Problem 2: StandardRobot Class

Each robot must also have some code that tells it how to move about a room, which will go in a method called `updatePositionAndClean`.

Ordinarily we would consider putting all the robot's methods in a single class. However, later in this problem set we'll consider robots with alternate movement strategies, to be implemented as different classes with the same interface. These classes will have a different implementation of `updatePositionAndClean` but are for the most part the same as the original robots. Therefore, we'd like to use inheritance to reduce the amount of duplicated code.

We have already refactored the robot code for you into two classes: the `Robot` class you completed in Problem 1 (which contains general robot code), and a `StandardRobot` class that inherits from it (which contains its own movement strategy).

Complete the `updatePositionAndClean` method of `StandardRobot` to simulate the motion of the robot after a single time-step (as described on the Simulation Overview page).

```
class StandardRobot(Robot):
    """
    A StandardRobot is a Robot with the standard movement strategy.

    At each time-step, a StandardRobot attempts to move in its current direction;
when
    it hits a wall, it chooses a new direction randomly.
    """
    def updatePositionAndClean(self):
        """
        Simulate the passage of a single time-step.

        Move the robot to a new position and mark the tile it is on as having
        been cleaned.
        """
```

We have provided the `getNewPosition` method of `Position`, which you may find helpful:

```
class Position(object):

    def getNewPosition(self, angle, speed):
        """
        Computes and returns the new Position after a single clock-tick has
        passed, with this object as the current position, and with the
        specified angle and speed.

        Does NOT test whether the returned position fits inside the room.

        angle: number representing angle in degrees, 0 <= angle < 360
        speed: positive float representing speed

        Returns: a Position object representing the new position.
```

```
        """
```

Note: You can pass in an integer or a float for the `angle` parameter.

Before moving on to Problem 3, check that your implementation of Standard Robot works by uncommenting the following line under your implementation of StandardRobot. Make sure that as your robot moves around the room, the tiles it traverses switch colors from gray to white. It should take about a minute for it to clean all the tiles.

```
testRobotMovement(StandardRobot, RectangularRoom)
```

Enter your code for classes `Robot` (from the previous problem) and `StandardRobot` below.

Test: 1 class creation

```
        robot = StandardRobot(RectangularRoom(1,2), 1.0)
```

Output:

Test: 2 test setRobotPosition

```
        robot = StandardRobot(RectangularRoom(5,8), 1.0)
        robot.getRobotPosition()
        loop 10 times:
            * Generate random x, y values
            * Check if Position(x,y) is in the room
                * If so, robot.setRobotPosition(Position(x, y))
                * robot.getRobotPosition()
```

Output:

```
    Random position 0: (6.00, 0.00)
    Random position 1: (1.00, 2.00)
       In room; setting position. Position is now: (1.00, 2.00)
    Random position 2: (0.00, 2.00)
       In room; setting position. Position is now: (0.00, 2.00)
    Random position 3: (4.00, 1.00)
       In room; setting position. Position is now: (4.00, 1.00)
    Random position 4: (5.00, 9.00)
    Random position 5: (1.00, 6.00)
       In room; setting position. Position is now: (1.00, 6.00)
    Random position 6: (5.00, 4.00)
    Random position 7: (4.00, 0.00)
       In room; setting position. Position is now: (4.00, 0.00)
    Random position 8: (0.00, 2.00)
       In room; setting position. Position is now: (0.00, 2.00)
    Random position 9: (3.00, 0.00)
       In room; setting position. Position is now: (3.00, 0.00)
```

Test: 3 test setRobotDirection

```
        robot = StandardRobot(RectangularRoom(5,8), 1.0)
```

```
    robot.getRobotDirection()
    loop 10 times:
        * Generate random direction value
        * robot.setRobotDirection(randDirection)
        * robot.getRobotDirection()
```

Output:

```
Random direction: 107
  Setting direction. Direction is now: 107
Random direction: 202
  Setting direction. Direction is now: 202
Random direction: 118
  Setting direction. Direction is now: 118
Random direction: 143
  Setting direction. Direction is now: 143
Random direction: 14
  Setting direction. Direction is now: 14
Random direction: 99
  Setting direction. Direction is now: 99
Random direction: 311
  Setting direction. Direction is now: 311
Random direction: 129
  Setting direction. Direction is now: 129
Random direction: 63
  Setting direction. Direction is now: 63
Random direction: 37
  Setting direction. Direction is now: 37
```

Test: 4 test updatePositionAndClean

```
    Test StandardRobot.updatePositionAndClean()
```

Output:

```
Creating room and robot...
Setting position and direction to Position(1.5, 2.5) and 90...
Calling updatePositionAndClean(); robot speed is 1.0
Passed; now calling updatePositionAndClean() 20 times
Passed test.
```

Test: 5 test updatePositionAndClean

```
    Test StandardRobot.updatePositionAndClean()
```

Output:

```
Creating randomly sized room: 10x7 - and robot at speed 0.68...
Robot initalized at random position
Was initial position cleaned? True
Robot initalized at random direction:
```

```
Number of cleaned tiles: 1

Calling updatePositionAndClean() 30 times...
Cleaned the minimum number of tiles; test passed.
```