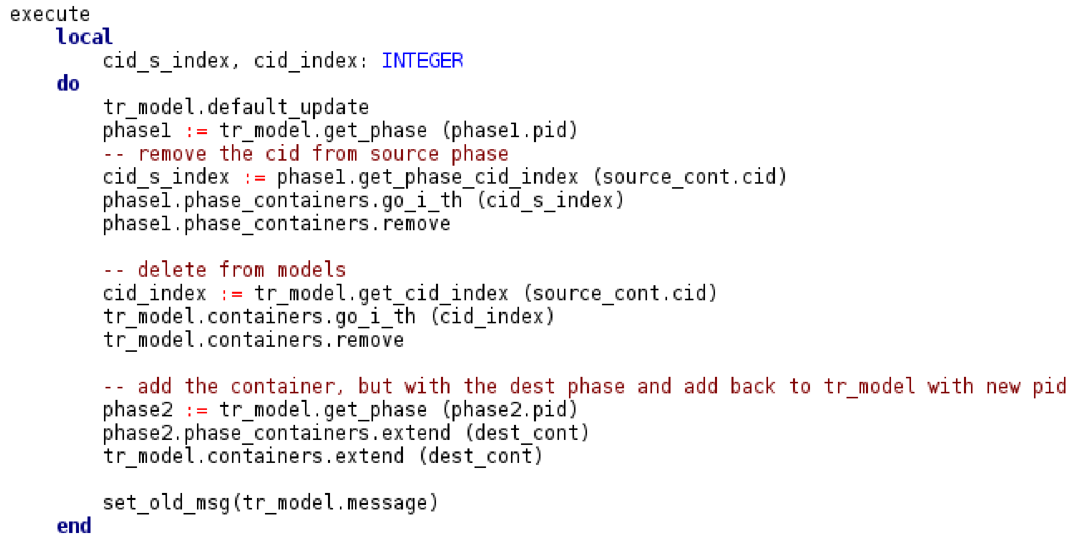
	instance. **Secret:** None	
Remove_Phase	**Responsibility:** This class identifies the phase and removes it. If there are no phases that exist then when placing the instance in the undo stack arbitrary phases are defined. Also stores the message associated with calling the instance. **Secret:** None	Allowed ETF Remove_Phase to incorporate defensive programming and catch all errors and then defined each attribute and accessed pids through the Phase class.
Move_Container	**Responsibility:** This class identifies the container in the source phase to remove and place in the destination phase. If there is no container or phase that exists then when placing the instance in the undo stack arbitrary phases and containers are defined. Also stores the message associated with calling the instance. **Secret:** None	Allowed ETF Remove_Container to incorporate defensive programming and catch all errors and then defined each attribute and accessed pids and cids through the Phase and Containor classes.
Tracker	**Responsibility:** The class that supplies to every class implemented. Phase, Containor, Tracker, Status_Msg, stacks for	All effective Operation classes and undo and redo are called here after calling their functions in their respective ETF classes.

	redo/undo, lists for phases and containors to print in the output function, and all the effective Operation classes are included in Tracker. **Secret:** All effective classes are called here as functions and has postconditions to test correctness.	Calls to Tracker only occur if all the error cases are false. A sorted two way list is also kept for phases and containers to print the output.

Expanded Description of MOVE_CONTAINER

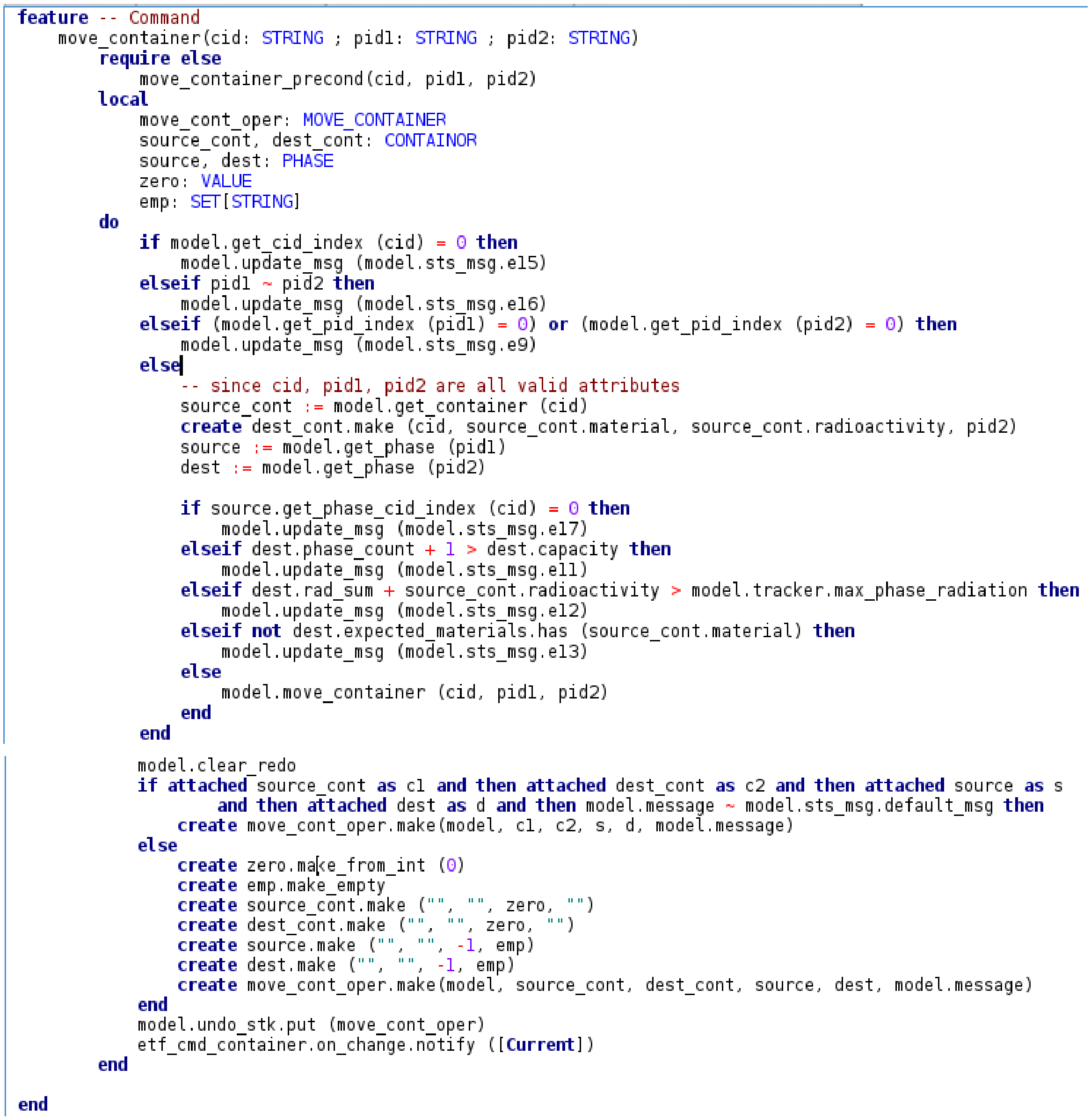


Below is the execute function in MOVE_CONTAINER that helps move containers to their next phase:



```
execute
    local
        cid_s_index, cid_index: INTEGER
    do
        tr_model.default_update
        phase1 := tr_model.get_phase (phase1.pid)
        -- remove the cid from source phase
        cid_s_index := phase1.get_phase_cid_index (source_cont.cid)
        phase1.phase_containers.go_i_th (cid_s_index)
        phase1.phase_containers.remove

        -- delete from models
        cid_index := tr_model.get_cid_index (source_cont.cid)
        tr_model.containers.go_i_th (cid_index)
        tr_model.containers.remove

        -- add the container, but with the dest phase and add back to tr_model with new pid
        phase2 := tr_model.get_phase (phase2.pid)
        phase2.phase_containers.extend (dest_cont)
        tr_model.containers.extend (dest_cont)

        set_old_msg(tr_model.message)
    end
```

