Refactoring
(Tetris #4 example)

Make the first test in FallingPiecesTest to compile.

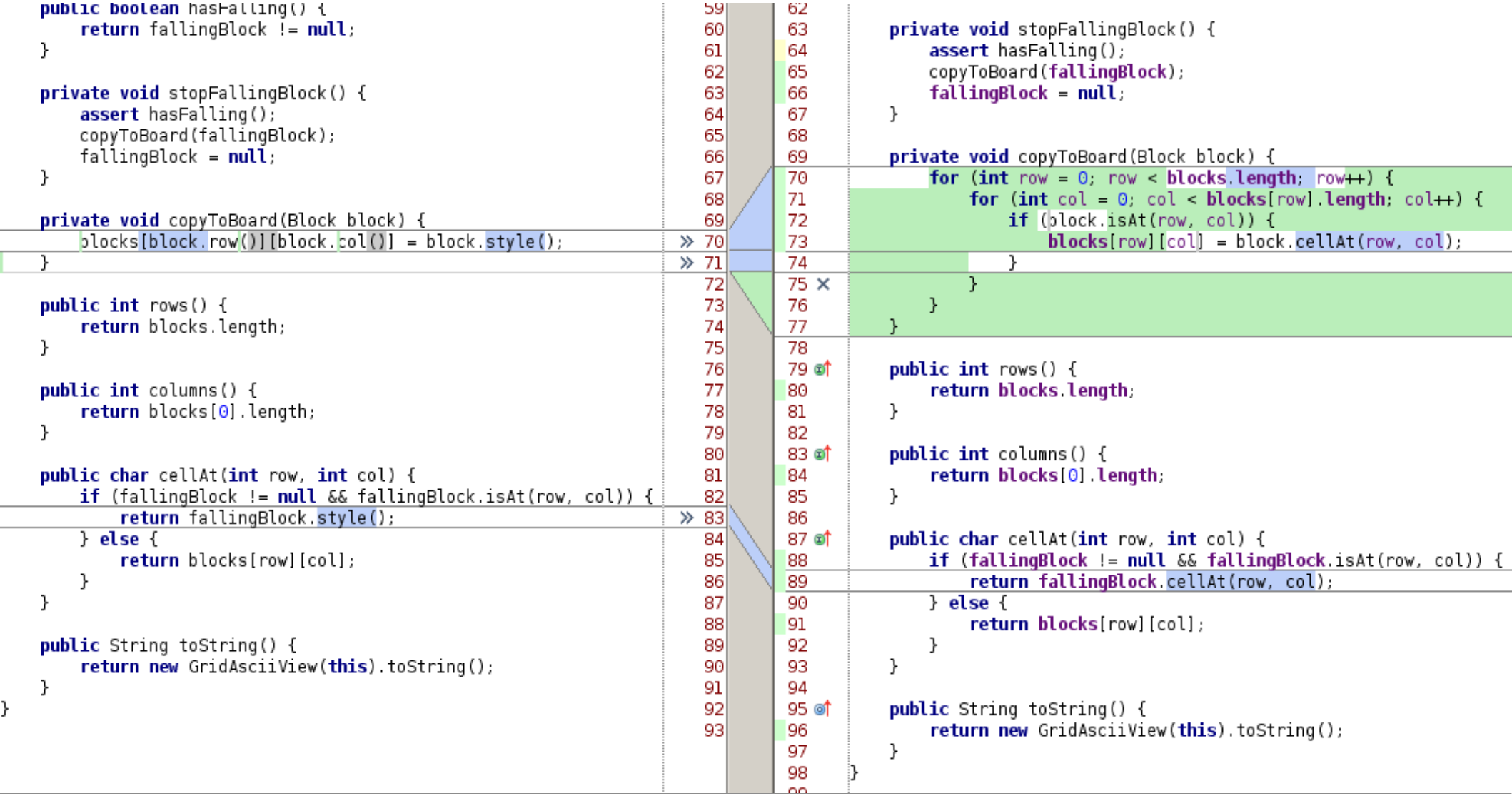
```
34          }
35      }
36
37      private boolean conflictsWithBoard(Block block) {
38          return outsideBoard(block) || hitsAnotherBlock(block);
39      }
40
41      private boolean outsideBoard(Block block) {
42          return block.row() >= rows();
43      }
44
45      private boolean hitsAnotherBlock(Block block) {
46          return blocks[block.row()][block.col()] != EMPTY;
47      }
48
49      public void drop(Tetrominoe tetrominoe) {
50      }
51
52      public void drop(Block block) {
53          if (hasFalling()) {
54              throw new IllegalStateException("Another block may not be dropped when one is a
55          }
56          fallingBlock = block.moveTo(0, columns() / 2);
57      }
58
59      public boolean hasFalling() {
60          return fallingBlock != null;
61      }
62
63      private void stopFallingBlock() {
64          assert hasFalling();
65          copyToBoard(fallingBlock);
66          fallingBlock = null;
67      }
68
69      private void copyToBoard(Block block) {
```

```
21
22      public static class When_a_piece_is_dropped extends TestCase {
23
24          private Board board;
25
26          protected void setUp() throws Exception {
27              board = new Board(6, 8);
28              board.drop(Tetrominoe.T_SHAPE);
29          }
30
31          public void test_It_starts_from_top_middle() {
32      //          assertEquals("" +
33      //                  "....T...\n" +
34      //                  "...TTT..\n" +
35      //                  "........\n" +
36      //                  "........\n" +
37      //                  "........\n" +
38      //                  "........\n", board.toString());
39          }
40      }
41
```

