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 TRAFFIC_GHOST

inherit TRAFFIC_FREE_MOVING

redefine
move_next
end

create make

feature -- Initialization

make (a_station: TRAFFIC_STATION; a_side: REAL)
-- Initialize positions to start.

require
a_station_exists : a_station /= Void
a_side_positive : a_side > 0.0

local
l: DS_ARRAYED_LIST [TRAFFIC_POINT]
p: TRAFFIC_POINT
x, y: REAL
do
create l.make (5)
x := a_station.location.x
y := a_station.location.y
create p.make (x - a_side/2, y - a_side/2)
l.put_last (p)
create p.make (x + a_side/2, y - a_side/2)
l.put_last (p)
create p.make (x + a_side/2, y + a_side/2)
l.put_last (p)
create p.make (x - a_side/2, y + a_side/2)
l.put_last (p)
create p.make (x - a_side/2, y - a_side/2)
l.put_last (p)
make_with_points (l, 10.0)

set_reiterate (True)
ensure
reiterating : is_reiterating
end

feature {NONE} -- Implementation

move_next
-- Move to following position
do
-- Set the locations to the corresponding ones of the line segment.
origin := poly_cursor.item
location := poly_cursor.item
if is_reiterating then
    poly_cursor.forth
    if poly_cursor.after then
        poly_cursor.start
        move_next
    else
        destination := poly_cursor.item
    end
else
    poly_cursor.forth
    if poly_cursor.after then
        has_finished := True
    else
        destination := poly_cursor.item
    end
end
end

Listing 2: Class GHOST_INVASION

class GHOST_INVASION
inhert TOURISM
feature -- Explore Paris
    invade
        -- Invade Paris with 10 ghosts.
        local
            g: TRAFFIC_GHOST
            r: RANDOM
            t: TIME
            i: INTEGER
            a: ARRAY[TRAFFIC_STATION]
        do
            Paris.display
            wait
            create t.make_now
            create r.set_seed((t.fine_seconds*1000).truncated_to_integer)
            from
                i := 1
                r. start
                a := Paris.stations.to_array
            until
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

```plaintext
class GAME
create make

feature {NONE} -- Initialization
make
   -- Run application.
local
   i: INTEGER
   p: PLAYER
do
create game_board.make
create die_1.make
create die_2.make
create players.make (1, number_of_players)
from
   i := 1
until
   i > players.count
loop
create p.make ("Player" + i.out)
p. credit (player_initial_amount)
p. set_location (game_board.start_square)
players [i] := p
   i := i + 1
end
end

feature -- Basic operations
```

play
   -- Start a game.
   local i: INTEGER
   do
       print ("%N*** Simple Boardgame ***")
       from
       until has_winner
       loop
           from
           i := 1
           until
           has_winner or else i > number_of_players
           loop
               players [i].play (die_1, die_2)
               if players [i].location = Void then
                   find_winner
               from
               winners.start
           until
           winners.after
           loop
               print ("%The winner(s): " + winners.item.name + " with money: " +
                  winners.item.money.out + " CHF")
               has_winner := True
               winners.forth
           end
           i := i + 1
       end
   end
   print ("%N*** Game Over ***")
ensure
game_has_winner: has_winner
end

find_winner
   -- Determine winner.
   local i: INTEGER
   do
       from
       i := 1
       create winners.make
       winners.extend (players [i])
       i := i + 1
   invariant
       i >= 2
       i <= number_of_players + 1
until
  i > number_of_players
loop
  if players[i].money > (winners.i_th(winners.count)).money then
    winners.wipe_out
    winners.extend(players[i])
  else if players[i].money = (winners.i_th(winners.count)).money then
    winners.extend(players[i])
  end
end
i := i + 1
variant
  number_of_players − i + 1
end

feature -- Status

  game_board: BOARD
    -- The game board.

  number_of_players: INTEGER = 2
    -- For testing purposes, the number of players is set to 2.

  players: ARRAY[PLAYER]
    -- Container for players.

  player_initial_amount: INTEGER = 7
    -- Players initial amount.

  die_1: DIE
    -- The first die.

  die_2: DIE
    -- The second die.

  has_winner: BOOLEAN
    -- Does the game have a winner?

  winners: LINKED_LIST[PLAYER]
    -- The winner(s) of the game.

invariant

  game_board_exists: game_board /= Void

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

  number_of_players_consistent: number_of_players >= 2 and number_of_players <= 6

  dice_exist: die_1 /= Void and die_2 /= Void

end
Listing 4: Class DIE

class DIE
create make

feature {NONE} -- Initialization
make -- Create a die with valid initial face value.
do
  face_value := rand.item \ 6 + 1
end

feature -- Access
face_value: INTEGER

feature -- Basic operations
roll -- Roll die
do
  rand.forth
  face_value := rand.item \ 6 + 1
end

