Rush Hour Sample Design
CSCI 134

This is one way to structure the program, but certainly not the only way. For instance, in the starter files we have included constants to represent the idea of a car being vertical or horizontal, but you might instead choose to represent a car’s orientation with a boolean.

This design covers the main properties of the game, but is not a complete implementation --- there may be additional useful methods, instance variables, constants, etc. to include in the code.

Feel free to use ideas from this design in addition to your original designs while working on the code.

class Car

Instance Vars:

- static final int HORIZONTAL = 1, VERTICAL = 2  For telling us if the Car is horizontal or vertical
- VisibleImage theCar  car body
- ParkingLot theLot  ParkingLot board we’re using
- int size  the number of spaces the car takes
- int orientation  1 if horizontal, 2 if vertical
- boolean isSpecialCar  Whether this is the special car
- Position topOrLeft  Where our last finalized upper left corner is

Constructor:

public Car(Image carImage, Position topLeft, int aSize, int orient, boolean isSpecial, ParkingLot aLot, DrawingCanvas canvas)

Create a car at the given position with the given picture, with the given orientation and size. Remember the ParkingLot, orientation, and size for later. Do what’s necessary for placing a car in its parking lot spot.

Methods:

- public boolean contains(Location point)
  Returns true if our car contains the given point.
• `private boolean isVertical() OR private boolean isHorizontal()`
  Helper method that returns true if the car is vertical, false if not.
  Alternatively, could have a method to determine whether the car is horizontal.

• `public void move(double xOffset, double yOffset)`
  If the car is vertical and it can move vertically, then move it along the vertical dimension.
  Proceed analogously if our car is horizontally oriented.

• `private boolean canMoveVertical(double yOffset)`
• `private boolean canMoveHorizontal(double xOffset)`
  Helper method that returns true if the car is allowed to move vertically/horizontally. We
  check to see if the car will fall off the board, or if we move xOffset/yOffset that there isn’t
  already a car in that new spot. Does not let either of the Car’s left or right edge
  (horizontal), nor the top or bottom edge (vertical) be dragged off the ParkingLot.

• `public void pickUp()`
  Set-up for being picked up while being dragged: save our current upper left corner and
  make sure the ParkingLot doesn’t think our current location is occupied while we’re
  moved around the board.

• `public void putDown()`
  Do what’s necessary for placing a car in its final spot when dragging is complete.
  Updates our finalized Position with where the car is now, and snaps the car to the
  appropriate grid point. Set the ParkingLot squares as having a car in them.

• `public void placeIt(Position topLeft)`
  Used for when we want to directly place our car in the given Position. Updates our
  finalized Position with the given one, and sets our ParkingLot squares as occupied.
  Useful while in “undo” mode, or when setting up the initial game board.

• `public Position getTopOrLeft()`
  Returns the car’s last finalized/confirmed Position, where the user placed it (after being
  snapped to the parking grid).

• `public boolean isSpecial()`
  Returns true if this is the special car that can win the game.

• `public boolean reachedGoal()`
  Returns true if our car’s upper left point is at the goal/winning position.

• `public void removeFromCanvas()`
  Removes the car from the canvas.
class ParkingLot

Instance Vars:

- static final int COLS=6  number of columns in our grid
- static final int ROWS=6 number of rows in our grid
- static final int START_X=10 x-offset for starting to draw grid
- static final int START_Y=10 y-offset for starting to draw grid
- static final int SPACE_SIZE=30 width & height of each grid square
  the Position of the exit/winning spot
- boolean[][] theLot keeps track of un/occupied Positions/squares

Constructor:

public ParkingLot(DrawingCanvas canvas)
  Draws the parking lot grid using the values stored as constants. Initializes our array for
  tracking which Positions are occupied by a car.

Methods:

- public Location getLocation(Position aPosition)
  Converts the given Position into a canvas Location.

- public Position getPosition(Location aLocation)
  Converts the given Location into a ParkingLot Position.

- public int getSpaceSize()
  Accessor method that returns the size of a grid square.

- public void setOccupied(Position aPosition)
  Sets the current ParkingLot Position as occupied (perhaps by a Car?)

- public void setUnoccupied(Position aPosition)
  Sets the current ParkingLot Position as unoccupied.

- public boolean isOccupied(Position aPosition)
  Returns true if the current ParkingLot grid position is set as occupied.
• public boolean isOutOfBounds(Position aPosition)
  Returns true if the given aPosition is not on the grid we drew.

• public Position adjustPosition(Location aLocation, Car aCar)
  Returns the nearest Position to snap our car to. Does not move the Car. [Alternatively, could pass the 'boolean isVertical' as a parameter, instead of aCar.]

• public Position getGoal()
  Accessor method that returns the Position of our exit/goal. Useful for determining if we’ve won.