Make Block implement the same interface as Tetrominoe.

```
11  /**
12   * @author Esko Luontola
13   */
14  public class Block implements Grid {
15
16      private final int row;
17      private final int col;
18      private final char style;
19
20      public Block(char style) {
21          this(0, 0, style);
22      }
23
24      private Block(int row, int col, char style) {
25          this.row = row;
26          this.col = col;
27          this.style = style;
28      }
29
30      public int row() {
31          return row;
32      }
33
34      public int col() {
35          return col;
36      }
37
38      public char style() {
39          return style;
40      }
41
42      public int rows() {
43          return 1;
44      }
45
46      public int columns() {
47          return 1;
48      }
49
50      public char cellAt(int row, int col) {
51          return style;
52      }
53
54      public boolean isAt(int row, int col) {
55          return row == this.row && col == this.col;
56      }
57
58      public Block moveTo(int row, int col) {
59          return new Block(row, col, style);
60      }
61
```

Use Block through the same interface as Tetrominoe.



```
    public boolean hasFalling() {
        return fallingBlock != null;
    }

    private void stopFallingBlock() {
        assert hasFalling();
        copyToBoard(fallingBlock);
        fallingBlock = null;
    }

    private void copyToBoard(Block block) {
        blocks[block.row()][block.col()] = block.style();
    }

    public int rows() {
        return blocks.length;
    }

    public int columns() {
        return blocks[0].length;
    }

    public char cellAt(int row, int col) {
        if (fallingBlock != null && fallingBlock.isAt(row, col)) {
            return fallingBlock.style();
        } else {
            return blocks[row][col];
        }
    }

    public String toString() {
        return new GridAsciiView(this).toString();
    }
}
```

```
    private void stopFallingBlock() {
        assert hasFalling();
        copyToBoard(fallingBlock);
        fallingBlock = null;
    }

    private void copyToBoard(Block block) {
        for (int row = 0; row < blocks.length; row++) {
            for (int col = 0; col < blocks[row].length; col++) {
                if (block.isAt(row, col)) {
                    blocks[row][col] = block.cellAt(row, col);
                }
            }
        }
    }

    public int rows() {
        return blocks.length;
    }

    public int columns() {
        return blocks[0].length;
    }

    public char cellAt(int row, int col) {
        if (fallingBlock != null && fallingBlock.isAt(row, col)) {
            return fallingBlock.cellAt(row, col);
        } else {
            return blocks[row][col];
        }
    }

    public String toString() {
        return new GridAsciiView(this).toString();
    }
}
```

Make Block contain a Grid.

