```java
/*
 * AP(r) Computer Science GridWorld Case Study:
 * Copyright(c) 2005-2006 Cay S. Horstmann (http://horstmann.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 Cay Horstmann
 */

package info.gridworld.actor;

import info.gridworld.grid.Grid;

/**
 * An <code>Actor</code> is an entity with a color and direction that can act.
 * <br />
 * The API of this class is testable on the AP CS A and AB exams.
 */
public class Actor
{
    private Grid<Actor> grid;
    private Location location;
    private int direction;
    private Color color;

    /**
     * Constructs a blue actor that is facing north.
     */
    public Actor()
    {
        color = Color.BLUE;
        direction = Location.NORTH;
        grid = null;
        location = null;
    }

    /**
     * Gets the color of this actor.
     * @return the color of this actor
     */
    public Color getColor()
    {
        return color;
    }

    /**
     * Sets the color of this actor.
     * @param newColor the new color
     */
    public void setColor(Color newColor)
    {
        color = newColor;
    }

    /**
     * Gets the current direction of this actor.
     * @return the direction of this actor, an angle between 0 and 359 degrees
     */
    public int getDirection()
    {
        return direction;
    }

    /**
     * Sets the current direction of this actor.
     * @param newDirection the new direction. The direction of this actor is set
```

```
77      * to the angle between 0 and 359 degrees that is equivalent to
78      * <code>newDirection</code>.
79      */
80     public void setDirection(int newDirection)
81     {
82         direction = newDirection % Location.FULL_CIRCLE;
83         if (direction < 0)
84             direction += Location.FULL_CIRCLE;
85     }
86
87     /**
88      * Gets the grid in which this actor is located.
89      * @return the grid of this actor, or <code>null</code> if this actor is
90      * not contained in a grid
91      */
92     public Grid<Actor> getGrid()
93     {
94         return grid;
95     }
96
97     /**
98      * Gets the location of this actor.
99      * @return the location of this actor, or <code>null</code> if this actor is
100     * not contained in a grid
101     */
102    public Location getLocation()
103    {
104        return location;
105    }
106
107    /**
108     * Puts this actor into a grid. If there is another actor at the given
109     * location, it is removed. <br />
110     * Precondition: (1) This actor is not contained in a grid (2)
111     * <code>loc</code> is valid in <code>gr</code>
112     * @param gr the grid into which this actor should be placed
113     * @param loc the location into which the actor should be placed
114     */
115    public void putSelfInGrid(Grid<Actor> gr, Location loc)
116    {
117        if (grid != null)
118            throw new IllegalStateException(
119                    "This actor is already contained in a grid.");
120
121        Actor actor = gr.get(loc);
122        if (actor != null)
123            actor.removeSelfFromGrid();
124        gr.put(loc, this);
125        grid = gr;
126        location = loc;
127    }
128
129    /**
130     * Removes this actor from its grid. <br />
131     * Precondition: This actor is contained in a grid
132     */
133    public void removeSelfFromGrid()
134    {
135        if (grid == null)
136            throw new IllegalStateException(
137                    "This actor is not contained in a grid.");
138        if (grid.get(location) != this)
139            throw new IllegalStateException(
140                    "The grid contains a different actor at location "
141                            + location + ".");
142
143        grid.remove(location);
144        grid = null;
145        location = null;
146    }
147
148    /**
149     * Moves this actor to a new location. If there is another actor at the
```

```
150      * given location, it is removed. <br />
151      * Precondition: (1) This actor is contained in a grid (2)
152      * <code>newLocation</code> is valid in the grid of this actor
153      * @param newLocation the new location
154      */
155     public void moveTo(Location newLocation)
156     {
157         if (grid == null)
158             throw new IllegalStateException("This actor is not in a grid.");
159         if (grid.get(location) != this)
160             throw new IllegalStateException(
161                     "The grid contains a different actor at location "
162                             + location + ".");
163         if (!grid.isValid(newLocation))
164             throw new IllegalArgumentException("Location " + newLocation
165                     + " is not valid.");
166 
167         if (newLocation.equals(location))
168             return;
169         grid.remove(location);
170         Actor other = grid.get(newLocation);
171         if (other != null)
172             other.removeSelfFromGrid();
173         location = newLocation;
174         grid.put(location, this);
175     }
176 
177     /**
178      * Reverses the direction of this actor. Override this method in subclasses
179      * of <code>Actor</code> to define types of actors with different behavior
180      *
181      */
182     public void act()
183     {
184         setDirection(getDirection() + Location.HALF_CIRCLE);
185     }
186 
187     /**
188      * Creates a string that describes this actor.
189      * @return a string with the location, direction, and color of this actor
190      */
191     public String toString()
192     {
193         return getClass().getName() + "[location=" + location + ",direction="
194                 + direction + ",color=" + color + "]";
195     }
196 }
```