EECS 3311 - W18 - Lab 4 Report

Name	Prism Login	Signature
Joshua Phillip	jep4444	
Santiago Elustondo	Santi92	
Submitted under Prism account: jep4444		

Table of Contents

1. Requirements

The business requirement here is for an application to track containers of hazardous waste as they pass through various phases of treatment in a waste management facility.

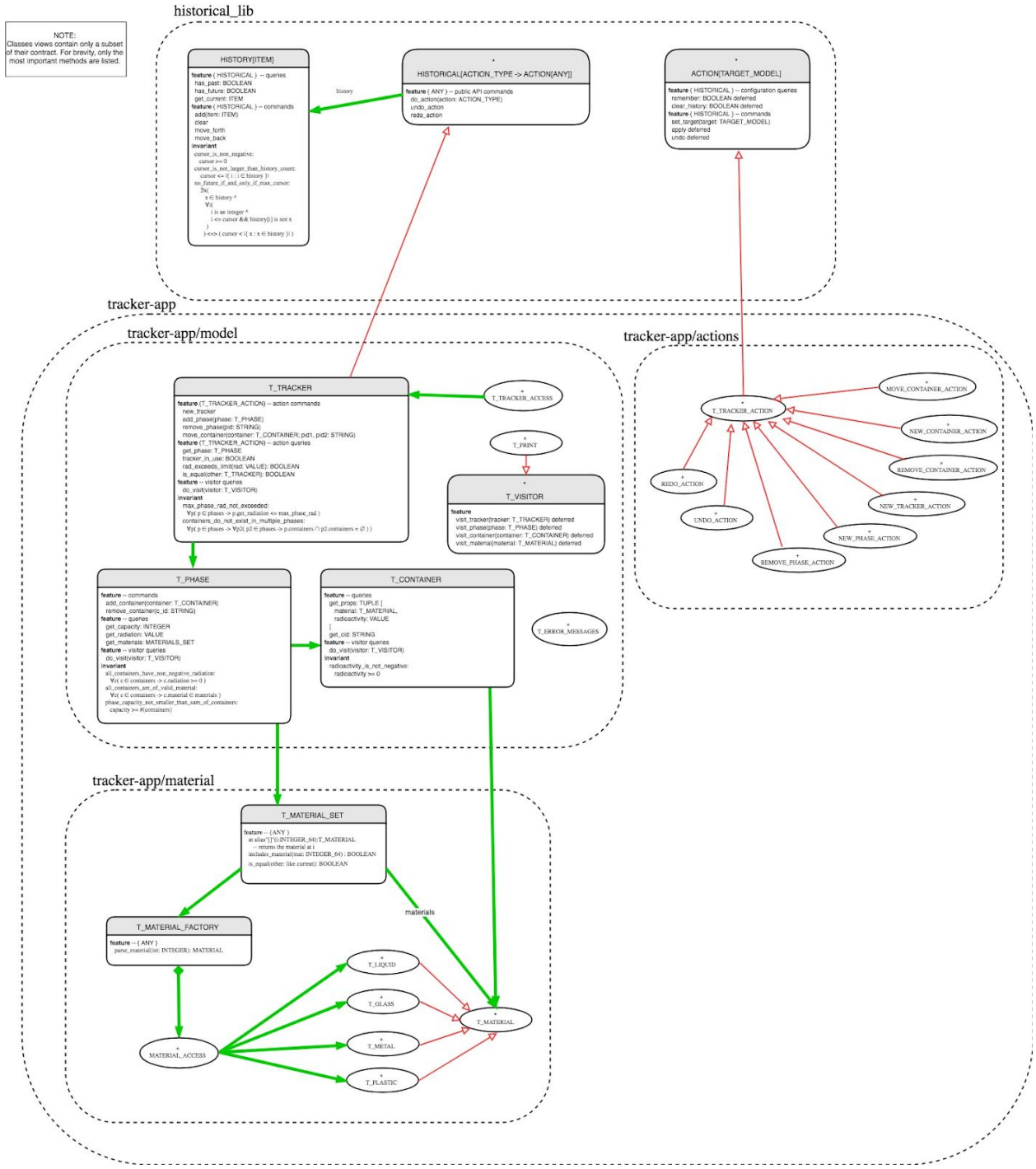
An operator needs to be able to configure the application for use in a particular facility, and then to register and track waste containers as they enter and move through the various configured phases.