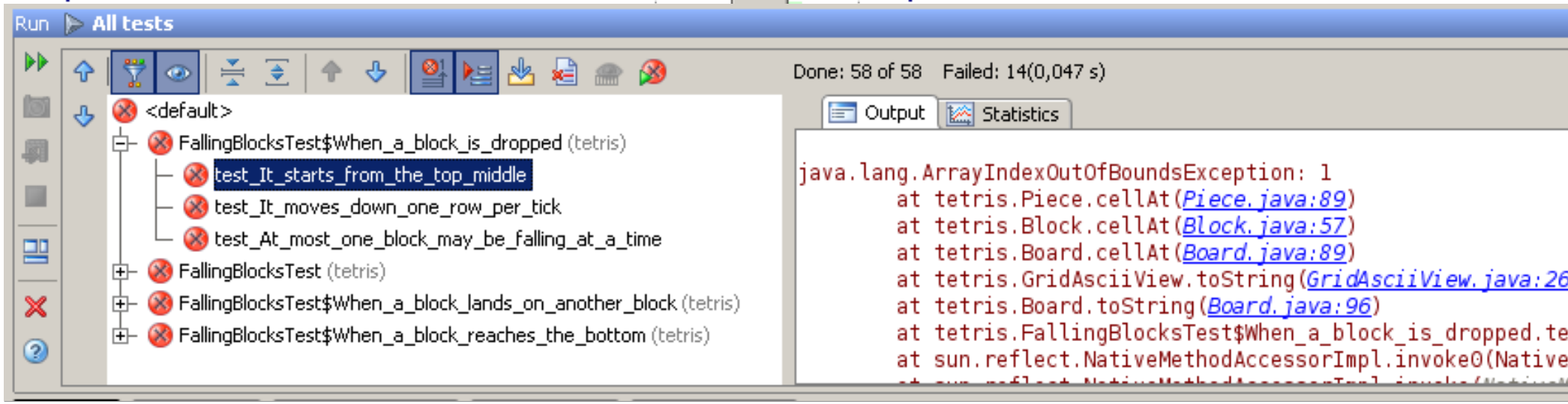
```
10
11  /**
12   * @author Esko Luontola
13   */
14  public class Block implements Grid {
15
16      private final int row;
17      private final int col;
18      private final char style;
19      private final Grid inner;
20
21      public Block(char style) {
22          this(0, 0, style, new Piece(style + "\n"));
23      }
24
25      public Block(Grid inner) {
26          this(0, 0, 'z', inner);
27      }
28
29      private Block(int row, int col, char style, Grid inner) {
30          this.row = row;
31          this.col = col;
32          this.style = style;
33          this.inner = inner;
34      }
35
36      public int row() {
37          return row;
38      }
39
```

```
55
56      public char cellAt(int row, int col) {
57          return style;
58      }
59
60      public boolean isAt(int row, int col) {
61          return row == this.row && col == this.col;
62      }
63
64      public Block moveTo(int row, int col) {
65          return new Block(row, col, style, inner);
66      }
67
68      public Block moveDown() {
69          return new Block(row + 1, col, style, inner);
70      }
71  }
72
```

Change Block to delegate its Grid methods to the contained Grid.

```
public char style() {
    return style;
}

public int rows() {
    return 1;
}

public int columns() {
    return 1;
}

public char cellAt(int row, int col) {
    return style;
}

public boolean isAt(int row, int col) {
    return row == this.row && col == this.col;
}

public Block moveTo(int row, int col) {
    return new Block(row, col, style, inner);
}

public Block moveDown() {
```

