

## Solution 8: Polymorphic Behaviors

ETH Zurich

### 1 Dynamic binding and polymorphic attachment

#### Task 1.1

Does the code compile?  Yes  No

The feature *make\_with\_device* is unknown in class *CAR\_DRIVER*.

#### Task 1.2

Does the code compile?  Yes  No

Creation instruction applies to a deferred type.

#### Task 1.3

Does the code compile?  Yes  No

"Julie walks 0.5 km"

#### Task 1.4

Does the code compile?  Yes  No

1. creation on a deferred type and second explicit type for creation does not conform to target type.

#### Task 1.5

Does the code compile?  Yes  No

Source of assignment does not conform to target type.

#### Task 1.6

Does the code compile?  Yes  No

Unknown feature drive.

#### Task 1.7

Does the code compile?  Yes  No

"Megan drives Renault 17.8 km"

### 2 Ghosts in Paris

Solution

Listing 1: Class `TRAFFIC_GHOST`

```
class
2  TRAFFIC_GHOST

4 inherit

6  TRAFFIC_FREE_MOVING
   redefine
8   move_next
   end

10
   create
12  make

14 feature -- Initialization

16  make (a_station: TRAFFIC_STATION; a_side: REAL)
   -- Initialize positions to start.
18  require
   a_station_exists : a_station /= Void
20  a_side_positive : a_side > 0.0
   local
22  l: DS_ARRAYED_LIST [TRAFFIC_POINT]
   p: TRAFFIC_POINT
24  x, y: REAL
   do
26  create l.make (5)
   x := a_station.location.x
28  y := a_station.location.y
   create p.make (x - a_side/2, y - a_side/2)
30  l.put_last (p)
   create p.make (x + a_side/2, y - a_side/2)
32  l.put_last (p)
   create p.make (x + a_side/2, y + a_side/2)
34  l.put_last (p)
   create p.make (x - a_side/2, y + a_side/2)
36  l.put_last (p)
   create p.make (x - a_side/2, y - a_side/2)
38  l.put_last (p)

40  make_with_points (l, 10.0)

42  set_reiterate (True)
   ensure
44  reiterating : is_reiterating
   end

46
   feature {NONE} --Implementation
48  move_next
50  -- Move to following position
   do
```

```
52  -- Set the locations to the corresponding ones of the line segment.
    origin := poly_cursor.item
54  location := poly_cursor.item
    if is_reiterating then
56      poly_cursor.forth
        if poly_cursor.after then
58            poly_cursor.start
                move_next
60        else
            destination := poly_cursor.item
62        end
    else
64        poly_cursor.forth
            if poly_cursor.after then
66                has_finished := True
            else
68                destination := poly_cursor.item
            end
70        end
    end
72 end
end
```

Listing 2: Class *GHOST\_INVASION*

```
1 class
    GHOST_INVASION
3
    inherit
5
    TOURISM
7
    feature -- Explore Paris
9
        invade
11         -- Invade Paris with 10 ghosts.
            local
13             g: TRAFFIC_GHOST
                r: RANDOM
15             t: TIME
                i: INTEGER
17             a: ARRAY [TRAFFIC_STATION]
            do
19             Paris.display
                wait
21
                create t.make_now
23             create r.set_seed ((t.fine_seconds*1000).truncated_to_integer)
                from
25                 i := 1
                    r.start
27                 a := Paris.stations.to_array
            until
```

```
29     i > 10
      loop
31     create g.make (a.item ((r.item_for_iteration \\ a.count) + 1), 50.0)
      g.start
33     Paris.put_free_moving (g)
      r.forth
35     i := i + 1
      end
37 end
39 end
```

### 3 Programming a boardgame: Part 3

#### Solution

Go through the code and test it on your machine. Some print statements have been added to facilitate your task. Notice polymorphism at work in feature *location.land\_on* (*Current*).