Part of the configuration step is to input certain rules on the capacity of the facility to accommodate certain kinds and quantities of materials in each of its various stages. The application should ensure that these rules are not broken due to human error by disallowing actions that would break these safety rules.

Another feature requirement of the application is to support undo/redo functionality for its various actions.

It must be written using good design that may be extended safely and efficiently in the future, as further needs arise.

2. BON Diagram



historical_lib

This is a framework library that facilitates the creation of applications with undo/redo functionality. It provides classes that, when used by an application following the prescribed framework pattern (based on “Object Oriented Software Construction”), will give the application undo/redo functionality out of the box.

The prescribed framework pattern is as follows:

- All application state must be accessible from a **central state repository (CSR)** class**.** This class and its sub-component classes must implement all state representation for the application.
- All application state transitions must be defined as atomic **action (A)** classes, where each action defines **undo** and **redo** methods, having access to the central state repository. This is an example of ***polymorphism*** and ***dynamic binding.***

This is a robust pattern that can be used to create most types of applications.

historical_lib can be used to give undo/redo functionality to an application following the above pattern by:

- Having the central state repository class (CSR) inherit from HISTORICAL
- Having the action classes (A) inherit from ACTION
- Executing new actions by calling CSR.do_action(A)
- Undoing/redoing actions in historical order by calling CSR.undo_action() / CSR.redo_action()

historical_lib has been already used for previous applications in this course (notably the *tictac* app for lab 4), demonstrating the versatility of the library to provide undo/redo functionality while remaining agnostic of the actual client application.

tracker-app/model

*tracker-app'*s main module is the tracker-app/model cluster, which defines all the **stateful classes** that define tracker-app's **central state repository (CSR)**. The 3 stateful classes are T_TRACKER (which inherits from HISTORICAL), T_PHASE, and T_CONTAINER.

All 3 of these classes maintain some state, and thus expose **commands** and define **class invariants**. They are composed together to create tracker-app's CSR.

Each class also provides a visitor query (do_visit) to enable state-reading through a **visitor pattern.** A visitor (T_PRINT) is used to traverse the CSR to generate a string output after each action.

The CSR singleton (T_TRACKER class), is exposed to the ETF framework through the T_TRACKER_ACCESS expanded class, which maintains a single instance. The ETF

framework's ETF_COMMANDS, then, only generate a new T_TRACKER_ACTION (which inherits from ACTION) with the user's arguments and provide it to the CSR instance for consumption.

T_TRACKER maintains the top-level state of the application, most importantly a number of T_PHASE instances. The T_PHASE class represents the state of a phase, maintains a number of T_CONTAINER instances.

By keeping our application state in one cluster, we make our app more **maintainable** and **robust**. All our **critical system rules** can exist here in the form of class **invariants**, and the state classes can be **tested in isolation.**

tracker-app/actions

*tracker-app'*s action classes are held in this cluster. In order to differentiate *tracker-app*'s actions from any other class that might inherit from historical_lib's ACTION class, all of *tracker-app*'s actions inherit from T_TRACKER_ACTION, which inherits from historical_lib's ACTION.

These classes use a **polymorphic pattern,** and **dynamic binding** when received by T_TRACKER for consumption, as T_TRACKER will accept any T_TRACKER_ACTION, and call its apply, undo, or redo method, which each action will provide a different implementation for.

This approach is very **extendible**, since only by adding ACTION classes, we may add features to the system while maintaining undo/redo functionality and keeping the state definition decoupled from behaviour.

tracker-app/material

tracker-app's containers and phases need to define different types of hazardous materials. the tracker-app/materials cluster defines classes that provide the rest of the application with **material type definitions** and utilities.

The central deferred class T_MATERIAL is used to denote any material, while the effective classes T_GLASS, T_PLASTIC, etc, specify particular types of materials.

T_MATERIAL_FACTORY, together with the expanded T_MATERIAL_ACCESS class, implement a **factory pattern** that converts material types from their integer representation to their typed object representation, returning the same instance of each material type, so they can easily be checked for equality using any comparison operator.