A query get_phase in Tracker records the phase in which the container is located in. Get_phase_cid_index in Phase checks for the index in the list of containers recorded. Within the source phase, the container is removed. Get_cid_index in Tracker then removes the container associated with that phase. Deleting the container in the container list in Tracker is necessary, as the pid of the same container has changed. Now the destination phase is extending for the new container and the container gets added back to the model class with a new pid.



```
feature -- Command
    move_container(cid: STRING ; pid1: STRING ; pid2: STRING)
        require else
            move_container_precond(cid, pid1, pid2)
        local
            move_cont_oper: MOVE_CONTAINER
            source_cont, dest_cont: CONTAINOR
            source, dest: PHASE
            zero: VALUE
            emp: SET[STRING]
        do
            if model.get_cid_index (cid) = 0 then
                model.update_msg (model.sts_msg.e15)
            elseif pid1 ~ pid2 then
                model.update_msg (model.sts_msg.e16)
            elseif (model.get_pid_index (pid1) = 0) or (model.get_pid_index (pid2) = 0) then
                model.update_msg (model.sts_msg.e9)
            else
                -- since cid, pid1, pid2 are all valid attributes
                source_cont := model.get_container (cid)
                create dest_cont.make (cid, source_cont.material, source_cont.radioactivity, pid2)
                source := model.get_phase (pid1)
                dest := model.get_phase (pid2)

                if source.get_phase_cid_index (cid) = 0 then
                    model.update_msg (model.sts_msg.e17)
                elseif dest.phase_count + 1 > dest.capacity then
                    model.update_msg (model.sts_msg.e11)
                elseif dest.rad_sum + source_cont.radioactivity > model.tracker.max_phase_radiation then
                    model.update_msg (model.sts_msg.e12)
                elseif not dest.expected_materials.has (source_cont.material) then
                    model.update_msg (model.sts_msg.e13)
                else
                    model.move_container (cid, pid1, pid2)
                end
            end

            model.clear_redo
            if attached source_cont as c1 and then attached dest_cont as c2 and then attached source as s
                    and then attached dest as d and then model.message ~ model.sts_msg.default_msg then
                create move_cont_oper.make(model, c1, c2, s, d, model.message)
            else
                create zero.make_from_int (0)
                create emp.make_empty
                create source_cont.make ("", "", zero, "")
                create dest_cont.make ("", "", zero, "")
                create source.make ("", "", -1, emp)
                create dest.make ("", "", -1, emp)
                create move_cont_oper.make(model, source_cont, dest_cont, source, dest, model.message)
            end
            model.undo_stk.put (move_cont_oper)
            etf_cmd_container.on_change.notify ([Current])
        end

end
```

In the ETF_MOVE_CONTAINER class all the error cases are checked in priority of the oracle before the move is initiated, in order to set error messages instead of moving the container. The error cases are checked through Boolean functions located in the model class (Tracker). This defensive programming logic is also used for the other ETF effective classes. The main design of the project was finding the cid and pid and removing or extending in to the phases and/or containers lists in Tracker.

Significant Contracts (Correctness)

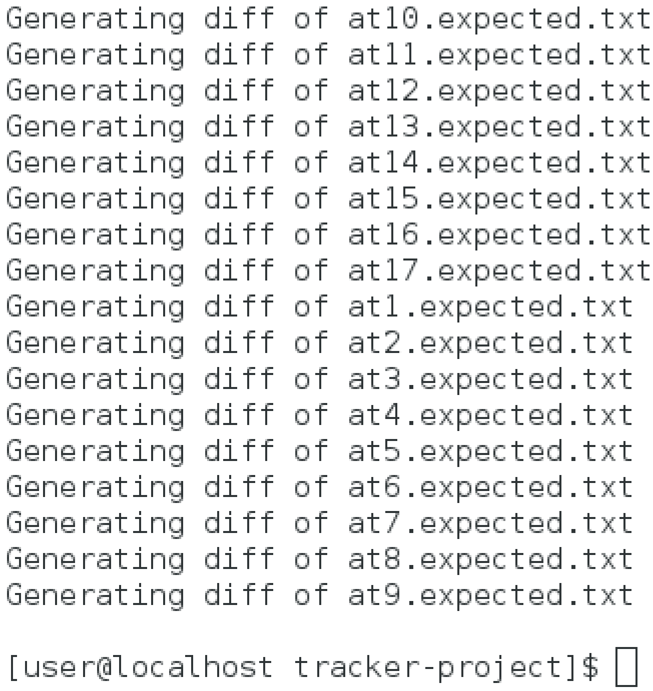
In the Tracker class (contract view can be found under Appendix) many postconditions were checked to ensure correctness. The design of the tracker-project was to use defensive programming with all the effective Operation classes. Once all error conditions in the ETF classes were bypassed and the user was able to perform an operation without issues, the model class (Tracker) was used and the function name identical to the class name was called. In that function post conditions are checked for the phases and containers lists.

The phases and containers lists are used for printing output, so it is crucial that the order and details of each phase and container are accurate. After each execution of the operation functions in Tracker, the size of the phases and/or containers list is checked and if the phase and/or container does or does not exist. These are the strongest postconditions that can be checked.

For the function new_phase, the phases list must increase in size by one and the get_pid_index function must not equal 0, indicating that a phase with the id just passed as a parameter does exist in the phases list. For the function new_container, the containers list must increase in size by one and the get_cid_index function must not equal 0, indicating that a container with the id just passed as a parameter does exist in the containers list. For the function remove_container, the containers list must decrease in size by one and the get_cid_index function must be equal to 0, indicating that a container with the id just passed as a parameter does not exist in the containers list. When iterating in get_cid_index, if the index exceeds the size of the containers list, then the result is 0 to indicate that there is no index that exists in the list. For the function remove_phase, the phases list must decrease in size by one and the get_pid_index function must be equal to 0, indicating that a phase with the id just passed as a parameter does not exist in the phases list. Similarly in get_pid_index, if the index exceeds the size of the phases list, the index value is reset to 0. Lastly, for the function move_container, the containers list and phases list must not change in size and the get_cid_index and get_pid_index functions must not be equal to 0, indicating that the container still exists in the tracker because it was only moved around, and that both the source and destination pids given as parameters exists too because their phases weren’t deleted yet.

Summary of Testing Procedures

Below is a screen shot of the acceptance tests I produced, which have all passed, along with a summary of all my test procedures.



```
Generating diff of at10.expected.txt vs at10.actual.txt
Generating diff of at11.expected.txt vs at11.actual.txt
Generating diff of at12.expected.txt vs at12.actual.txt
Generating diff of at13.expected.txt vs at13.actual.txt
Generating diff of at14.expected.txt vs at14.actual.txt
Generating diff of at15.expected.txt vs at15.actual.txt
Generating diff of at16.expected.txt vs at16.actual.txt
Generating diff of at17.expected.txt vs at17.actual.txt
Generating diff of at1.expected.txt vs at1.actual.txt
Generating diff of at2.expected.txt vs at2.actual.txt
Generating diff of at3.expected.txt vs at3.actual.txt
Generating diff of at4.expected.txt vs at4.actual.txt
Generating diff of at5.expected.txt vs at5.actual.txt
Generating diff of at6.expected.txt vs at6.actual.txt
Generating diff of at7.expected.txt vs at7.actual.txt
Generating diff of at8.expected.txt vs at8.actual.txt
Generating diff of at9.expected.txt vs at9.actual.txt

[user@localhost tracker-project]$
```

