

```

1  package Pieces;
2
3  import Game.Board;
4  import Game.Colour;
5  import Game.Player;
6
7  /**
8   * This class represents the King piece
9   * @author E
10  */
11  public class King extends Piece {
12
13      public boolean canMove;
14      private boolean hasMoved;
15
16      public King(Colour colour) {
17          super(PieceType.King, colour, Integer.MAX_VALUE);
18          hasMoved = false;
19      }
20
21      @Override
22      public int threats(Board board, int row, int column) {
23          Piece[][] currentBoard = board.getBoard();
24          Piece toExamine;
25          int threatened = 0;
26          if (row >= 1) {
27              // top
28              toExamine = currentBoard[row - 1][column];
29              if (toExamine != null) {
30                  if (this.isOppositeColour(toExamine)) {
31                      threatened += toExamine.weight;
32                  }
33              }
34              if (column >= 1) {
35                  // top-left
36                  toExamine = currentBoard[row - 1][column - 1];
37                  if (toExamine != null) {
38                      if (this.isOppositeColour(toExamine)) {
39                          threatened += toExamine.weight;
40                      }
41                  }
42              }
43              if (column <= 6) {
44                  // top-right
45                  toExamine = currentBoard[row - 1][column + 1];
46                  if (toExamine != null) {
47                      if (this.isOppositeColour(toExamine)) {
48                          threatened += toExamine.weight;
49                      }
50                  }
51              }
52          }
53          if (row <= 6) {
54              // bottom
55              toExamine = currentBoard[row + 1][column];
56              if (toExamine != null) {
57                  if (this.isOppositeColour(toExamine)) {
58                      threatened += toExamine.weight;
59                  }
60              }
61              if (column >= 1) {
62                  // bottom-left
63                  toExamine = currentBoard[row + 1][column - 1];
64                  if (toExamine != null) {
65                      if (this.isOppositeColour(toExamine)) {
66                          threatened += toExamine.weight;
67                      }
68                  }
69              }

```

```

70         if (column <= 6) {
71             // bottom-right
72             toExamine = currentBoard[row + 1][column + 1];
73             if (toExamine != null) {
74                 if (this.isOppositeColour(toExamine)) {
75                     threatened += toExamine.weight;
76                 }
77             }
78         }
79     }
80     if (column >= 1) {
81         // left
82         toExamine = currentBoard[row][column - 1];
83         if (toExamine != null) {
84             if (this.isOppositeColour(toExamine)) {
85                 threatened += toExamine.weight;
86             }
87         }
88     }
89     if (column <= 6) {
90         // right
91         toExamine = currentBoard[row][column + 1];
92         if (toExamine != null) {
93             if (this.isOppositeColour(toExamine)) {
94                 threatened += toExamine.weight;
95             }
96         }
97     }
98     return threatened;
99 }
100
101 @Override
102 public int[][] attacks(Board board, int row, int column) {
103     int[][] attacked = new int[8][8];
104     int nextX, nextY;
105     if (row >= 1) {
106         // top
107         nextX = row - 1;
108         nextY = column;
109         attacked[nextX][nextY]++;
110         if (column >= 1) {
111             // top-left
112             nextX = row - 1;
113             nextY = column - 1;
114             attacked[nextX][nextY]++;
115         }
116         if (column <= 6) {
117             // top-right
118             nextX = row - 1;
119             nextY = column + 1;
120             attacked[nextX][nextY]++;
121         }
122     }
123     if (row <= 6) {
124         // bottom
125         nextX = row + 1;
126         nextY = column;
127         attacked[nextX][nextY]++;
128         if (column >= 1) {
129             // bottom-left
130             nextX = row + 1;
131             nextY = column - 1;
132             attacked[nextX][nextY]++;
133         }
134         if (column <= 6) {
135             // bottom-right
136             nextX = row + 1;
137             nextY = column + 1;
138             attacked[nextX][nextY]++;