This approach allows for using the material classes as types throughout the code, which **strengthens the class contracts**. The material types are versatile and may be compared at runtime with each other as well as at compile time.

3. Table of Modules

1	HISTORICAL [ACTION_TYPE -> ACTION[ANY]]	**Responsibility:** Serves as an inheritable interface that is used to generate a history.	**Alternative:** History could have been implemented directly inside tracker.
	Abstract	**Secret:** none	
1.1	HISTORY[G]	**Responsibility:** Concrete implementation of History to be used by historical	**Alternative:** none
	Concrete	**Secret:** none	

2	ACTION [TARGET_MODEL]	**Responsibility:** Serves as an abstract framework for all actions.	**Alternative:** History could have held T_TRACKER_ACTION objects instead.
	Abstract	**Secret:** none	
2.1	T_TRACKER_ACTION	**Responsibility:** Parent to all of trackers actions. Handles any common operations between actions. The children are the actions themselves.	**Alternative:** An error state could have been used instead of the error being of it's called type.
	Concrete	**Secret:** All tracker states are stored as their called type whether they succeed or not.	

3	T_TRACKER	**Responsibility:** Core structure of the programs implementation.	**Alternative:** Tracker could maintain its own list of containers instead of delegating to phases.
	Concrete	**Secret:** The Tracker has a collection of phases. Containers are handled inside each phase.	
3.1	T_PHASE	**Responsibility:** Each phase is responsible for tracking its own containers and compatible materials.	**Alternative:** none

	Concrete	**Secret:** none	
3.2	T_CONTAINER	**Responsibility:** Each container is responsible for tracking its own material and radioactivity.	**Alternative:** none
	Concrete	**Secret:** None	
3.3	T_MATERIAL_SET	**Responsibility:** Generating and storing a set of materials.	**Alternative:** Could have merely been a collection within phases.
	Concrete	**Secret:** None	

4	T_MATERIAL	**Responsibility:** A generic material type to be inherited by concrete types.	**Alternative:** Materials can be kept as integers until they need to be printed.
	Abstract	**Secret:** None	
4.1	T_MATERIAL_FACTORY	**Responsibility:** Used to map materials from ETF to an actual material object.	**Alternative:** none
	Concrete	**Secret:** None	
4.2	T_MATERIAL_ACCESS	**Responsibility:** Used to generate material singletons so each material only exists once.	**Alternative:** none
	Concrete	**Secret:** None	

5	T_VISITOR	**Responsibility:** A generic visitor type to be inherited by concrete types.	**Alternative:** none
	Abstract	**Secret:** None	
5.1	T_PRINT	**Responsibility:** Allows all the methods to delegate their output to an external method.	**Alternative:** Printing of each object can be handled within their own classes.
	Concrete	**Secret:** None	

4. Expanded Description

The most complex and business-critical part of *tracker-app* is the stateful classes of the tracker-app/model cluster. These are the most business-critical, since the **correctness** of our application is enforced here. The goal of the T_TRACKER, T_PHASE, and T_CONTAINER classes is to **encapsulate** the different layers of our application's state, hide all implementation details, and expose a safe API to their client such that they can be **composed** safely into one central state object. Since we are enforcing the business critical considerations of our application in this cluster, all methods require **strict, testable contracts and strong class invariants.**

Furthermore, all classes that make up the CSR for *tracker-app* expose a visitor query `do_visit` in order to make the entire application state traversable via **visitor pattern,** which provides an extendible way to read the application state for various purposes, like printing out responses to user actions or inspecting state snapshots during unit tests**.**

All three of the above-mentioned classes are written in the same manner and style, satisfying the same considerations, thus, we will focus on one of them here. T_PHASE is a good example to take since it acts as both a client (to T_CONTAINER) and provider (to T_TRACKER).

The state of T_PHASE can be only updated by its client/parent T_TRACKER through its command API. Since these affect the state of the application, they must be strictly checked for correctness. They both provide complete precondition and postcondition checks. The queries expose the state data to the client.

T_PHASE is a client of T_CONTAINER, and maintains a list of T_CONTAINER instances as part of its state. The inner workings of T_CONTAINER are unknown to T_PHASE. As per any client-provider relationship, T_CONTAINER does not hold a reference to its parent T_PHASE, and cannot update its state.