---

**class CarCollection**

Instance vars:

• static final int MAX_CARS  
  the maximum number of cars in our game

• Car[] cars  
  all our cars in the game

• int numCars  
  the number of cars we currently have in our collection

Constructor:

```java
public CarCollection()
  Initializes our array and variable necessary for tracking what’s inside of it.
```

Methods:

• public void add(Car aCar)
  If we haven’t hit our max yet, add the given car to our collection.

• public Car selectCar(Location point)
  Returns the car located at the given point, null if there isn’t a car there. Useful for determining which car a user clicked on.

• public void clear()
  Remove all cars from the canvas and our game board. Useful when a player selects a new game.

---

**class Position (provided)**

Instance vars:
• int col the column number of this Position
• int row the row number of this Position

Constructor:

public Position(int r, int c)
    Store the given row and column as this Position's row and column values.

Methods:
• public int getCol()
    Accessor method that returns the column number.

• public int getRow()
    Accessor method that returns the row number.

• public boolean equals(Position other)
    Compares row and column numbers of this Position and the given other Position to see if they are the same (i.e., true).

class Move

Instance vars:
• Car theCar the car that was moved
• Position carPos the car's Position

Constructor:

public Move(Car aCar, Position cPos)
    Stores the given values in our instance level variables.

Methods:
• public Car getCar()
    Returns the Car that was moved.

• public Position getPosition()
    Returns the car's Position.

class MoveListInterface (provided)
Methods:
  • public Move getFirst()
    Returns the first item in our list.

  • public MoveListInterface getRest()
    Returns our list, without the first item

  • public boolean isEmpty()
    Returns true if our list is empty.

---

class NonEmptyMoveList

Instance vars:
  • Move first the first item in our list
  • MoveListInterface rest the rest of the list, without the first item

Constructor:

```java
public NonEmptyMoveList(Move aMove, MoveListInterface aMoveList)
    Stores the given values into our instance level variables.
```

Methods:
  • public Move getFirst()
    Returns the first item in our list.

  • public MoveListInterface getRest()
    Returns our list, without the first item

  • public boolean isEmpty()
    Returns true if our list is empty (it isn’t).

---

class EmptyMoveList

Constructor:

```java
• public EmptyMoveList()
  Default constructor.
```

Methods:
• public Move getFirst()
  Returns the first item in our list -- but there is none, so return null.

• public MoveListInterface getRest()
  Returns an EmptyMoveList.

• public boolean isEmpty()
  Returns true if our list is empty (it is). If a list is empty, we can stop undoing Moves.

---

class MovePlayer extends ActiveObject

Instance vars:
• Static final int DELAY=500  // Pause amount for our animation
• MoveListInterface moves    // The moves we’re playing backwards

Constructor:

public MovePlayer(MoveListInterface aMoves)
  Saves our list of moves so we can step through them, then starts the playback of all
  moves in reverse (undo all).

Methods:
• public void run()
  while (we still have moves left to play) {
    Pause the stored amount. Get our first move from the list, which includes the car
    that moved and its position. Pick up the car, place it in its last finalized Position in
    the ParkingLot. Repeat until we run out of Moves in our list.
  }

---

class RushHour extends WindowController
  implements ActionListener, ItemListener

Instance vars:
• Constants for game setup --- change as you like:
  ○ String NOV= "NOVICE "    // Strings for selecting game levels
  ○ String INTE= "INTERMEDIATE "
  ○ String EXP= "EXPERT "
• GUI
• JButton undo, undoAll  
  Undo buttons
• JComboBox gameChoice  
  Game difficulty selection
• Text winMessage, winInstructions  
  Winning messages

• Movement
  • Car selectedCar  
  The car that the user is moving
  • Location lastPoint  
  Last location of car (for dragging)
  • boolean carSelected  
  If the user selected a car to move
  • MoveListInterface moves  
  Running list of all moves user has made
  • Position startingPos  
  Original Position of the car at selection time
    (useful for identifying a new Move)

  • ParkingLot theLot  
  The grid for gameplay
  • CarCollection allCars  
  All the cars in play

Methods:
  • public void begin()  
    Create the Graphical User Interface elements and the objects stored in the instance variables. Also create the Text shown after winning.
  • public void onMousePress(Location point)  
    If user pressed on a car, then store the selected car and location for dragging. Have selectedCar do other additional set-up for being picked up.
  • public void onMouseDrag(Location point)  
    If user pressed on a car, then tell car to move as it’s dragged. Update lastPoint.
  • public void onMouseRelease(Location point)  
    If user pressed on a car, then have selectedCar do what’s necessary for placing a car in its final spot (i.e., snap to grid Position, etc.). Add the new Move to our list of moves. Check to see if the user won, and show the message if appropriate.
  • public void actionPerformed(ActionEvent evt)  
    Undo one move if the Undo button was selected, updating our Moves list. Have a MovePlayer play every move if the Undo All button was selected.
  • public void itemStateChanged(ItemEvent evt)  
    Set-up a new game board according to which option the user selected.
  • private void setNewGame(String filename)  
    Helper method clears old data and reads in the game set-up data from the provided filename.