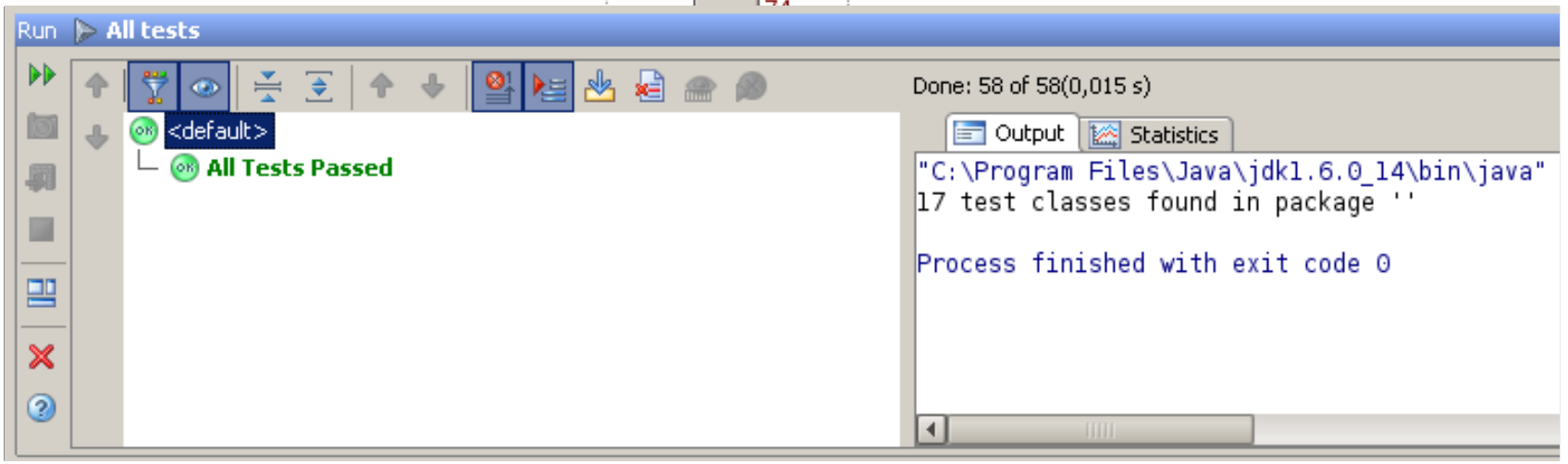
```
public char style() {
    return style;
}

public int rows() {
    return inner.rows();
}

public int columns() {
    return inner.columns();
}

public char cellAt(int row, int col) {
    return inner.cellAt(row, col);
}

public boolean isAt(int row, int col) {
    return row == this.row && col == this.col;
}

public Block moveTo(int row, int col) {
    return new Block(row, col, style, inner);
}

public Block moveDown() {
```



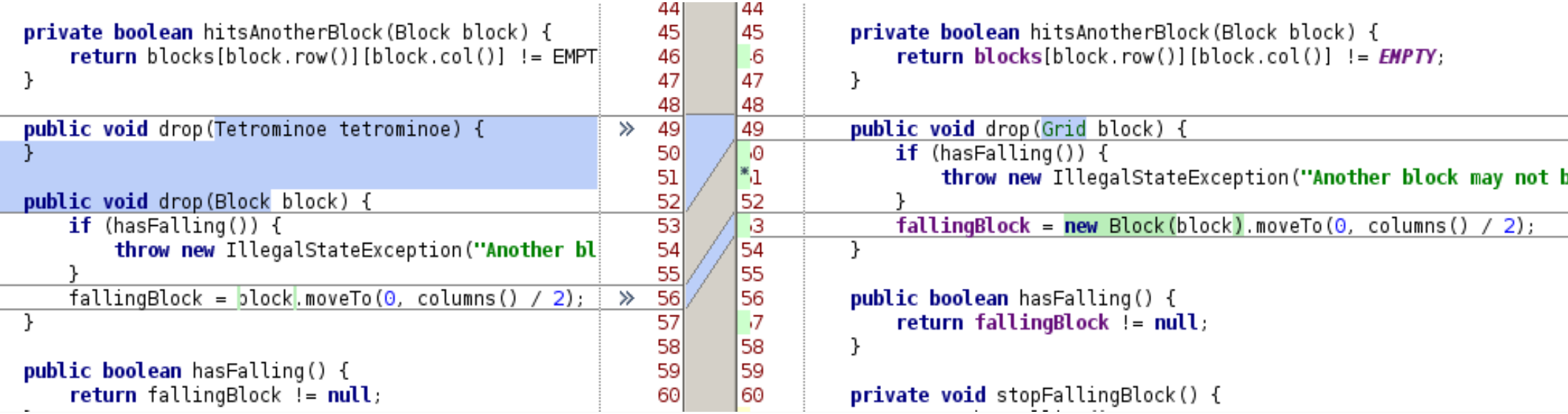
Fix the row/column coordinate mismatch in Block.cellAt().

```
    }

    public int columns() {
        return inner.columns();
    }

    public char cellAt(int row, int col) {
        return inner.cellAt(row, col);
    }

    public boolean isAt(int row, int col) {
        return row == this.row && col == this.col;
    }

    public Block moveTo(int row, int col) {
        return new Block(row, col, style, inner);
    }

    public Block moveDown() {
        return new Block(row + 1, col, style, inner);
    }
}
```

```
    public int columns() {
        return inner.columns();
    }

    public char cellAt(int row, int col) {
        int innerRow = row - rowOffset;
        int innerCol = col - colOffset;
        return inner.cellAt(innerRow, innerCol);
    }

    public boolean isAt(int row, int col) {
        return row == this.rowOffset && col == this.colOffset;
    }

    public Block moveTo(int row, int col) {
        return new Block(row, col, style, inner);
    }

    public Block moveDown() {
        return new Block(rowOffset + 1, colOffset, style, inner);
    }
}
```