Listing 3: Class *GAME*

```
1 class
  GAME
3
  create
5  make

7 feature {NONE} -- Initialization

9  make
  -- Run application.
11 local
  i: INTEGER
13  p: PLAYER
  do
15  create game_board.make
  create die_1.make
17  create die_2.make
  create players.make (1, number_of_players)
19  from
  i := 1
21  until
  i > players.count
23  loop
  create p.make ("Player" + i.out)
25  p.credit (player_initial_amount)
  p.set_location (game_board.start_square)
27  players [i] := p
  i := i + 1
29  end
  end
31
33 feature -- Basic operations
```

```

35  play
    -- Start a game.
    local
37  i: INTEGER
    do
39  print ("%N*** Simple Boardgame ***")
    from
41
    until
43  has_winner
    loop
45  from
        i := 1
47  until
        has_winner or else i > number_of_players
49  loop
        players [i].play (die_1, die_2)
51  if players [i].location = Void then
        find_winner
53  from
        winners.start
55  until
        winners.after
57  loop
        print ("%The winner(s): " + winners.item.name + " with money: " +
            winners.item.money.out + " CHF")
59  has_winner := True
        winners.forth
61  end
    end
    i := i + 1
65  end
    end
67  print ("%N*** Game Over ***")
    ensure
69  game_has_winner: has_winner
    end
71
find_winner
73  -- Determine winner.
    local
75  i: INTEGER
    do
77  from
        i := 1
79  create winners.make
        winners.extend (players [i])
81  i := i + 1
    invariant
83  i >= 2
        i <= number_of_players + 1
    
```

```

85     until
      i > number_of_players
87     loop
      if players [i].money > (winners.i_th (winners.count)).money then
89         winners.wipe_out
          winners.extend (players [i])
91     else if players [i].money = (winners.i_th (winners.count)).money then
          winners.extend (players [i])
93     end
      end
95     i := i + 1
      variant
97         number_of_players - i + 1
      end
99     end

101 feature -- Status

103 game_board: BOARD
      -- The game board.
105
106 number_of_players: INTEGER = 2
107     -- For testing purposes, the number of players is set to 2.

109 players: ARRAY [PLAYER]
      -- Container for players.
111
112 player_initial_amount: INTEGER = 7
113     -- Players initial amount.

115 die_1: DIE
      -- The first die.
117
118 die_2: DIE
119     -- The second die.

121 has_winner: BOOLEAN
      -- Does the game have a winner?
123
124 winners: LINKED_LIST [PLAYER]
125     -- The winner(s) of the game.

127 invariant

129 game_board_exists: game_board /= Void

131 players_exist: players /= Void and then not players.is_empty

133 number_of_players_consistent: number_of_players >= 2 and number_of_players <= 6

135 dice_exist: die_1 /= Void and die_2 /= Void
end
    
```

Listing 4: Class *DIE*

```
class
2  DIE

4 create
  make
6

8 feature {NONE} -- Initialization

10 make
  -- Create a die with valid initial face value.
12 do
  face_value := rand.item \\ 6 + 1
14 end

16 feature -- Access

18 face_value: INTEGER

20 feature -- Basic operations

22 roll
  -- Roll die
24 do
  rand.forth
26 face_value := rand.item \\ 6 + 1
  end
28

feature {NONE} -- Implementation
30
  rand: RANDOM
32  -- Pseudo-random number generator.
  local
34    t: TIME
    seed: INTEGER
36  once
    create t.make_now
38    seed := (t.fine_seconds * 1000).rounded
    create Result.set_seed (seed)
40    Result.start
  end
42

44 invariant
  six_sided_die : face_value >= 1 and face_value <= 6
46

end
```

Listing 5: Class *PLAYER*

```
1 class
```

```

3  PLAYER
create
5  make
7
9  feature -- Access
11  name: STRING
    -- Player name.
13  location: SQUARE
    -- 'Current' location.
15  money: INTEGER
17  -- player current amount of money.
19 feature {NONE} -- Initialization
21  make (n: STRING)
    -- Create a player with name.
23  require
    name_exists: n /= Void and then not n.is_empty
25  do
    name := n
27  ensure
    name_set: name = n
29  end
31 feature -- Status setting
33  set_location (loc: SQUARE)
    -- Set location for 'Current'.
35  require
    location_exists : loc /= Void
37  do
    location := loc
39  ensure
    location_set : location = loc
41  end
43 feature -- Basic operations
45  play (d1,d2: DIE)
    -- Play a turn.
47  require
    dice_exist : d1 /= Void and d2 /= Void
49  local
    dice_result : INTEGER
51  do
    d1.roll
    print ("%Nd1:" + d1.face_value.out)
53
```

```

    d2.roll
55    print ("%Nd2:" + d2.face_value.out)
        dice_result := d1.face_value + d2.face_value
57    move (dice_result)
    ensure
59    player_moved: old location /= location
    end
61
move (n: INTEGER)
63    -- Move 'Current' n steps forward.
    require
65    n_consistent: n >= 2 and n <= 12
    local
67    i: INTEGER
    do
69    from
        i := 1
71    until
        location = Void or else i > n
73    loop
        location := location.next
75    i := i + 1
    end
77    if location /= Void then
        location.land_on (Current)
79    end
    end
81
credit (amount: INTEGER)
83    -- Credit amount to 'Current'.
    require
85    amount_positive: amount > 0
    do
87    money := money + amount
        print ("%N" + name + " credited with " + amount.out + " CHF. Total money:
            " + money.out)
89    ensure
        money_credited: money = old money + amount
91    end
93
debit (amount: INTEGER)
    -- Debit amount to 'Current'.
95    require
        amount_positive: amount > 0
97    do
        if is_money_enough (amount) then
99            money := money - amount
                print ("%N" + name + " debited with " + amount.out + " CHF. Total
                    money: " + money.out)
101        else
            money := 0