5. Significant Contacts

Safety Critical Invariants:

While there are a number of contracts pertaining to ensuring that goal 3 of section 3 is maintained, the following invariants are most pertinent so they are constantly verified.

- **No phase in the system shall have radiation greater than the maximum allowable.**
  This loops through all phases and ensures that no phase exceeds its limit for radiation

```
across get_phases as p all
        p.item.get_radiation <= get_max_phase_rad
end
```

- **No container in the system shall have radiation greater than the maximum allowable.**
  This loops through all containers to ensure no container exceeds its limit for radiation

```
across get_phases as p all
        across p.item.get_containers as c all
                not get_container_rad_exceeded(c.item.get_props.radioactivity)
        end
end
```

- **The count of containers in a phase shall not be greater than the maximum allowable.**
  This loops through all phases to ensure no phase exceeds its limit for containers

```
across get_phases as p all
        p.item.get_containers.count <= p.item.get_capacity
End
```

- **No phase handles an unexpected material.**
  This loops through all materials in all phases to make sure each phase only has expected materials.

```
across get_phases as p all
        across p.item.get_materials as m all
                p.item.get_materials.material_expected (m.item.get_mid)
        end
end
```

- **A container resides in only one phase:**
  This loops through all the phases and checks that as long as p1 and p2 are not the same phase, their intersection of range(which is all of its containers) is empty.

```
across get_phases as p1 all
        across get_phases as p2 all
                p1.item /= p2.item implies
                        (p1.item.model.range |/\| p2.item.model.range).is_empty
        end
end
```

Contracting to a model

T_TRACKER model: FUN [STRING, T_PHASE]
T_PHASE model: FUN [STRING, T_CONTAINER]

The above two models are important to verify the consistency of the system when phases/containers are added/removed. The lend themselves to the following contracts that correspond to ETF features.

new_tracker(a_max_phase_rad: VALUE; a_max_container_rad: VALUE)
ensure: model_unchanged: model ~ old model.deep_twin

add_phase (a_phase: T_PHASE)
require: phase_not_exists: not model.has ([a_phase.get_pid, a_phase])
ensure: phase_added: model ~ old model.deep_twin + [a_phase.get_pid, a_phase]

remove_phase(a_pid: STRING)
require: phase_exists: model.has ([a_pid, model[a_pid]])
ensure: phased_removed: model ~ old model.deep_twin - old [a_pid, get_phase(a_pid)]

add_container(a_container: T_CONTAINER)
require: container_doesnt_exist: not model.has ([a_container.get_cid, a_container])]
ensure: container_added: model ~ old model.deep_twin + [a_container.get_cid, a_container]

remove_container(a_cid: STRING)
require: has_container: model.has ([a_cid, model[a_cid]])
ensure: container_removed: model ~ old model.deep_twin - old [a_cid, get_container (a_cid)]

move_container(a_container: T_CONTAINER; a_pid1, a_pid2: STRING)
require: old_has_container: get_phase(a_pid1).model.has ([a_container.get_cid, a_container])
new_not_has_container: not get_phase(a_pid2).model.has ([a_container.get_cid, a_container])
ensure: container_removed_from_old: get_phase(a_pid1).model ~ old
get_phase(a_pid1).model - [a_container.get_cid, a_container]
container_added_to_new: get_phase(a_pid2).model ~ old get_phase(a_pid2).model +
[a_container.get_cid, get_phase(a_pid2).get_container (a_container.get_cid)]

While each of these have many important contracts that verify the integrity of inputs., The above contracts are crucial for showing that each ETF command is behaving as intended. If one of these were to work incorrectly, it would be difficult to reason about the integrity of the entire system. For example if add container did not work, no invariant could reasonably be sure that the radiation limit for phase was correct.