Change Board.drop() to accept any Grid.
Block contains the movement logic, so we wrap the Grid in it.

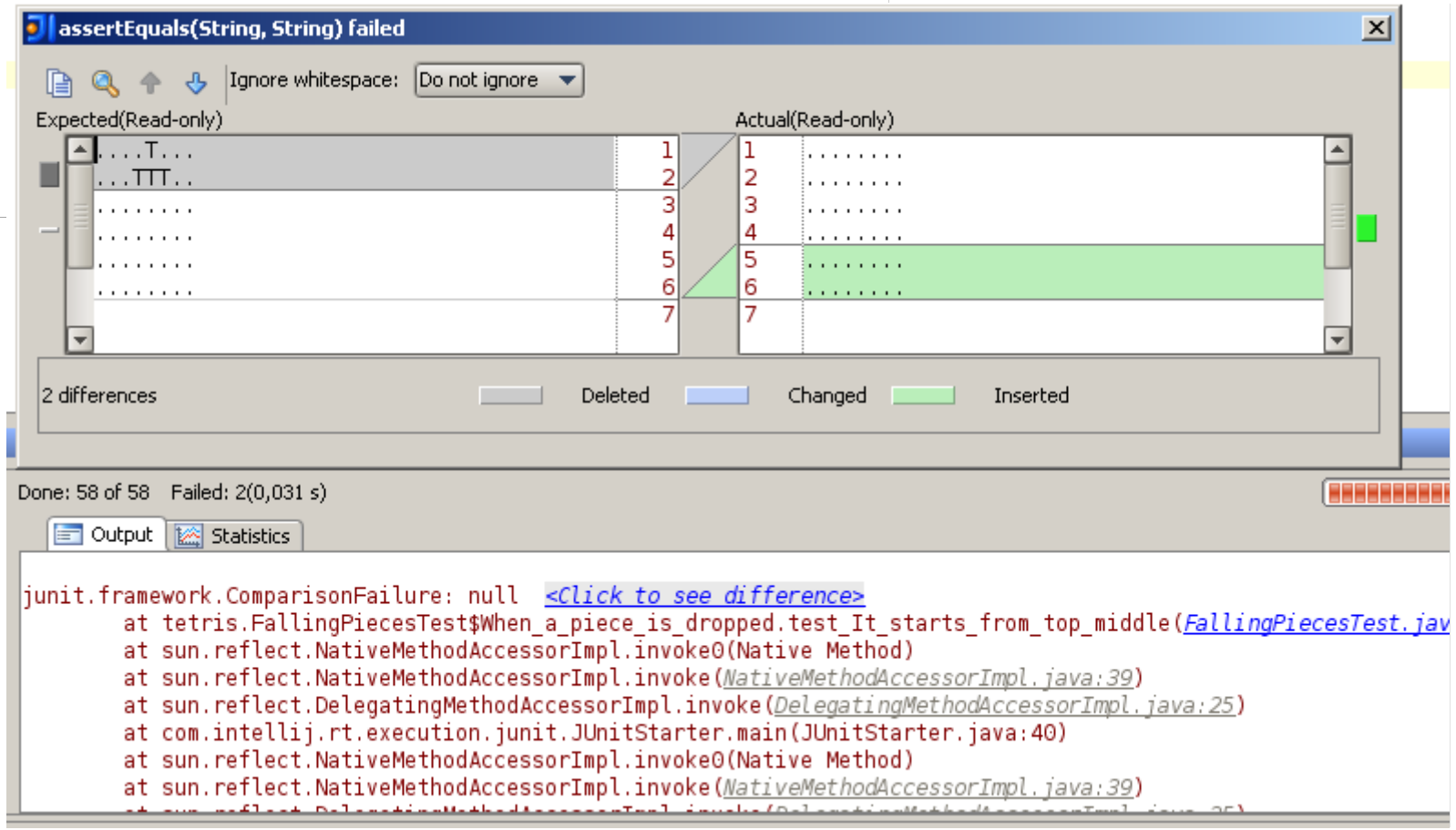


```
private boolean hitsAnotherBlock(Block block) {
    return blocks[block.row()][block.col()] != EMPT
}

public void drop(Tetrominoe tetrominoe) {
}

public void drop(Block block) {
    if (hasFalling()) {
        throw new IllegalStateException("Another bl
    }
    fallingBlock = block.moveTo(0, columns() / 2);
}

public boolean hasFalling() {
    return fallingBlock != null;
```

```
    private boolean hitsAnotherBlock(Block block) {
        return blocks[block.row()][block.col()] != EMPTY;
    }

    public void drop(Grid block) {
        if (hasFalling()) {
            throw new IllegalStateException("Another block may not b
        }
        fallingBlock = new Block(block).moveTo(0, columns() / 2);
    }

    public boolean hasFalling() {
        return fallingBlock != null;
    }

    private void stopFallingBlock() {
```