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

invariant
  six-sided_die : face_value >1 and face_value <= 6
end

Listing 5: Class PLAYER

class
PLAYER

create
make

feature -- Access

name: STRING
-- Player name.

location: SQUARE
-- 'Current' location.

money: INTEGER
-- player current amount of money.

feature {NONE} -- Initialization

make (n: STRING)
-- Create a player with name.

require

name_exists: n /= Void and then not n.is_empty

name := n

ensure

name_set: name = n

end

feature -- Status setting

set_location (loc: SQUARE)
-- Set location for 'Current'.

require

location_exists: loc /= Void

location := loc

ensure

location_set: location = loc

end

feature -- Basic operations

play (d1,d2: DIE)
-- Play a turn.

require

dice_exist: d1 /= Void and d2 /= Void

local

dice_result: INTEGER

do

d1.roll

print ("%Nd1:" + d1.face_value.out)
\begin{verbatim}
d2. roll
   print ("%Nd2:" + d2.face_value.out)
dice_result := d1.face_value + d2.face_value
move (dice_result)
ensure
   player Moved: old location /= location
end

move (n: INTEGER)
   -- Move ‘Current’ n steps forward.
require
   n_consistent: n >= 2 and n <= 12
local
   i: INTEGER
do
   from
      i := 1
   until
      location = Void or else i > n
loop
   location := location.next
   i := i + 1
end
   if location /= Void then
      location.land_on (Current)
end
end

credit (amount: INTEGER)
   -- Credit amount to ‘Current’.
require
   amount_positive: amount > 0
do
   money := money + amount
   print ("%N" + name + " credited with " + amount.out + " CHF. Total money:
   " + money.out)
ensure
   money_credited: money = old money + amount
end

debit (amount: INTEGER)
   -- Debit amount to ‘Current’.
require
   amount_positive: amount > 0
do
   if is_money_enough (amount) then
      money := money - amount
      print ("%N" + name + " debited with " + amount.out + " CHF. Total
      money: " + money.out)
   else
      money := 0
\end{verbatim}
print ("%N" + name + " delected with " + amount.out + " CHF but did not own enough money. Total money: ") + money.out)

end

ensure
money_debited: money = (old money - amount).max (0)

end

feature {NONE} —— Implementation

is_money_enough (amount: INTEGER): BOOLEAN
—- Is ‘Current’s money enough for withdrawing ‘amount’?

do
Result := money - amount >= 0
end

invariant

name_exists: name /= Void and then not name.is_empty
money_non_negative: money >= 0

end

Listing 6: Class BOARD

class
BOARD

create
make

feature —— Creation

make
—- Create a board with squares.

local
i: INTEGER
square_x, square_y: SQUARE

do
from
i := 1
create start_square, make ("%NSquare" + i.out)

square_x := start_square
print (square_x.name + " created.%N")

i := i + 1
until
i > max_number_of_squares

loop
inspect i
when 5, 15, 25, 35 then
create {BAD_INVESTMENT_SQUARE}square_y.make ("Bad Investment Square" + i.out)
when 10, 20, 30, 40 then
create \{LOTTERY_WIN_SQUARE\} square_y.make ("Lottery Win Square" + i.out)

else
  create square_y.make ("Square" + i.out)
end

print (square_y.name + " created.

square_x := square_y
i := i + 1

end
end

feature -- Access

max_number_of_squares: INTEGER = 40
  -- The max number of squares supported by the current board.

start_square: SQUARE
  -- The start square.

invariant
max_number_of_squares_consistent: max_number_of_squares = 40
  start_square_exists: start_square /= Void

end

Listing 7: Class SQUARE

class SQUARE
  create
  make

feature \{NONE\} -- Initialization

make (n: STRING)
  -- Initialization for ‘Current’.
  require
    name_exists: n /= Void and then not n.is_empty
  do
    name := n
  ensure
    name_set: name = n
  end

feature -- Access

name: STRING
  -- The square name.

next: SQUARE
  -- 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’.
22 do
24 precursor (p)
26 p.credit (amount_to_be_credited)
Listing 9: Class BAD_INVESTMENT_SQUARE

class BAD_INVESTMENT_SQUARE
inherited SQUARE

rdefine land_on end

create make

feature -- Access

amount_to_be_debited: INTEGER = 5

feature -- Basic operations

land_on (p: PLAYER)

-- Action to be executed when player lands on 'Current'.

do
precurser (p)
p.debit (amount_to_be_debited)
ensure then

player_debited: p.money = (old p.money - amount_to_be_debited).max (0)
end

end