```

```

103     print ("%N" + name + " debited with " + amount.out + " CHF but did not
        own enough money. Total money: " + money.out)
        end
105     ensure
        money_debited: money = (old money - amount).max (0)
107     end

109 feature {NONE} --- Implementation

111 is_money_enough (amount: INTEGER): BOOLEAN
        -- Is 'Current's money enough for withdrawing 'amount'?
113     do
        Result := money - amount >= 0
115     end

117 invariant

119 name_exists: name /= Void and then not name.is_empty
    money_non_negative: money >= 0
121
    end
    
```

Listing 6: Class *BOARD*

```

class
2  BOARD

4 create
    make

6
feature --- Creation
8
    make
10    -- Create a board with squares.
    local
12    i: INTEGER
        square_x, square_y: SQUARE
14    do
        from
16    i := 1
        create start_square.make ("%NSquare" + i.out)
18    square_x := start_square
        print (square_x.name + " created.%N")
20    i := i + 1
        until
22    i > max_number_of_squares
        loop
24    inspect i
        when 5, 15, 25, 35 then
26    create {BAD_INVESTMENT_SQUARE} square_y.make ("Bad Investment
            Square" + i.out)
        when 10, 20, 30, 40 then
    
```

```

28     create {LOTTERY_WIN_SQUARE} square_y.make ("Lottery Win Square" + i.
        out)
    else
30     create square_y.make ("Square" + i.out)
    end
32     print (square_y.name + " created.%N")
        square_x.set_next (square_y)
34     square_x := square_y
        i := i + 1
36     end
    end
38
feature -- Access
40
    max_number_of_squares: INTEGER = 40
42     -- The max number of squares supported by the current board.
44     start_square: SQUARE
        -- The start square.
46
invariant
48     max_number_of_squares_consistent: max_number_of_squares = 40
        start_square_exists : start_square /= Void
50
end
    
```

Listing 7: Class *SQUARE*

```

1 class
    SQUARE
3
    create
5     make

7 feature {NONE} -- Initialization

9     make (n: STRING)
        -- Initialization for 'Current'.
11    require
        name_exists: n /= Void and then not n.is_empty
13    do
        name := n
15    ensure
        name_set: name = n
17    end

19 feature -- Access

21     name: STRING
        -- The square name.
23
        next: SQUARE
25     -- The next square.
    
```

```
27 feature -- Status setting
29   set_next (sq: SQUARE)
      -- Set next square.
31   require
      square_exists: sq /= Void
33   do
      next := sq
35   ensure
      next_square_set: next = sq
37   end

39 feature -- Basic operations

41   land_on (p: PLAYER)
      -- Action to be executed when player lands on 'Current'.
43   require
      player_exists: p /= Void
45   do
      print ("%N" + p.name + " landed on " + name)
47   end

49 invariant
      name_exists: name /= Void and then not name.is_empty
51 end
```

Listing 8: Class *LOTTERY\_WIN\_SQUARE*

```
class
2  LOTTERY_WIN_SQUARE

4  inherit
   SQUARE
6
   redefine land_on end
8
   create
10  make

12 feature -- Access

14  amount_to_be_credited: INTEGER = 10
      -- amount to be debited.
16

   feature -- Basic operations
18
      land_on (p: PLAYER)
20        -- Action to be executed when player lands on 'Current'.
          do
22          precursor (p)
              p.credit (amount_to_be_credited)
```

```
24  ensure then
    player_credited: p.money = old p.money + amount_to_be_credited
26  end
end
```

Listing 9: Class *BAD\_INVESTMENT\_SQUARE*

```
1  class
    BAD_INVESTMENT_SQUARE
3
    inherit
5    SQUARE

7  redefine land_on end

9  create
    make

11  feature -- Access
13
    amount_to_be_debited: INTEGER = 5
15    -- amount to be debited.

17  feature -- Basic operations

19  land_on (p: PLAYER)
    -- Action to be executed when player lands on 'Current'.
21  do
    precursor (p)
23    p.debit (amount_to_be_debited)
    ensure then
25    player_debited: p.money = (old p.money - amount_to_be_debited).max (0)
    end
27
end
```