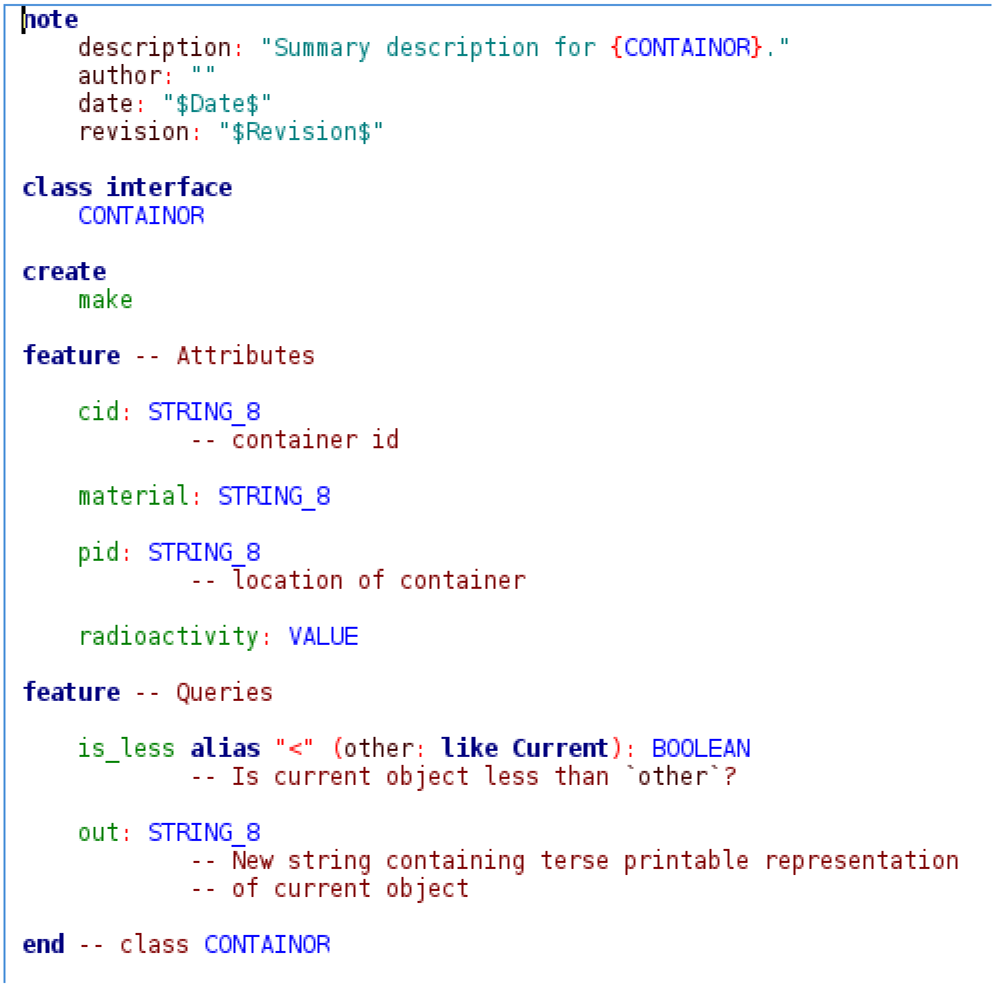
Test File	**Description**	**Passed**
at1.txt	Tests error cases for New_Phase and New_Container. Tests valid id cases and also priority cases, such as having containers not being empty error check before capacity size which should be called before the identifier in New_Phase. Also tested standard error-free output for new_container and the order of the cids (fixed issue by using SORTED_TWO_WAY_LIST).	Yes
at2.txt	Similar to at1.txt. Tests boundary cases with exceeding phase capacity for new_container and error cases for move_container, having the cid existing being the most	Yes

	important test case. Remove_Container is also tested for standard error-free output.	
at3.txt	Testing cases for new_container when phase radiation sum reaches capacity and removing a phase with a container still existing.	Yes
at4.txt	Standard calls to new_tracker, new_phase and new_container until container exceeds phase capacity. Also tests individual container exceeding own capacity, which takes priority over phase capacity exceeding.	Yes
at5.txt	Similar to at1 and at2 for error checking. Adds cid that already exists in tracker and removes a cid that does not exist.	Yes
at6.txt	Tests moving containers from different phases around where capacity is a border test case and capacity is exceeded. Also tested container radioactivity and if it rounds to two decimal places with Phase and Containor output.	Yes
at7.txt	Tests undo and redo on New_Tracker calls on error-free and error cases. Calls new_phase after to ensure max_phase_radiation and max_container_radiation are still defined.	Yes
at8.txt	Tests move_container and new_container undo and redo, with error-free cases and error e17.	Yes
at9.txt	Similar to at8 but also tests undo for new_phase after each call for it.	Yes
at10.txt	Tests Unpacking and Assay phase types. Redo on New_tracker with nothing in redo stack.	Yes
at11.txt	Tests undo and redo on New_Tracker again.	Yes
at12.txt	Calls new_phase before new_tracker and initializes no max_phase_radiation and max_container_radiation. New_tracker is called after and undo commands are done on both new_tracker and new_phase.	Yes
at13.txt	Calls undo on new_tracker error-free test case to test message, when previous command was an error-case for new_tracker.	Yes
at14.txt	Tests undo on new_phase, new_container and calls undo multiple times in a row with both error and error-free cases.	Yes

	Calls a new function when undoing and before redoing those actions.	
at15.txt	Similar to at14 with multiple undo commands followed by redo commands. Tests error cases.	Yes
at16.txt	Tests decimal inputs to new_tracker with large numbers. Also tests radiation sum with new_containers and the output functions associated with the Phase and Containor classes.	Yes
at17.txt	Similar to at14 and at15, making multiple calls to undo and how it affects the model containers and phases lists.	Yes

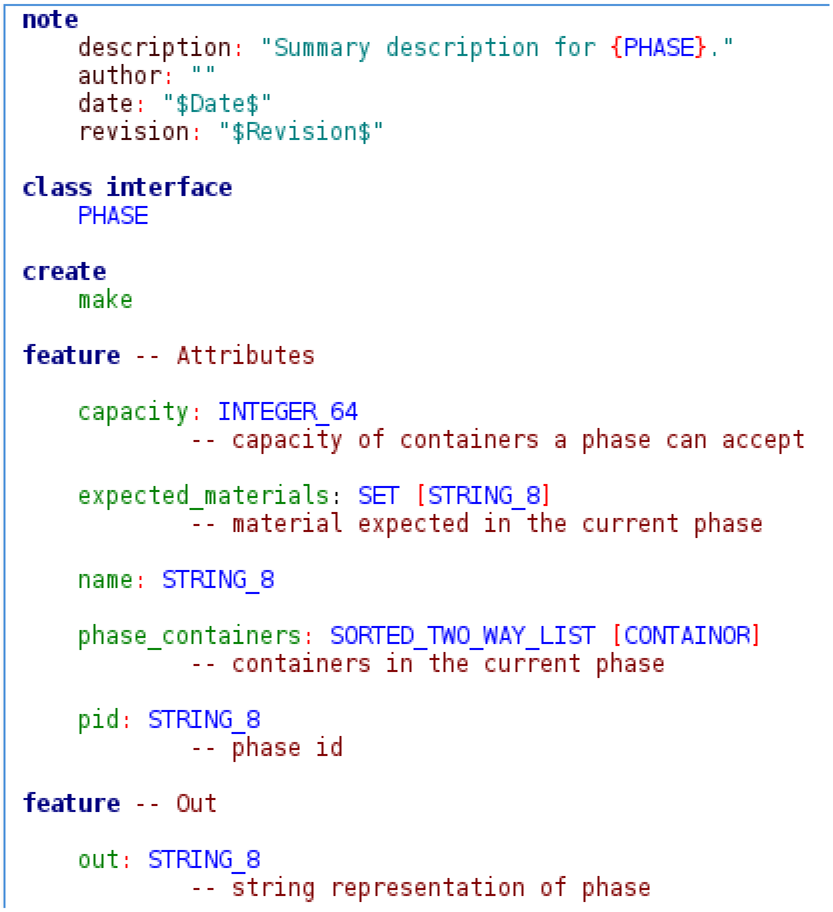
Appendix (Contract Views)

Containor:

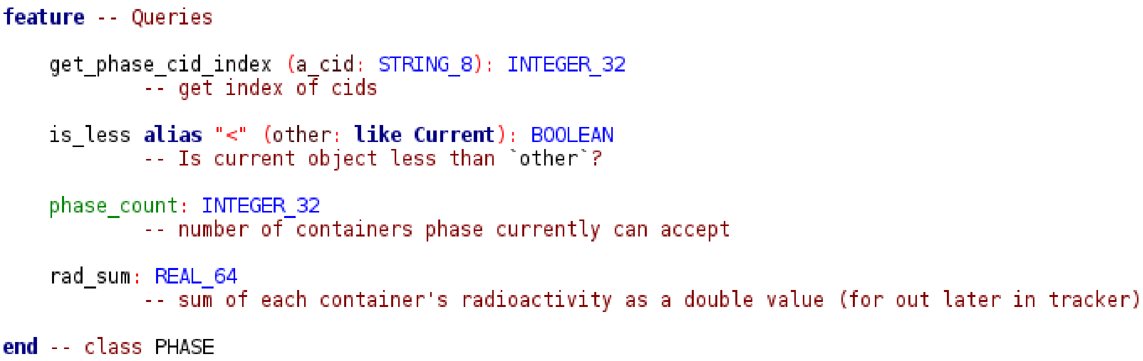


```
note
    description: "Summary description for {CONTAINOR}."
    author: ""
    date: "$Date$"
    revision: "$Revision$"

class interface
    CONTAINOR

create
    make

feature -- Attributes

    cid: STRING_8
            -- container id

    material: STRING_8

    pid: STRING_8
            -- location of container

    radioactivity: VALUE

feature -- Queries

    is_less alias "<" (other: like Current): BOOLEAN
            -- Is current object less than `other`?

    out: STRING_8
            -- New string containing terse printable representation
            -- of current object

end -- class CONTAINOR
```

Phase:

