1. (4 points) Square.toString()
   Here’s what I would do for that. A switch statement is fine too:

   ```java
   public String toString() {
       if (type == EMPTY)  
           return "_";
       else if (type == WALL)
           return "#";
       else if (type == START)
           return "S";
       else if (type == EXIT)
           return "E";
   }
   ```

2. (4 points) Maze.toString()
   ```java
   public String toString() {
       String S = new String();
       for (int i = 0; i < Rows; i++) {
           for (int j = 0; j < Cols; j++)
               S = S+maze[i][j].toString();
           S = S+"\n";
       }
       return S;
   }
   ```

3. operations

<table>
<thead>
<tr>
<th>Queue</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>addL(1)</td>
<td>1</td>
</tr>
<tr>
<td>add(2)</td>
<td>1 2</td>
</tr>
<tr>
<td>remove()</td>
<td>2</td>
</tr>
<tr>
<td>add(3)</td>
<td>2 3</td>
</tr>
<tr>
<td>add(4)</td>
<td>2 3 4</td>
</tr>
<tr>
<td>remove()</td>
<td>3 4</td>
</tr>
<tr>
<td>remove()</td>
<td>4</td>
</tr>
<tr>
<td>add(5)</td>
<td>4 5</td>
</tr>
<tr>
<td>Stack</td>
<td>( \text{add}(1) )</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

4. Maze algorithm

<table>
<thead>
<tr>
<th>worklist as a stack</th>
<th>at the start</th>
<th>after step 1</th>
<th>after step 2</th>
<th>after step 3</th>
<th>after step 4</th>
<th>after step 5</th>
<th>after step 6</th>
<th>after step 7</th>
<th>after step 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>(6,4)</td>
<td>(6,3)</td>
<td>(6,2)</td>
<td>(6,1)</td>
<td>(6,0)</td>
<td>(5,0)</td>
<td>(4,0)</td>
<td>(4,1)</td>
<td>(3,0)</td>
<td>(4,2)</td>
</tr>
</tbody>
</table>

| newly explored square | N/A | (6,4) | (6,3) | (6,2) | (6,1) | (6,0) | (5,0) | (4,0) | (4,1) |

| newly marked square(s) | N/A | (6,3) | (6,5) | (6,2) | (6,1) | (6,0) | (5,0) | (4,0) | (4,1) |

5. Printing the solution
   make a stack of Squares
   set \( p = \text{exit square} \)
   while (\( p != \text{start square} \)) {
       push \( p \) onto the stack
       \( p = p.\text{previous} \)
   }
push the start square onto the stack
while (stack is not empty) {
    print the stack top row and column
    pop the stack
}