Try running FallingPiecesTest. It fails.

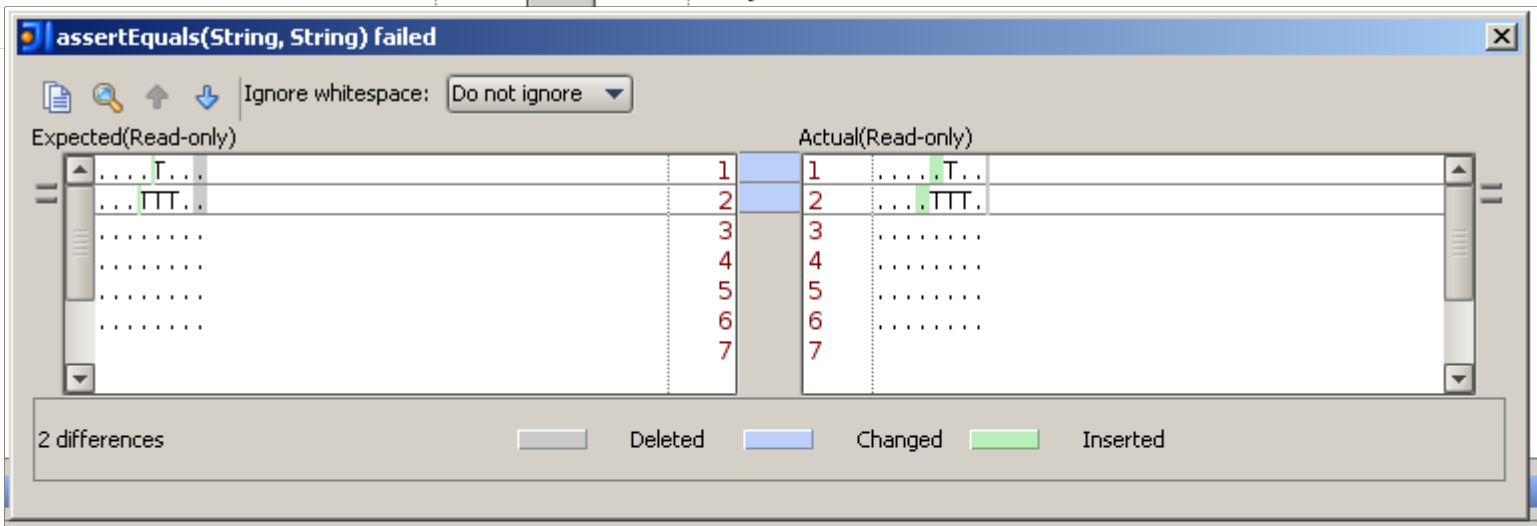
```
21
22      public static class When_a_piece_is_dropped extends TestCase {
23
24          private Board board;
25
26          protected void setUp() throws Exception {
27              board = new Board(6, 8);
28              board.drop(Tetrominoe.T_SHAPE);
29          }
30
31          public void test_It_starts_from_top_middle() {
32              assertEquals("" +
33                      "....T...\n" +
34                      "...TTT..\n" +
35                      "........\n" +
36
37
38
39          }
40      }
41
```