6. Summary of Testing Procedures

Test file	Description	Passed
at1.txt	Basic test, no errors	✓
at2.txt	A test designed for basic error checking	✓
at3.txt	A test designed to check error priorities	✓
at4.txt	Full test, many errors with undo/redo	✓

PASSED (18 out of 18)

Case Type	Passed	Total
Violation	12	12
Boolean	6	6
All Cases	18	18

State	Contract Violation	Test Name
Test1	STUDENT_TESTS	
PASSED	NONE	t0: Creation of Tracker
PASSED	NONE	t1: Create New Phase
PASSED	NONE	t2: Remove Phase
PASSED	NONE	*e2: max phase radiation must be non-negative value
PASSED	NONE	*e3: max container radiation must be non-negative value
PASSED	NONE	*e4: max container must not be more than max phase radiation
PASSED	NONE	*e5: identifiers/names must start with A-Z, a-z or 0..9
PASSED	NONE	*e6: phase identifier already exists
PASSED	NONE	*e8: phase capacity must be a positive integer
PASSED	NONE	t3: Create New Contaner
PASSED	NONE	t4: Move Container
PASSED	NONE	t5: Remove Container
PASSED	NONE	*e1: current tracker is in use
PASSED	NONE	*e9: phase identifier not in the system
PASSED	NONE	*e10: this container identifier already in tracker
PASSED	NONE	*e11: this container will exceed phase capacity
PASSED	NONE	*e12: this container will exceed phase safe radiation
PASSED	NONE	*e13: phase does not expect this container material