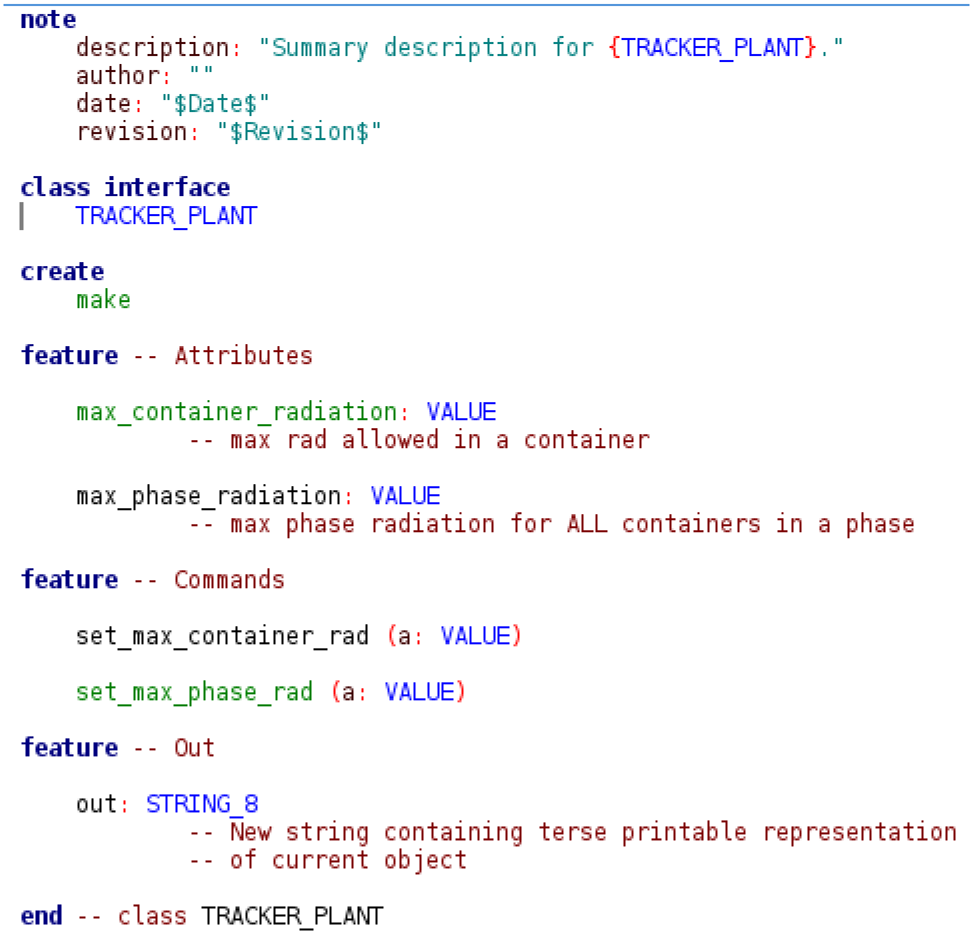


```
note
    description: "Summary description for {PHASE}."
    author: ""
    date: "$Date$"
    revision: "$Revision$"

class interface
    PHASE

create
    make

feature -- Attributes

    capacity: INTEGER_64
            -- capacity of containers a phase can accept

    expected_materials: SET [STRING_8]
            -- material expected in the current phase

    name: STRING_8

    phase_containers: SORTED_TWO_WAY_LIST [CONTAINOR]
            -- containers in the current phase

    pid: STRING_8
            -- phase id

feature -- Out

    out: STRING_8
            -- string representation of phase
```



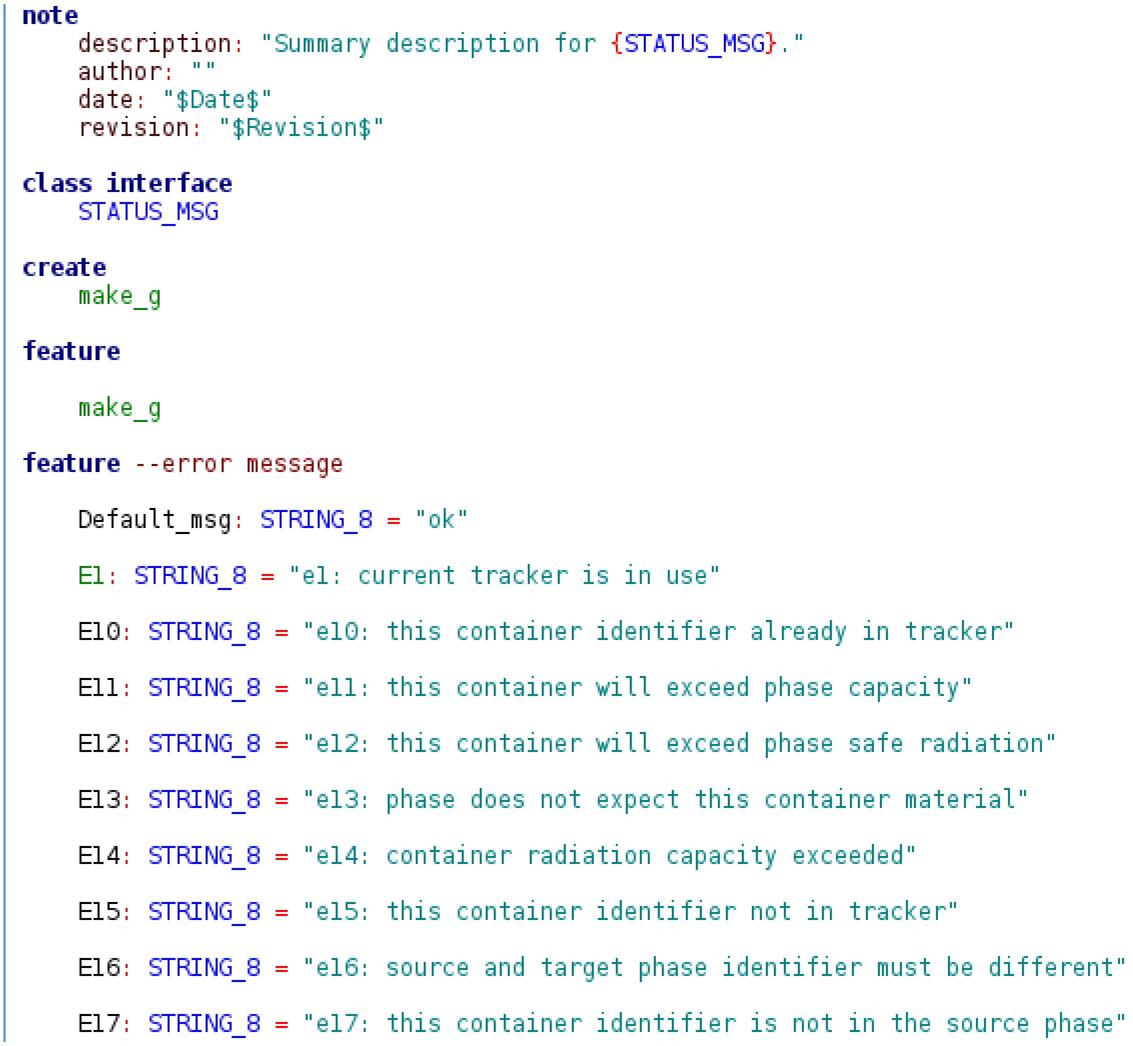
```
feature -- Queries

    get_phase_cid_index (a_cid: STRING_8): INTEGER_32
            -- get index of cids

    is_less alias "<" (other: like Current): BOOLEAN
            -- Is current object less than `other`?

    phase_count: INTEGER_32
            -- number of containers phase currently can accept

    rad_sum: REAL_64
            -- sum of each container's radioactivity as a double value (for out later in tracker)

end -- class PHASE
```

Tracker_Plant:

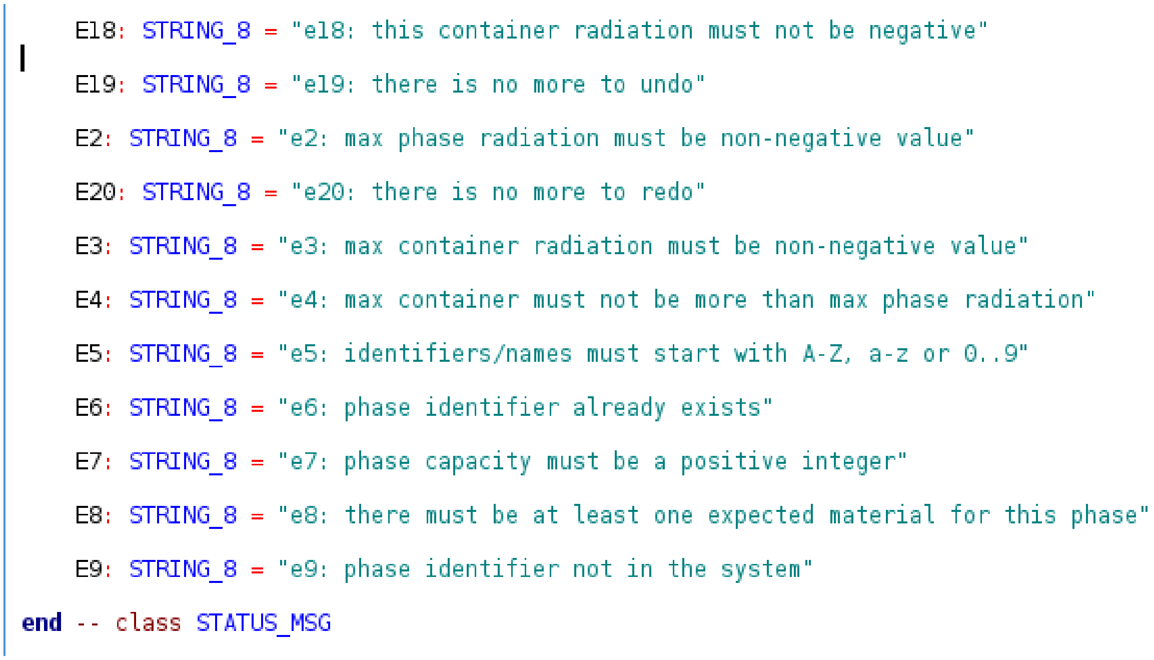


```
note
    description: "Summary description for {TRACKER_PLANT}."
    author: ""
    date: "$Date$"
    revision: "$Revision$"

class interface
    TRACKER_PLANT

create
    make

feature -- Attributes

    max_container_radiation: VALUE
            -- max rad allowed in a container

    max_phase_radiation: VALUE
            -- max phase radiation for ALL containers in a phase

feature -- Commands

    set_max_container_rad (a: VALUE)

    set_max_phase_rad (a: VALUE)

feature -- Out

    out: STRING_8
            -- New string containing terse printable representation
            -- of current object

end -- class TRACKER_PLANT
```

Status_Msg:

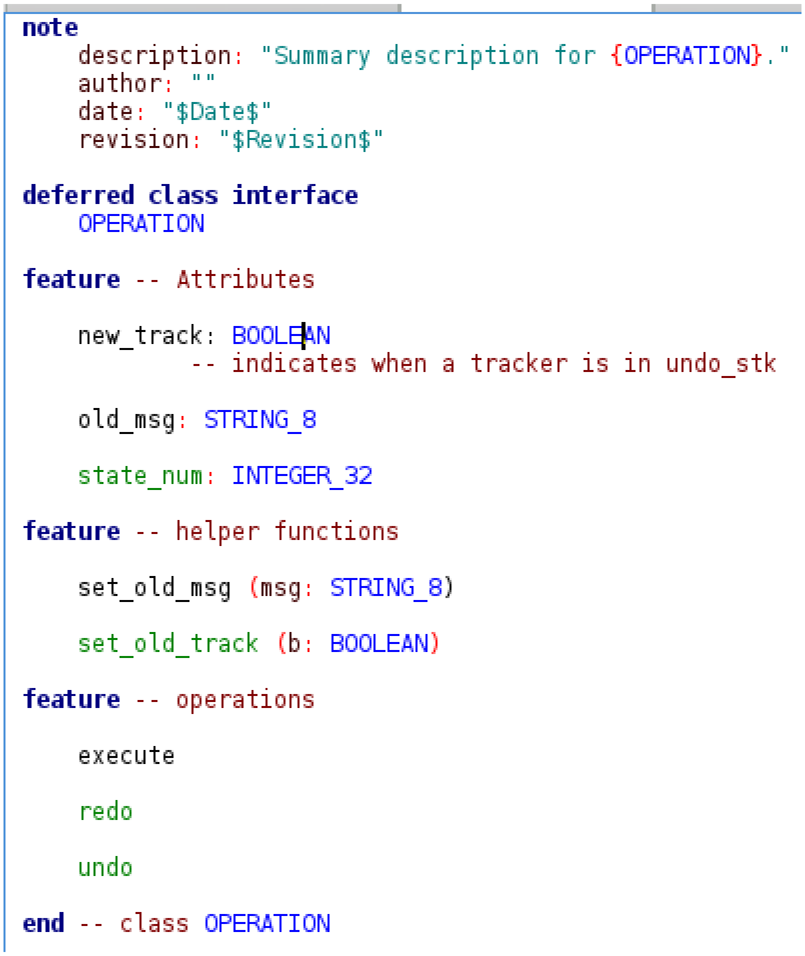


```
note
    description: "Summary description for {STATUS_MSG}."
    author: ""
    date: "$Date$"
    revision: "$Revision$"

class interface
    STATUS_MSG

create
    make_g

feature

    make_g

feature --error message

    Default_msg: STRING_8 = "ok"

    E1: STRING_8 = "e1: current tracker is in use"

    E10: STRING_8 = "e10: this container identifier already in tracker"

    E11: STRING_8 = "e11: this container will exceed phase capacity"

    E12: STRING_8 = "e12: this container will exceed phase safe radiation"

    E13: STRING_8 = "e13: phase does not expect this container material"

    E14: STRING_8 = "e14: container radiation capacity exceeded"

    E15: STRING_8 = "e15: this container identifier not in tracker"

    E16: STRING_8 = "e16: source and target phase identifier must be different"

    E17: STRING_8 = "e17: this container identifier is not in the source phase"
```



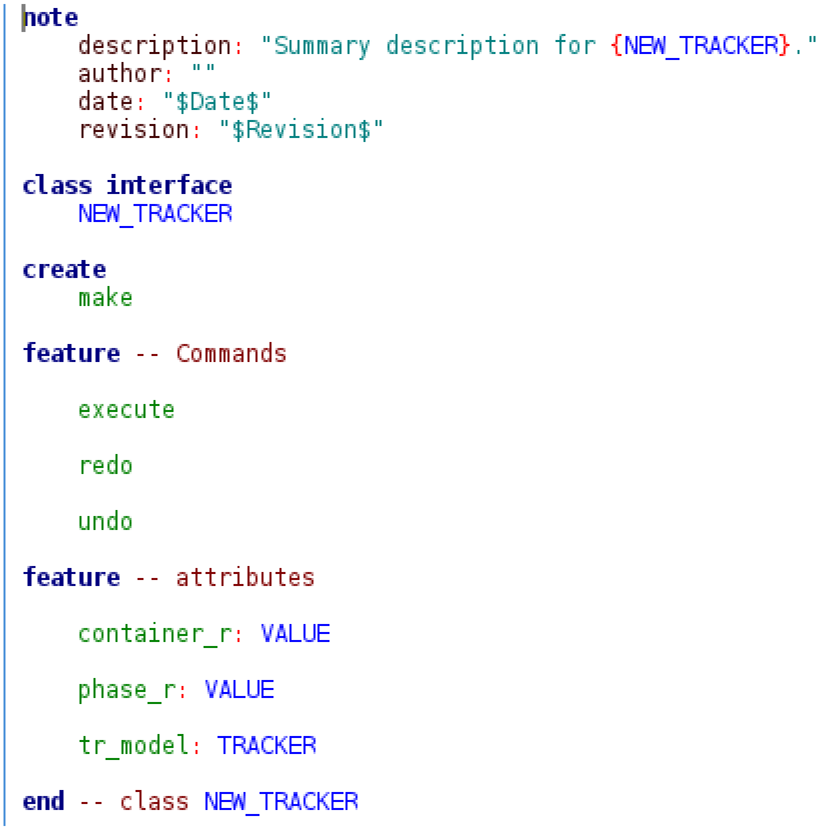
```
    E18: STRING_8 = "e18: this container radiation must not be negative"

    E19: STRING_8 = "e19: there is no more to undo"

    E2: STRING_8 = "e2: max phase radiation must be non-negative value"

    E20: STRING_8 = "e20: there is no more to redo"

    E3: STRING_8 = "e3: max container radiation must be non-negative value"

    E4: STRING_8 = "e4: max container must not be more than max phase radiation"

    E5: STRING_8 = "e5: identifiers/names must start with A-Z, a-z or 0..9"

    E6: STRING_8 = "e6: phase identifier already exists"

    E7: STRING_8 = "e7: phase capacity must be a positive integer"

    E8: STRING_8 = "e8: there must be at least one expected material for this phase"

    E9: STRING_8 = "e9: phase identifier not in the system"

end -- class STATUS_MSG
```

Operation:



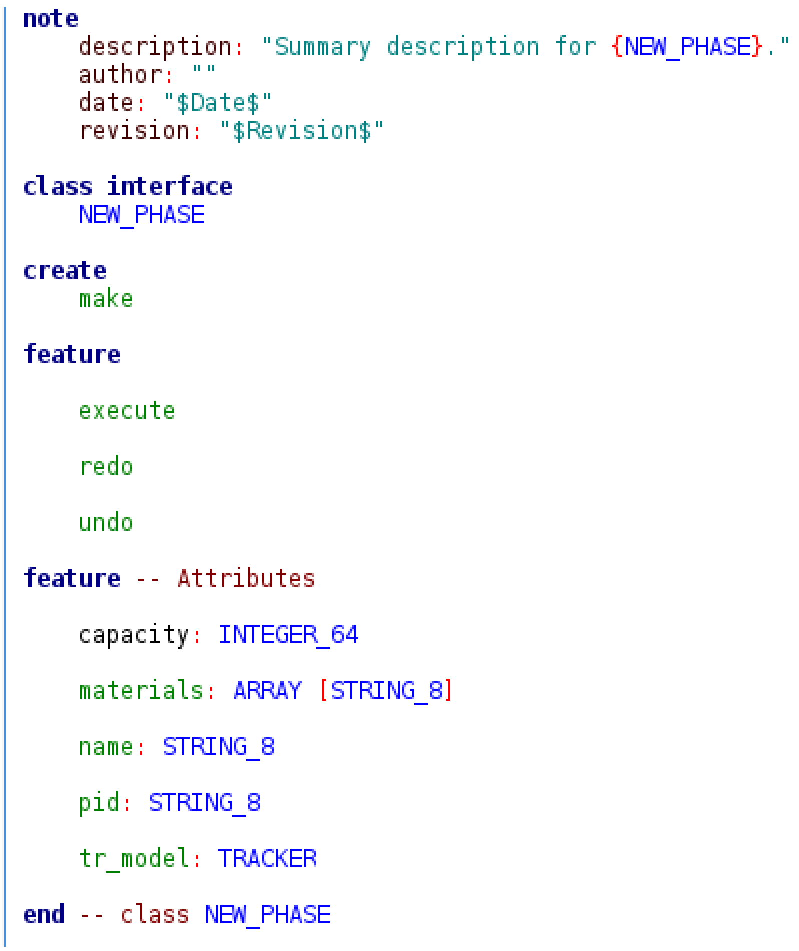
```
note
    description: "Summary description for {OPERATION}."
    author: ""
    date: "$Date$"
    revision: "$Revision$"

deferred class interface
    OPERATION

feature -- Attributes

    new_track: BOOLEAN
            -- indicates when a tracker is in undo_stk

    old_msg: STRING_8

    state_num: INTEGER_32

feature -- helper functions

    set_old_msg (msg: STRING_8)

    set_old_track (b: BOOLEAN)

feature -- operations

    execute

    redo

    undo

end -- class OPERATION
```

New_Tracker:



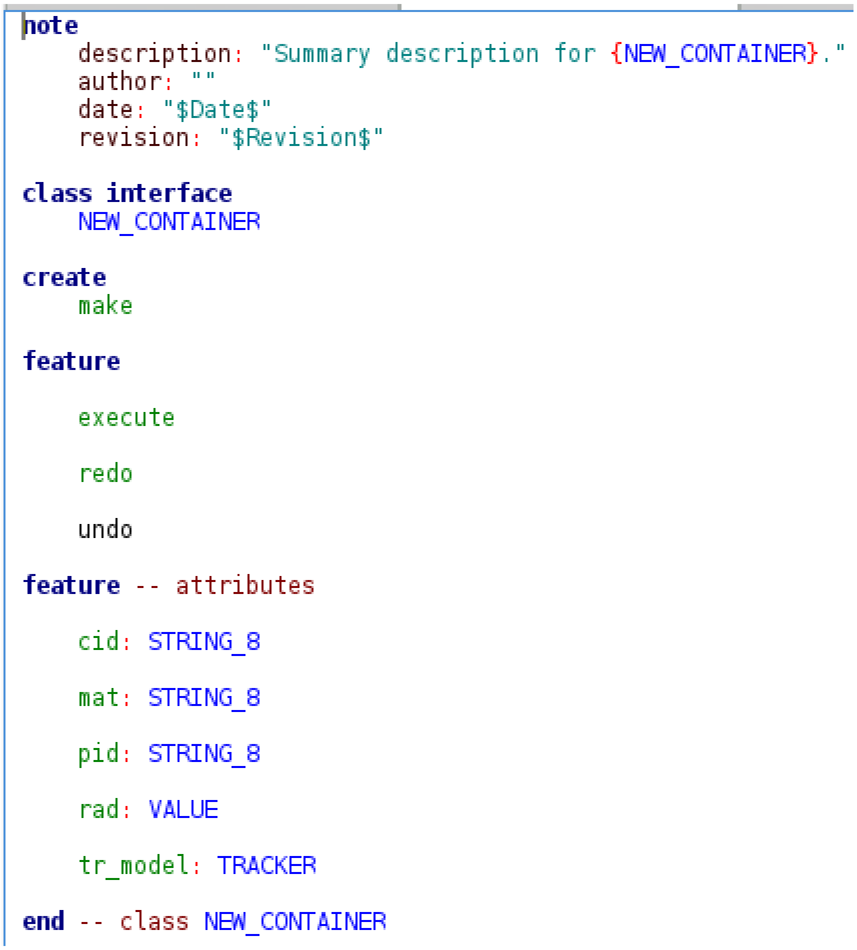
```
note
    description: "Summary description for {NEW_TRACKER}."
    author: ""
    date: "$Date$"
    revision: "$Revision$"

class interface
    NEW_TRACKER

create
    make

feature -- Commands

    execute

    redo

    undo

feature -- attributes

    container_r: VALUE

    phase_r: VALUE

    tr_model: TRACKER

end -- class NEW_TRACKER
```

New_Phase:



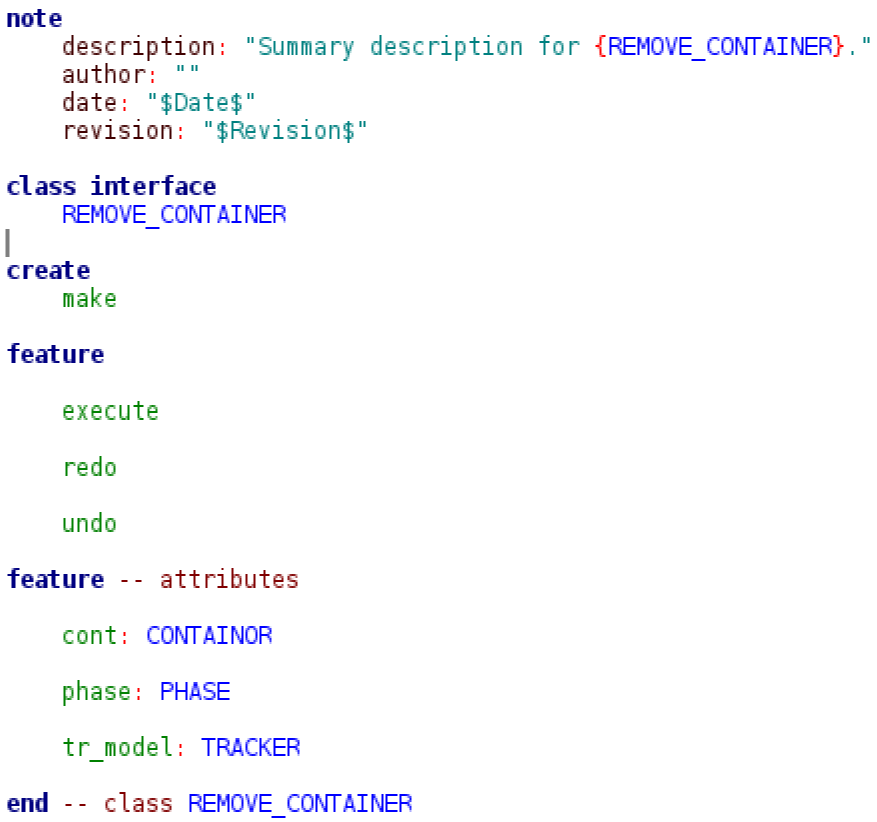
```
note
    description: "Summary description for {NEW_PHASE}."
    author: ""
    date: "$Date$"
    revision: "$Revision$"

class interface
    NEW_PHASE

create
    make

feature

    execute

    redo

    undo

feature -- Attributes

    capacity: INTEGER_64

    materials: ARRAY [STRING_8]

    name: STRING_8

    pid: STRING_8

    tr_model: TRACKER

end -- class NEW_PHASE
```

