1 Contracts

ETH students recently designed a special kind of oven for cooking potatoes. Here are some facts about such an oven:

- each oven is equipped with a door which is either open or closed;
- the oven is fairly small, therefore only one potato can fit inside;
- it is only possible to put a potato in or take one out when the door is open;
- to start or stop cooking, one has to use the start/stop switch;
- for safety reasons, the oven would not start cooking if its door is open or there is nothing to cook;
- the door cannot be opened during cooking: cooking has to be stopped first.

The following class POTATO_OVEN models such an oven. Please fill in the missing contracts (preconditions, postconditions, and class invariants), so that each fact from the informal specification above is reflected in the class interface.

Please note the number of dotted lines does not indicate the number of missing contracts.

defered class
POTATO_OVEN

feature -- Access

potato_to_cook: POTATO
-- The potato inside the oven.

feature -- Status report

is_door_open: BOOLEAN
-- Is the oven door open?

is_cooking: BOOLEAN
-- Is the oven cooking?

is_empty: BOOLEAN
-- Is the oven empty?
defered
ensure
    Result = (potato_to_cook = Void)
end

feature -- Basic operation

    open_door
    -- Open the door.
    require
    deferred
    ensure
    end

    close_door
    -- Close the door.
    require
    deferred
    ensure

end

\textit{put} (\texttt{a\_potato}: \texttt{POTATO})
\hspace{1cm} -- Put `a\_potato' into the oven.
\textbf{require}

end

\textit{deferred}
\textbf{ensure}

end

\textit{remove}
\hspace{1cm} -- Remove the potato.
\textbf{require}

\textbf{deferred}
\textbf{ensure}
end

switch_on
-- Turn on the start/stop switch.
require

defered
ensure

end

switch_off
-- Turn off the start/stop switch.
require

defered
ensure
end

invariant

……………………………………………………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………………………………………………

end
2 Inheritance

Below you see the class \texttt{GAME\_CHARACTER}. The class represents game characters. There are three types of game characters: dragon, marshmallow man and zombie. Every character has a health level in the range of 0 to 100, where 0 means that the character is dead and 100 that it has full strength. Since zombies are dead by definition, their health level stays at 0 at all times. Each of the character types has a damage potential that it can inflict on others. For all of them the damage doubles if the character is angry.

Listing 1: Class \texttt{GAME\_CHARACTER}

```plaintext
class GAME\_CHARACTER

create
make

feature -- Initialization
make (t: INTEGER)
   -- Initialize with type ‘t’.
require
t\_valid: (t = marshmallow\_man xor t = dragon xor t = zombie) and not (t = marshmallow\_man and t = dragon and t = zombie)
do
type := t
if type = zombie then
   health := 0
else
   health := 100
end
ensure
type\_set: type = t
end

feature -- Access
type: INTEGER
   -- Type of character

health: INTEGER
   -- Health of character (0: dead, 100: full strength)

damage: INTEGER
   -- Damage that the character can do
do
if type = zombie then
   Result := zombie\_damage
elseif type = marshmallow\_man then
   Result := marshmallow\_man\_damage
else
   Result := dragon\_damage
end
```

if is_angular then
    Result := Result * 2
end

ensure
    zombie: not is_angular and type = zombie implies Result = zombie_damage
    angry_zombie: is_angular and type = zombie implies Result = 2*zombie_damage
    dragon: not is_angular and type = dragon implies Result = dragon_damage
    angry_dragon: is_angular and type = dragon implies Result = 2*dragon_damage
    marshmallow_man: not is_angular and type = marshmallow_man implies Result = marshmallow_man_damage
    angry_marshmallow_man: is_angular and type = marshmallow_man implies Result = 2*marshmallow_man_damage
end

feature -- Status report

is_dead: BOOLEAN
    -- Is the character dead?
do
    Result := (health = 0)
ensure
    Result_set: Result = (health = 0)
end

is_angular: BOOLEAN
    -- Is the character angry?
    -- (Then it can do more damage!)

feature -- Element change

set_health (h: INTEGER)
    -- Set 'health' to 'h'.
require
    h_valid: h >= 0 and h <= 100
    h_for_zombie: type = zombie implies h = 0
do
    health := h
ensure
    health_set: health = h
end

set_angular (b: BOOLEAN)
    -- Set 'is_angular' to 'b'.
do
    is_angular := b
ensure
    is_angular_set: is_angular = b
end

feature -- Constants

marshmallow_man: INTEGER = 1
−− Marshmallow man

94 dragon: INTEGER = 2
  −− Dragon

96 zombie: INTEGER = 3
  −− Zombie (is always dead)

98 zombie_damage: INTEGER = 1
  −− Damage that a zombie does

100 dragon_damage: INTEGER = 2
  −− Damage that a dragon does

104 marshmallow_man_damage: INTEGER = 3
  −− Damage that a marshmallow man does

108 invariant

112 type_valid: (type = marshmallow_man xor type = dragon xor type = zombie) and not (type = marshmallow_man and type = dragon and type = zombie)

114 health_valid: health >= 0 and health <= 100

116 zombie_always_dead: type = zombie implies health = 0

end

The above code does not exhibit a nice object-oriented design and it can hardly be called reusable. Redesign the code such that it uses inheritance instead of the type attribute to represent the three types of game characters. Write a deferred ancestor class NEW_GAME_CHARACTER and effective descendants ZOMBIE, MARSHMALLOW_MAN, and DRAGON that inherit from NEW_GAME_CHARACTER.

Your design should

• result in the deletion of the type attribute.

• result in the same behavior for the three types of game characters as the original code of class GAME_CHARACTER.

• include semantically equivalent contracts as the original code of class GAME_CHARACTER.

If a feature stays the same in your re-factored code as in the original code, please indicate it by giving the full feature signature and adding a comment -- See original.

Example:

is_dead: BOOLEAN
  −− See original.
deferred class NEW_GAME_CHARACTER
class ZOMBIE

end
class MARSHMALLOW_MAN

end
class DRAGON

end