7. Appendix

```
deferred class interface
	HISTORICAL [ACTION_TYPE -> ACTION [ANY]]

end -- class HISTORICAL
```

```
class interface
	HISTORY [G -> ANY]

create
	make

feature -- model

	model: SEQ [G]

feature -- queries

	has_past: BOOLEAN

	has_future: BOOLEAN

	get_element: G

feature -- commands

	add (item: G)
		-- adds item to history
	ensure
		cursor_incremented: cursor = old cursor + 1
		current_item_is_new_one: get_element = item
		no_future: not has_future
		model ~ old model.deep_twin.subsequenced (1, cursor) |-> item

	prev_element
		-- goes back in history
	require
		not_first: has_past
	ensure
		cursor = old cursor - 1

	next_element
		-- goes forward in history
	ensure
		cursor = old cursor + 1

	clear_future
		-- removes all future elements of history
	ensure
		cursor_end_position: cursor = implementation.count
		no_future: model ~ old model.deep_twin.subsequenced (1, cursor)

	clear_all
		-- removes all of history
	ensure
		is_empty: model.is_empty
```

```
		cursor_end_position: cursor = implementation.count
		cursor_start_position: cursor = 0

invariant
	cursor_is_non_negative: cursor >= 0
	cursor_is_not_larger_than_history_count: cursor <= implementation.count
	no_future_if_and_only_if_max_cursor: (has_future implies (cursor < implementation.count)) and
		((cursor < implementation.count) implies has_future)

end -- class HISTORY
```

```
deferred class interface
	ACTION [TARGET_MODEL]

feature --private

	target: TARGET_MODEL

end -- class ACTION
```

```
deferred class interface
	T_TRACKER_ACTION

invariant
	prev_state_not_beyond_model: prev_state_id <= target.get_current_num_actions
	future_implies_not_last: target.get_history.has_future implies (prev_state_id < target.get_current_num_actions)
	past_implies_not_first: target.get_history.has_past implies (0 < target.get_current_num_actions)

end -- class T_TRACKER_ACTION
```

```
class interface
	T_TRACKER

create {ANY}
	Make, reset

feature -- model

	model: FUN [STRING_8, T_PHASE]
		-- abstraction function
	ensure
		model.is_function

feature -- commands

	new_tracker (a_max_phase_rad: VALUE; a_max_container_rad: VALUE)
		-- creates a new tracker with a_max_phase_rad and a_max_container_rad
	require
		tracker_not_in_use: not tracker_in_use
		max_phase_rad_is_positive: not (a_max_phase_rad.as_double < 0.0)
		max_container_rad_is_positive: not (a_max_container_rad.as_double < 0.0)
		max_phase_rad_is_not_smaller_than_max_container_rad: not (a_max_container_rad > a_max_phase_rad)
	ensure
		model_unchanged: model ~ old model.deep_twin

	add_phase (a_phase: T_PHASE)
		-- adds a_phase to tracker
	require
		tracker_not_in_use: not tracker_in_use
		pid_is_valid: not a_phase.get_pid.is_empty and then a_phase.get_pid [1].is_alpha_numeric
```

```
                name_is_valid: not a_phase.get_name.is_empty and then a_phase.get_name [1].is_alpha_numeric
                capacity_not_negative: not (a_phase.get_capacity <= 0)
                materials_expected: not (a_phase.get_materials.count = 0)
                phase_not_exists: not model.domain.has (a_phase.get_pid)
        ensure
                phase_added: model ~ old model.deep_twin + create {PAIR [STRING_8, T_PHASE]}.make_from_tuple
                        ([a_phase.get_pid, a_phase])

        remove_phase (a_pid: STRING_8)
                -- removes phase associated with a_pid from tracker
        require
                tracker_not_in_use: not tracker_in_use
                phase_exists: model.domain.has (a_pid)
        ensure
                phased_removed: model ~ old model.deep_twin - create {PAIR [STRING_8, T_PHASE]}.make_from_tuple
                        (old [a_pid, get_phase (a_pid)])

        move_container (a_container: T_CONTAINER; a_pid1, a_pid2: STRING_8)
                -- moves a_container from a_pid1 to a_pid2
        require
                container_doesnt_exist: get_phase (a_pid1).get_containers.has (a_container.get_cid)
                cid_is_valid: not a_container.get_cid.is_empty and then a_container.get_cid [1].is_alpha_numeric
                radioactivity_non_negative: not (a_container.get_props.radioactivity.as_double < 0.0)
                max_capacity_not_exceeded: not get_phase (a_pid2).max_capacity
                phase_rad_not_exceeded: not get_phase_rad_exceeded (a_pid2, a_container.get_props.radioactivity)
                material_expected: get_phase (a_pid2).get_materials.material_expected
                        (a_container.get_props.material.get_mid)
                old_has_container: get_phase (a_pid1).model.domain.has (a_container.get_cid)
                new_not_has_container: not get_phase (a_pid2).model.domain.has (a_container.get_cid)
        ensure
                container_removed_from_old: get_phase (a_pid1).model ~ old get_phase (a_pid1).model - create {PAIR
                        [STRING_8, T_CONTAINER]}.make_from_tuple ([a_container.get_cid, a_container])
                container_added_to_new: get_phase (a_pid2).model ~ old get_phase (a_pid2).model + create {PAIR
                        [STRING_8, T_CONTAINER]}.make_from_tuple ([a_container.get_cid, get_phase
                        (a_pid2).get_container (a_container.get_cid)])

        set_error (a_error: STRING_8)
                -- sets error message
        ensure
                error = a_error

        set_current_state_id (a_id: INTEGER_32)
                -- sets the id of the current state
        ensure
                current_state_id = a_id

        increment_num_actions
