|  | /\* |
| --- | --- |
|  | \* 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 APCS Development Committee |
|  | \* @author Cay Horstmann |
|  | \*/ |
|  |  |
|  | package info.gridworld.grid; |
|  |  |
|  | import java.util.ArrayList; |
|  |  |
|  | /\*\* |
|  | \* A <code>BoundedGrid</code> is a rectangular grid with a finite number of |
|  | \* rows and columns. <br /> |
|  | \* The implementation of this class is testable on the AP CS AB exam. |
|  | \*/ |
|  | public class BoundedGrid<E> extends AbstractGrid<E> |
|  | { |
|  | private Object[][] occupantArray; // the array storing the grid elements |
|  |  |
|  | /\*\* |
|  | \* Constructs an empty bounded grid with the given dimensions. |
|  | \* (Precondition: <code>rows > 0</code> and <code>cols > 0</code>.) |
|  | \* @param rows number of rows in BoundedGrid |
|  | \* @param cols number of columns in BoundedGrid |
|  | \*/ |
|  | public BoundedGrid(int rows, int cols) |
|  | { |
|  | if (rows <= 0) |
|  | throw new IllegalArgumentException("rows <= 0"); |
|  | if (cols <= 0) |
|  | throw new IllegalArgumentException("cols <= 0"); |
|  | occupantArray = new Object[rows][cols]; |
|  | } |
|  |  |
|  | public int getNumRows() |
|  | { |
|  | return occupantArray.length; |
|  | } |
|  |  |
|  | public int getNumCols() |
|  | { |
|  | // Note: according to the constructor precondition, numRows() > 0, so |
|  | // theGrid[0] is non-null. |
|  | return occupantArray[0].length; |
|  | } |
|  |  |
|  | public boolean isValid(Location loc) |
|  | { |
|  | return 0 <= loc.getRow() && loc.getRow() < getNumRows() |
|  | && 0 <= loc.getCol() && loc.getCol() < getNumCols(); |
|  | } |
|  |  |
|  | public ArrayList<Location> getOccupiedLocations() |
|  | { |
|  | ArrayList<Location> theLocations = new ArrayList<Location>(); |
|  |  |
|  | // Look at all grid locations. |
|  | for (int r = 0; r < getNumRows(); r++) |
|  | { |
|  | for (int c = 0; c < getNumCols(); c++) |
|  | { |
|  | // If there's an object at this location, put it in the array. |
|  | Location loc = new Location(r, c); |
|  | if (get(loc) != null) |
|  | theLocations.add(loc); |
|  | } |
|  | } |
|  |  |
|  | return theLocations; |
|  | } |
|  |  |
|  | public E get(Location loc) |
|  | { |
|  | if (!isValid(loc)) |
|  | throw new IllegalArgumentException("Location " + loc |
|  | + " is not valid"); |
|  | return (E) occupantArray[loc.getRow()][loc.getCol()]; // unavoidable warning |
|  | } |
|  |  |
|  | public E put(Location loc, E obj) |
|  | { |
|  | if (!isValid(loc)) |
|  | throw new IllegalArgumentException("Location " + loc |
|  | + " is not valid"); |
|  | if (obj == null) |
|  | throw new NullPointerException("obj == null"); |
|  |  |
|  | // Add the object to the grid. |
|  | E oldOccupant = get(loc); |
|  | occupantArray[loc.getRow()][loc.getCol()] = obj; |
|  | return oldOccupant; |
|  | } |
|  |  |
|  | public E remove(Location loc) |
|  | { |
|  | if (!isValid(loc)) |
|  | throw new IllegalArgumentException("Location " + loc |
|  | + " is not valid"); |
|  |  |
|  | // Remove the object from the grid. |
|  | E r = get(loc); |
|  | occupantArray[loc.getRow()][loc.getCol()] = null; |
|  | return r; |
|  | } |
|  | } |