/\*

\* AP(r) Computer Science GridWorld Case Study:

\* Copyright(c) 2002-2006 College Entrance Examination Board

\* (http://www.collegeboard.com).

\*

\* This code is free software; you can redistribute it and/or modify

\* it under the terms of the GNU General Public License as published by

\* the Free Software Foundation. \*

\* This code is distributed in the hope that it will be useful,

\* but WITHOUT ANY WARRANTY; without even the implied warranty of

\* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

\* GNU General Public License for more details.

\*

\* @author Alyce Brady

\* @author Chris Nevison

\* @author APCS Development Committee

\* @author Cay Horstmann

\*/

package info.gridworld.grid;

/\*\*

\* A <code>Location</code> object represents the row and column of a location

\* in a two-dimensional grid. <br />

\* The API of this class is testable on the AP CS A and AB exams.

\*/

public class Location implements Comparable

{

private int row; // row location in grid

private int col; // column location in grid

/\*\*

\* The turn angle for turning 90 degrees to the left.

\*/

public static final int LEFT = -90;

/\*\*

\* The turn angle for turning 90 degrees to the right.

\*/

public static final int RIGHT = 90;

/\*\*

\* The turn angle for turning 45 degrees to the left.

\*/

public static final int HALF\_LEFT = -45;

/\*\*

\* The turn angle for turning 45 degrees to the right.

\*/

public static final int HALF\_RIGHT = 45;

/\*\*

\* The turn angle for turning a full circle.

\*/

public static final int FULL\_CIRCLE = 360;

/\*\*

\* The turn angle for turning a half circle.

\*/

public static final int HALF\_CIRCLE = 180;

/\*\*

\* The turn angle for making no turn.

\*/

public static final int AHEAD = 0;

/\*\*

\* The compass direction for north.

\*/

public static final int NORTH = 0;

/\*\*

\* The compass direction for northeast.

\*/

public static final int NORTHEAST = 45;

/\*\*

\* The compass direction for east.

\*/

public static final int EAST = 90;

/\*\*

\* The compass direction for southeast.

\*/

public static final int SOUTHEAST = 135;

/\*\*

\* The compass direction for south.

\*/

public static final int SOUTH = 180;

/\*\*

\* The compass direction for southwest.

\*/

public static final int SOUTHWEST = 225;

/\*\*

\* The compass direction for west.

\*/

public static final int WEST = 270;

/\*\*

\* The compass direction for northwest.

\*/

public static final int NORTHWEST = 315;

/\*\*

\* Constructs a location with given row and column coordinates.

\* @param r the row

\* @param c the column

\*/

public Location(int r, int c)

{

row = r;

col = c;

}

/\*\*

\* Gets the row coordinate.

\* @return the row of this location

\*/

public int getRow()

{

return row;

}

/\*\*

\* Gets the column coordinate.

\* @return the column of this location

\*/

public int getCol()

{

return col;

}

/\*\*

\* Gets the adjacent location in any one of the eight compass directions.

\* @param direction the direction in which to find a neighbor location

\* @return the adjacent location in the direction that is closest to

\* <tt>direction</tt>

\*/

public Location getAdjacentLocation(int direction)

{

// reduce mod 360 and round to closest multiple of 45

int adjustedDirection = (direction + HALF\_RIGHT / 2) % FULL\_CIRCLE;

if (adjustedDirection < 0)

adjustedDirection += FULL\_CIRCLE;

adjustedDirection = (adjustedDirection / HALF\_RIGHT) \* HALF\_RIGHT;

int dc = 0;

int dr = 0;

if (adjustedDirection == EAST)

dc = 1;

else if (adjustedDirection == SOUTHEAST)

{

dc = 1;

dr = 1;

}

else if (adjustedDirection == SOUTH)

dr = 1;

else if (adjustedDirection == SOUTHWEST)

{

dc = -1;

dr = 1;

}

else if (adjustedDirection == WEST)

dc = -1;

else if (adjustedDirection == NORTHWEST)

{

dc = -1;

dr = -1;

}

else if (adjustedDirection == NORTH)

dr = -1;

else if (adjustedDirection == NORTHEAST)

{

dc = 1;

dr = -1;

}

return new Location(getRow() + dr, getCol() + dc);

}

/\*\*

\* Returns the direction from this location toward another location. The

\* direction is rounded to the nearest compass direction.

\* @param target a location that is different from this location

\* @return the closest compass direction from this location toward

\* <code>target</code>

\*/

public int getDirectionToward(Location target)

{

int dx = target.getCol() - getCol();

int dy = target.getRow() - getRow();

// y axis points opposite to mathematical orientation

int angle = (int) Math.toDegrees(Math.atan2(-dy, dx));

// mathematical angle is counterclockwise from x-axis,

// compass angle is clockwise from y-axis

int compassAngle = RIGHT - angle;

// prepare for truncating division by 45 degrees

compassAngle += HALF\_RIGHT / 2;

// wrap negative angles

if (compassAngle < 0)

compassAngle += FULL\_CIRCLE;

// round to nearest multiple of 45

return (compassAngle / HALF\_RIGHT) \* HALF\_RIGHT;

}

/\*\*

\* Indicates whether some other <code>Location</code> object is "equal to"

\* this one.

\* @param other the other location to test

\* @return <code>true</code> if <code>other</code> is a

\* <code>Location</code> with the same row and column as this location;

\* <code>false</code> otherwise

\*/

public boolean equals(Object other)

{

if (!(other instanceof Location))

return false;

Location otherLoc = (Location) other;

return getRow() == otherLoc.getRow() && getCol() == otherLoc.getCol();

}

/\*\*

\* Generates a hash code.

\* @return a hash code for this location

\*/

public int hashCode()

{

return getRow() \* 3737 + getCol();

}

/\*\*

\* Compares this location to <code>other</code> for ordering. Returns a

\* negative integer, zero, or a positive integer as this location is less

\* than, equal to, or greater than <code>other</code>. Locations are

\* ordered in row-major order. <br />

\* (Precondition: <code>other</code> is a <code>Location</code> object.)

\* @param other the other location to test

\* @return a negative integer if this location is less than

\* <code>other</code>, zero if the two locations are equal, or a positive

\* integer if this location is greater than <code>other</code>

\*/

public int compareTo(Object other)

{

Location otherLoc = (Location) other;

if (getRow() < otherLoc.getRow())

return -1;

if (getRow() > otherLoc.getRow())

return 1;

if (getCol() < otherLoc.getCol())

return -1;

if (getCol() > otherLoc.getCol())

return 1;

return 0;

}

/\*\*

\* Creates a string that describes this location.

\* @return a string with the row and column of this location, in the format

\* (row, col)

\*/

public String toString()

{

return "(" + getRow() + ", " + getCol() + ")";

}

}