```

```

139         }
140     }
141     if (column >= 1) {
142         // right
143         nextX = row;
144         nextY = column + 1;
145         attacked[nextX][nextY]++;
146     }
147     if (column <= 6) {
148         // left
149         nextX = row;
150         nextY = column - 1;
151         attacked[nextX][nextY]++;
152     }
153     return attacked;
154 }
155
156 @Override
157 public boolean[][] validMoves(Player opponent, Board board, int row, int column) {
158     Piece[][] currentBoard = board.getBoard();
159     Piece toExamine;
160     // reset to false and check
161     canMove = false;
162     boolean[][] validPositions = new boolean[8][8];
163     int nextX, nextY;
164     if (row >= 1) {
165         // top
166         nextX = row - 1;
167         nextY = column;
168         toExamine = currentBoard[nextX][nextY];
169         if ((toExamine == null | toExamine != null &&
170             this.isOppositeColour(toExamine))
171             && this.isThreatened(board, nextX, nextY) == 0) {
172             validPositions[nextX][nextY] = true;
173             canMove = true;
174         }
175         if (column >= 1) {
176             // top-left
177             nextX = row - 1;
178             nextY = column - 1;
179             toExamine = currentBoard[nextX][nextY];
180             if ((toExamine == null | toExamine != null &&
181                 this.isOppositeColour(toExamine))
182                 && this.isThreatened(board, nextX, nextY) == 0) {
183                 validPositions[nextX][nextY] = true;
184                 canMove = true;
185             }
186         }
187         if (column <= 6) {
188             // top-right
189             nextX = row - 1;
190             nextY = column + 1;
191             toExamine = currentBoard[nextX][nextY];
192             if ((toExamine == null | toExamine != null &&
193                 this.isOppositeColour(toExamine))
194                 && this.isThreatened(board, nextX, nextY) == 0) {
195                 validPositions[nextX][nextY] = true;
196                 canMove = true;
197             }
198         }
199     }
200     if (row <= 6) {
201         // bottom
202         nextX = row + 1;
203         nextY = column;
204         toExamine = currentBoard[nextX][nextY];
205         if ((toExamine == null | toExamine != null &&
206             this.isOppositeColour(toExamine))
207             && this.isThreatened(board, nextX, nextY) == 0) {

```

```

204         validPositions[nextX][nextY] = true;
205         canMove = true;
206     }
207     if (column >= 1) {
208         // bottom-left
209         nextX = row + 1;
210         nextY = column - 1;
211         toExamine = currentBoard[nextX][nextY];
212         if ((toExamine == null | toExamine != null &&
213             this.isOppositeColour(toExamine))
214             && this.isThreatened(board, nextX, nextY) == 0) {
215             validPositions[nextX][nextY] = true;
216             canMove = true;
217         }
218     }
219     if (column <= 6) {
220         // bottom-right
221         nextX = row + 1;
222         nextY = column + 1;
223         toExamine = currentBoard[nextX][nextY];
224         if ((toExamine == null | toExamine != null &&
225             this.isOppositeColour(toExamine))
226             && this.isThreatened(board, nextX, nextY) == 0) {
227             validPositions[nextX][nextY] = true;
228             canMove = true;
229         }
230     }
231     if (row >= 1) {
232         // right
233         nextX = row;
234         nextY = column + 1;
235         toExamine = currentBoard[nextX][nextY];
236         if ((toExamine == null | toExamine != null &&
237             this.isOppositeColour(toExamine))
238             && this.isThreatened(board, nextX, nextY) == 0) {
239             validPositions[nextX][nextY] = true;
240             canMove = true;
241         }
242     }
243     if (column <= 6) {
244         // left
245         nextX = row;
246         nextY = column - 1;
247         toExamine = currentBoard[nextX][nextY];
248         if ((toExamine == null | toExamine != null &&
249             this.isOppositeColour(toExamine))
250             && this.isThreatened(board, nextX, nextY) == 0) {
251             validPositions[nextX][nextY] = true;
252             canMove = true;
253         }
254     }
255     }
256     // castle check
257     int pos = colour == Colour.White ? 7 : 0;
258     int[][] oppAttacks = opponent.getAttacks();
259     if (column == 4 && !hasMoved && this.isThreatened(board, pos, 4) == 0) {
260         // examine rooks
261         // check queen side
262         toExamine = currentBoard[pos][0];
263         if (toExamine != null && toExamine.validSpecial()) {
264             // check queen side
265             if (oppAttacks[pos][column - 1] == 0
266                 && oppAttacks[pos][column - 2] == 0) {
267                 validPositions[pos][2] = true;
268                 canMove = true;
269             }
270         }
271     }
272     toExamine = currentBoard[pos][7];
273     if (toExamine != null && toExamine.validSpecial()) {

```

```

269         // check king side
270         if (oppAttacks[pos][column + 1] == 0
271             && oppAttacks[pos][column + 2] == 0) {
272             validPositions[pos][6] = true;
273             canMove = true;
274         }
275     }
276 }
277 return validPositions;
278 }
279
280 @Override
281 public boolean validSpecial() {
282     return !hasMoved; // if it hasn't moved it can castle
283 }
284
285 @Override
286 public void modifySpecial() {
287     hasMoved = true;
288 }
289
290 @Override
291 public String printToBoard() {
292     return this.colour == Colour.White ? "\u2654" : "\u265A";
293 }
294
295 public String printToLog() {
296     return "K";
297 }
298
299 public boolean castle(Rook rook) {
300     return false; // need to check if rook has moved
301 }
302
303 @Override
304 public boolean getCanMove() {
305     return canMove;
306 }
307 }
308

```