                -- incremements the action counter
        ensure
                current_num_actions = old current_num_actions + 1

feature -- public queries

        tracker_in_use: BOOLEAN
                -- returns whether tracker is in use
        ensure
                Result = across
                phases as p
                some
                p.item.get_containers.count /= 0
```

```
            end

    get_current_num_actions: INTEGER_32
            -- returns action counter
    ensure
            Result = current_num_actions

    get_current_state_id: INTEGER_32
            -- returns the current state id
    ensure
            Result = current_state_id

    get_error: STRING_8
            -- returns the current error message
    ensure
            Result = error

    get_max_phase_rad: VALUE
            -- returns the max radiation for phase
    ensure
            Result = max_phase_rad

    get_max_container_rad: VALUE
            -- returns the max radiation per container
    ensure
            Result = max_container_rad

    get_phase_rad_exceeded (pid: STRING_8; rad: VALUE): BOOLEAN
            -- returns whether max phase radiation is exceeded
    ensure
            get_phase (pid).get_radiation = old get_phase (pid).get_radiation
            Result = ((get_phase (pid).get_radiation + rad) > get_max_phase_rad)

    get_container_rad_exceeded (rad: VALUE): BOOLEAN
            -- returns whether max container radiation is exceeded
    ensure
            Result = (rad > max_container_rad)

    get_phase (pid: STRING_8): T_PHASE
            -- returns the phase associated with pid
    require
            pid_exists: model.domain.has (pid)
    ensure
            Result = model [pid]

    get_phases: STRING_TABLE [T_PHASE]
            -- returns a list of phases
    ensure
            Result = phases

    has_phase (pid: STRING_8): BOOLEAN
            -- returns whether phase associated with pid exists
    ensure
            Result = model.domain.has (pid)

    find_container (cid: STRING_8): detachable T_PHASE
            -- returns the phase associated with cid if it exists
    ensure
            attached Result implies Result.model.domain.has (cid)
```

```
		print_old_state: BOOLEAN
			-- prints the old state

		is_equal (other: like Current): BOOLEAN
			-- Is other attached to an object considered
			-- equal to current object?

feature -- print

		do_visit (visitor: T_VISITOR)

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

invariant
		max_phase_rad_not_smaller_than_max_container_rad: not (max_phase_rad < max_container_rad)
		max_rads_are_positive: not (max_container_rad.as_double < 0.0)
		current_num_actions_not_smaller_than_current_state_id: not (current_num_actions < current_state_id)
		current_actions_and_state_id_are_not_negative: not (current_num_actions < 0)
		capacity_not_exceeded:
			across get_phases as p all
			p.item.get_containers.count.to_integer_64 <= p.item.get_capacity
		end
		phase_rad_not_exceeded:
			across get_phases as p all
			p.item.get_radiation <= get_max_phase_rad
		end
		con_rad_not_exceeded:
			across get_phases as p all
				across p.item.get_containers as c all
					not get_container_rad_exceeded (c.item.get_props.radioactivity)
				end
			end
		mat_expected:
			across get_phases as p all
				across p.item.get_materials as m all
					p.item.get_materials.material_expected (m.item.get_mid)
				end
			end
		con_in_only_one_phase:
			across get_phases as p1 all
				across get_phases as p2 all
					p1.item /= p2.item implies (p1.item.model.range |/\| p2.item.model.range).is_empty
				end
			end

end -- class T_TRACKER
```

```
class interface
		T_PHASE

create
		make

feature -- commands

		add_container (a_container: T_CONTAINER)
			-- adds a_container to tracker
		require
```

```
                cid_is_valid: not a_container.get_cid.is_empty and then a_container.get_cid [1].is_alpha_numeric
                radioactivity_non_negative: not (a_container.get_props.radioactivity.as_double < 0.0)
                max_capacity_not_exceeded: not max_capacity
                material_expected: get_materials.material_expected (a_container.get_props.material.get_mid)
                container_doesnt_exist: not model.domain.has (a_container.get_cid)
        ensure
                container_added: model ~ old model.deep_twin + create {PAIR [STRING_8,
T_CONTAINER]}.make_from_tuple ([a_container.get_cid, a_container])

        remove_container (a_cid: STRING_8)
                -- removes container associated with a_cid from tracker
        require
                cid_is_valid: not a_cid.is_empty and then a_cid [1].is_alpha_numeric
                has_container: model.domain.has (a_cid)
        ensure
                container_removed: model ~ old model.deep_twin - create {PAIR [STRING_8,
T_CONTAINER]}.make_from_tuple (old [a_cid, get_container (a_cid)])

feature -- model

        model: FUN [STRING_8, T_CONTAINER]
                -- abstraction function
        ensure
                model.is_function

feature -- queries

        get_pid: STRING_8
                -- returns the pid of the phase
        ensure
                Result = pid

        get_name: STRING_8
                -- returns the name of the phase
        ensure
                Result = name

        get_capacity: INTEGER_64
                -- returns the capacity of the phase
        ensure
                Result = capacity

        get_radiation: VALUE