New_Container:



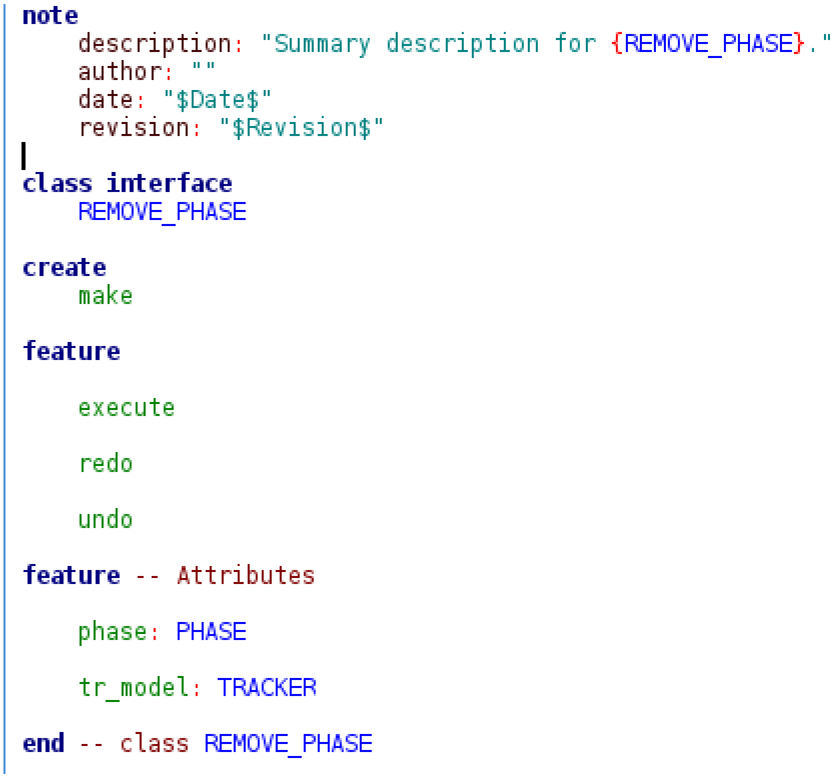
```
note
    description: "Summary description for {NEW_CONTAINER}."
    author: ""
    date: "$Date$"
    revision: "$Revision$"

class interface
    NEW_CONTAINER

create
    make

feature

    execute

    redo

    undo

feature -- attributes

    cid: STRING_8

    mat: STRING_8

    pid: STRING_8

    rad: VALUE

    tr_model: TRACKER

end -- class NEW_CONTAINER
```

Remove_Container:



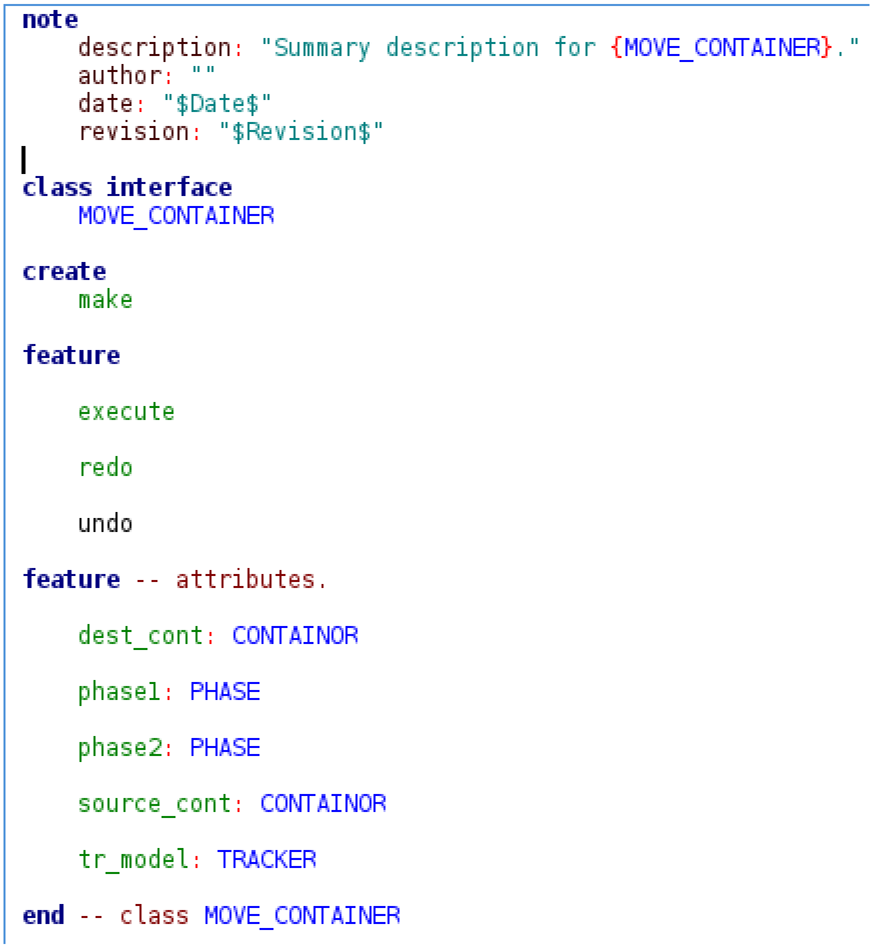
```
note
    description: "Summary description for {REMOVE_CONTAINER}."
    author: ""
    date: "$Date$"
    revision: "$Revision$"

class interface
    REMOVE_CONTAINER

create
    make

feature

    execute

    redo

    undo

feature -- attributes

    cont: CONTAINOR

    phase: PHASE

    tr_model: TRACKER

end -- class REMOVE_CONTAINER
```

Remove_Phase:



```
note
    description: "Summary description for {REMOVE_PHASE}."
    author: ""
    date: "$Date$"
    revision: "$Revision$"

class interface
    REMOVE_PHASE

create
    make

feature

    execute

    redo

    undo

feature -- Attributes

    phase: PHASE

    tr_model: TRACKER

end -- class REMOVE_PHASE
```

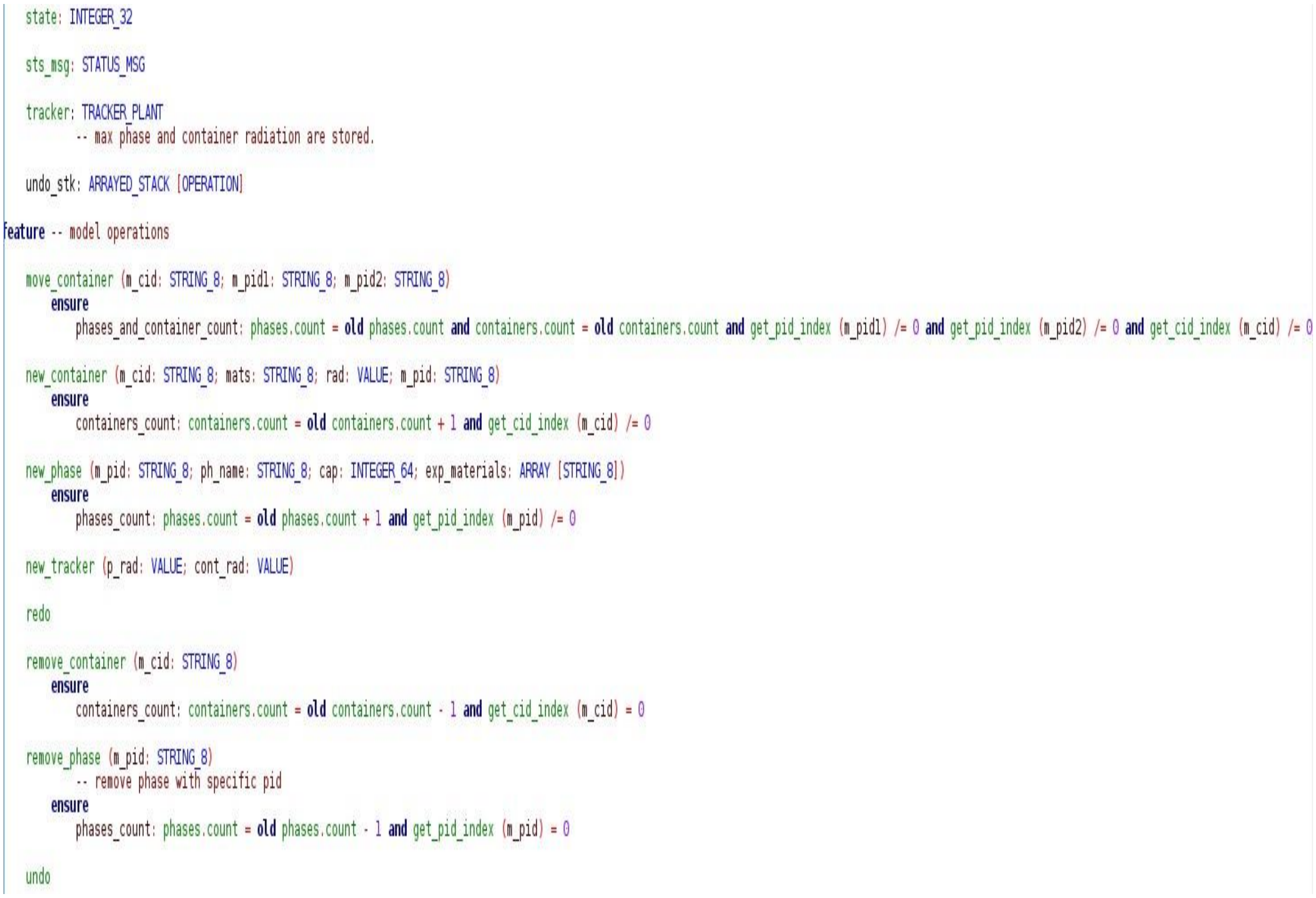
Move_Container:



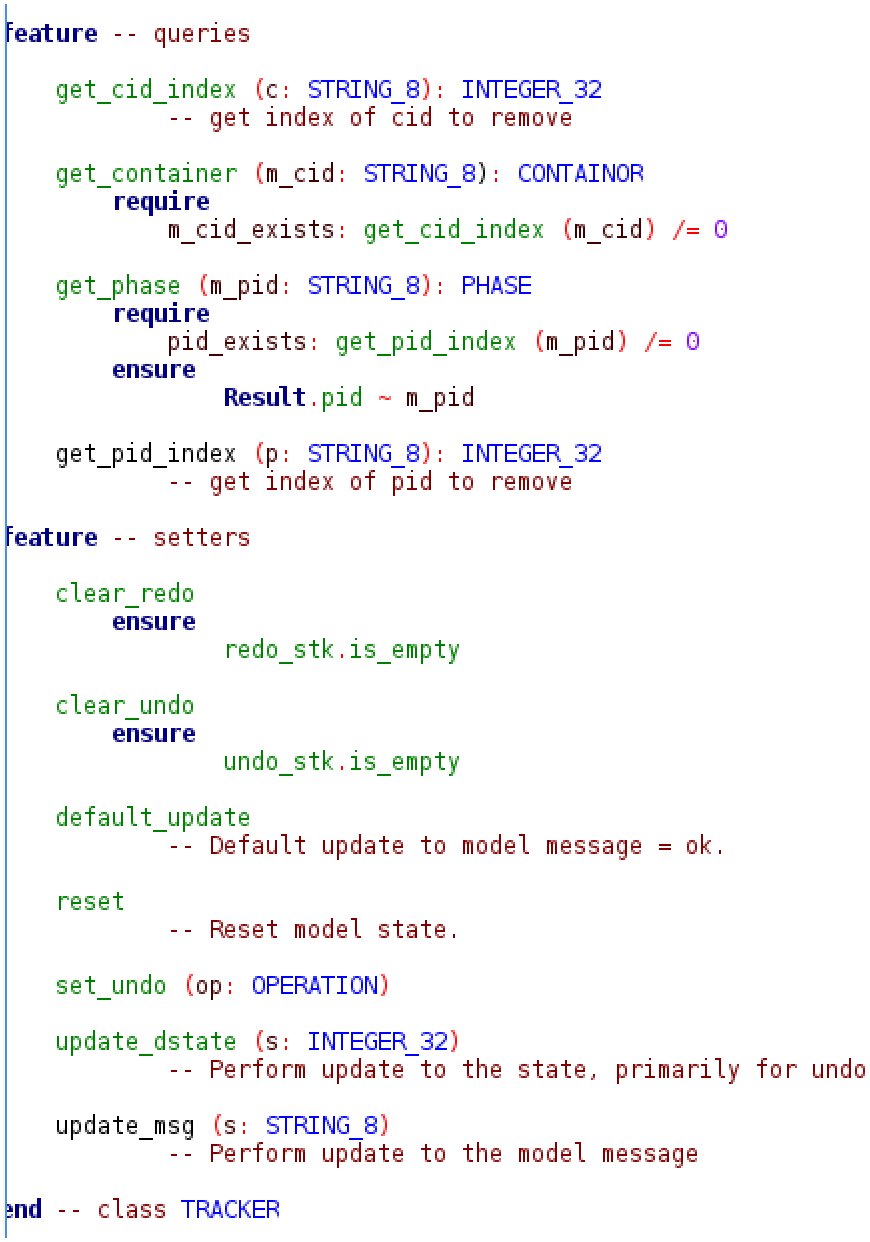
```
note
    description: "Summary description for {MOVE_CONTAINER}."
    author: ""
    date: "$Date$"
    revision: "$Revision$"

class interface
    MOVE_CONTAINER

create
    make

feature

    execute

    redo

    undo

feature -- attributes.

    dest_cont: CONTAINOR

    phase1: PHASE

    phase2: PHASE

    source_cont: CONTAINOR

    tr_model: TRACKER

end -- class MOVE_CONTAINER
```

Tracker:

```
note
    description: "A default business model."
    author: "Jackie Wang"
    date: "$Date$"
    revision: "$Revision$"

class interface
    TRACKER

create {TRACKER_ACCESS}
    make

feature -- Out

    containers_out: STRING_8

    out: STRING_8
            -- dest only in use for undo and redo, initializes back to 0. No tracker_out when error msg.

    phases_out: STRING_8

    tracker_plant_out: STRING_8
            -- print out tracker, phases and containers information when there is no error message

feature -- model attributes

    containers: SORTED_TWO_WAY_LIST [CONTAINOR]

    dest: STRING_8

    dstate: INTEGER_32

    message: STRING_8

    phases: SORTED_TWO_WAY_LIST [PHASE]

    redo_stk: ARRAYED_STACK [OPERATION]
```



```
    state: INTEGER_32

    sts_msg: STATUS_MSG

    tracker: TRACKER_PLANT
            -- max phase and container radiation are stored.

    undo_stk: ARRAYED_STACK [OPERATION]

feature -- model operations

    move_container (m_cid: STRING_8; m_pid1: STRING_8; m_pid2: STRING_8)
        ensure
            phases_and_container_count: phases.count = old phases.count and containers.count = old containers.count and get_pid_index (m_pid1) /= 0 and get_pid_index (m_pid2) /= 0 and get_cid_index (m_cid) /= 0

    new_container (m_cid: STRING_8; mats: STRING_8; rad: VALUE; m_pid: STRING_8)
        ensure
            containers_count: containers.count = old containers.count + 1 and get_cid_index (m_cid) /= 0

    new_phase (m_pid: STRING_8; ph_name: STRING_8; cap: INTEGER_64; exp_materials: ARRAY [STRING_8])
        ensure
            phases_count: phases.count = old phases.count + 1 and get_pid_index (m_pid) /= 0

    new_tracker (p_rad: VALUE; cont_rad: VALUE)

    redo

    remove_container (m_cid: STRING_8)
        ensure
            containers_count: containers.count = old containers.count - 1 and get_cid_index (m_cid) = 0

    remove_phase (m_pid: STRING_8)
            -- remove phase with specific pid
        ensure
            phases_count: phases.count = old phases.count - 1 and get_pid_index (m_pid) = 0

    undo
```



```
feature -- queries

    get_cid_index (c: STRING_8): INTEGER_32
            -- get index of cid to remove

    get_container (m_cid: STRING_8): CONTAINOR
        require
            m_cid_exists: get_cid_index (m_cid) /= 0

    get_phase (m_pid: STRING_8): PHASE
        require
            pid_exists: get_pid_index (m_pid) /= 0
        ensure
                Result.pid ~ m_pid

    get_pid_index (p: STRING_8): INTEGER_32
            -- get index of pid to remove

feature -- setters

    clear_redo
        ensure
                redo_stk.is_empty

    clear_undo
        ensure
                undo_stk.is_empty

    default_update
            -- Default update to model message = ok.

    reset
            -- Reset model state.

    set_undo (op: OPERATION)

    update_dstate (s: INTEGER_32)
            -- Perform update to the state, primarily for undo

    update_msg (s: STRING_8)
            -- Perform update to the model message

end -- class TRACKER
```