                -- returns the current radiation of the phase

        get_materials: T_MATERIAL_SET
                -- returns the set of materials that can be used in the phase
        ensure
                Result = materials

        get_container (cid: STRING_8): T_CONTAINER
                -- returns a container associated with cid
        ensure
                attached Result implies Result = model [cid]

        get_containers: STRING_TABLE [T_CONTAINER]
                -- returns a list of containers
        ensure
                Result = containers
        max_capacity: BOOLEAN
```

```
                -- returns the max capacity of the phase
        ensure
                Result = (get_containers.count.to_integer_64 = get_capacity)

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

        is_equal (other: like Current): BOOLEAN
                -- Is other attached to an object of the same type
                -- as current object and identical to it?

        do_visit (visitor: T_VISITOR)
                -- allows a visitor to visit Current

feature -- print

        print_phase: STRING_8

invariant
        all_containers_have_non_negative_radiation:
                across containers as c all
                        c.item.get_props.radioactivity.as_double >= 0.0
                end
        all_containers_are_of_valid_material:
                across containers as c all
                        across materials as m some
                                c.item.get_props.material = m.item
                        end
                end
        capacity_not_smaller_than_sum_of_containers: not (capacity < containers.count.to_integer_64)

end -- class T_PHASE
```

```
class interface
        T_CONTAINER

create
        make

feature -- queries

        get_props: TUPLE [material: T_MATERIAL; radioactivity: VALUE]
                -- returns the props of container
        ensure
                Result = props

        get_cid: STRING_8
                -- returns the cid of container
        ensure
                Result = cid

        get_pid: STRING_8
                -- returns the pid that container is in
        ensure
                Result = pid

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

        is_equal (other: like Current): BOOLEAN
```

```
                -- Is other attached to an object of the same type
                -- as current object and identical to it?

feature -- print

        do_visit (visitor: T_VISITOR)

invariant
        radioactivity_is_not_negative: not (props.radioactivity.as_double < .0)

end -- class T_CONTAINER
```

```
class interface
        T_MATERIAL_SET

create
        make

feature -- model

        model: SET [T_MATERIAL]

feature --queries

        count: INTEGER_32
                --returns number of materials in set

        material_expected (mat: INTEGER_64): BOOLEAN
                -- returns whether material is acceptable to the set

        at alias "[]" (i: INTEGER_64): T_MATERIAL
                -- returns the material at i

        is_equal (other: like Current): BOOLEAN
                -- Is other attached to an object considered
                -- equal to current object?

        as_array: ARRAY [T_MATERIAL]
                -- returns set as an array

        as_integer_array: ARRAY [INTEGER_64]
                -- returns set as an array of ints

feature --commands

        add_materials (a_materials: ARRAY [INTEGER_64]): LINKED_SET [T_MATERIAL]
                -- takes an array of a_material and produces a set

feature -- print

        do_visit (visitor: T_VISITOR)

invariant
        all_elements_unique:
                across materials as i all
                        across materials as j all
                                (i.item ~ j.item) implies (i.cursor_index = j.cursor_index)
                        end
                end
end -- class T_MATERIAL_SET
```

```
deferred class interface
	T_MATERIAL

feature -- queries

	get_name: STRING_8
		-- name of material

	get_mid: INTEGER_64
		-- id of material

end -- class T_MATERIAL
```

```
expanded class interface
	T_MATERIAL_FACTORY

feature

	parse_material (a_material: INTEGER_64): T_MATERIAL
		-- converts a_material to a material

end -- class T_MATERIAL_FACTORY
```

```
expanded class interface
	T_MATERIAL_ACCESS

feature

	Glass: T_GLASS
		-- singleton for glass

	Liquid: T_LIQUID
		-- singleton for liquid

	Metal: T_METAL
		-- singleton for metal

	Plastic: T_PLASTIC
		-- singleton for plastic

invariant
	Glass = Glass
	Liquid = Liquid
	Metal = Metal
	Plastic = Plastic

end -- class T_MATERIAL_ACCESS
```

```
deferred class interface
	T_VISITOR

feature

	visit_phase (phase: T_PHASE)
		-- handles output for phase

	visit_container (con: T_CONTAINER)
		-- handles output for con

	visit_materials (mat: T_MATERIAL_SET)
```

```
                -- handles output for mat

        visit_tracker (tracker: T_TRACKER)
                -- handled output for tracker

end -- class T_VISITOR
```

```
class interface
        T_PRINT

create
        make

feature

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

feature

        visit_phase (phase: T_PHASE)
                -- handles output for phase

        visit_container (con: T_CONTAINER)
                -- handles output for con

        visit_materials (mat: T_MATERIAL_SET)
                -- handles output for mat

        visit_tracker (tracker: T_TRACKER)
                -- handled output for tracker

end -- class T_PRINT
```