Fix the row/column coordinate mismatch in Block.isAt(). Now FallingPiecesTest *almost* passes.

```
    public char cellAt(int row, int col) {
        int innerRow = row - rowOffset;
        int innerCol = col - colOffset;
        return inner.cellAt(innerRow, innerCol);
    }

    public boolean isAt(int row, int col) {
        return row == this.rowOffset && col == this.colOffset;
    }

    public Block moveTo(int row, int col) {
        return new Block(row, col, style, inner);
    }

    public Block moveDown() {
        return new Block(rowOffset + 1, colOffset, style, inner
    }
}
```

```
    public char cellAt(int row, int col) {
        int innerRow = row - rowOffset;
        int innerCol = col - colOffset;
        return inner.cellAt(innerRow, innerCol);
    }

    public boolean isAt(int row, int col) {
        int innerRow = row - rowOffset;
        int innerCol = col - colOffset;
        return innerRow >= 0 && innerRow < inner.rows() &&
                innerCol >= 0 && innerCol < inner.columns();
    }

    public Block moveTo(int row, int col) {
        return new Block(row, col, style, inner);
    }

    public Block moveDown() {
        return new Block(rowOffset + 1, colOffset, style, inner);
    }
```



Done: 58 of 58 Failed: 2(0,047 s)

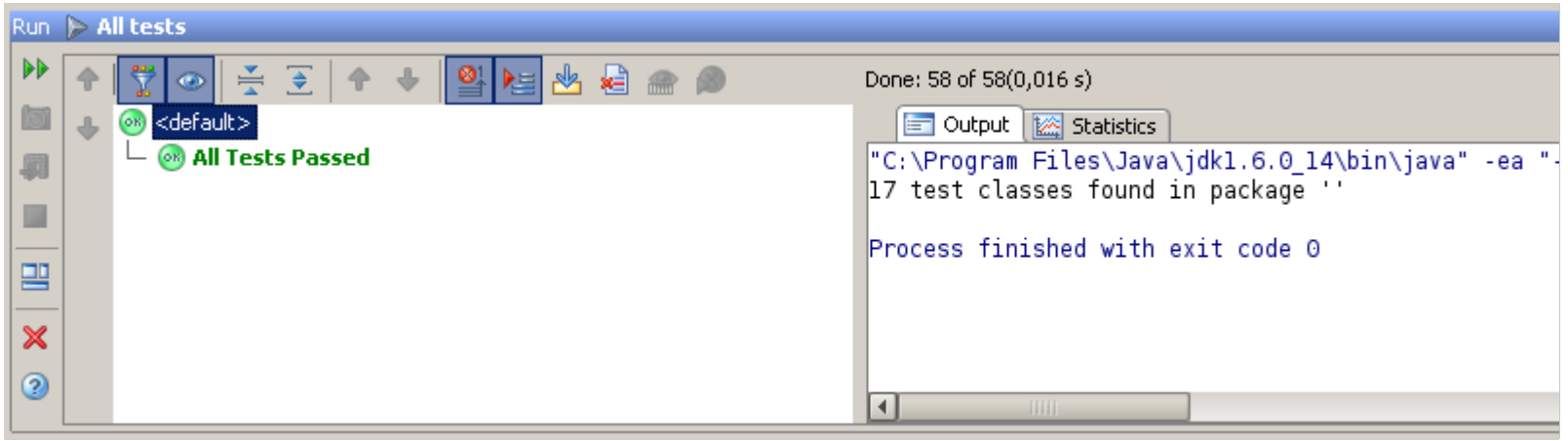
Output | Statistics

```
junit.framework.ComparisonFailure: null <Click to see difference>
        at tetris.FallingPiecesTest$When_a_piece_is_dropped.test_It_starts_from_top_middle(FallingPiecesTest.java:32)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
```

Fix the falling Grid's initial position in Board.drop(). Now FallingPiecesTest passes. (Wohoo!)

```
lock(Block block) {
)][block.col()] != EMPTY;

 {

teException("Another block may not

(block).moveTo(0, columns() / 2);

{
ull;

x() {

);
```

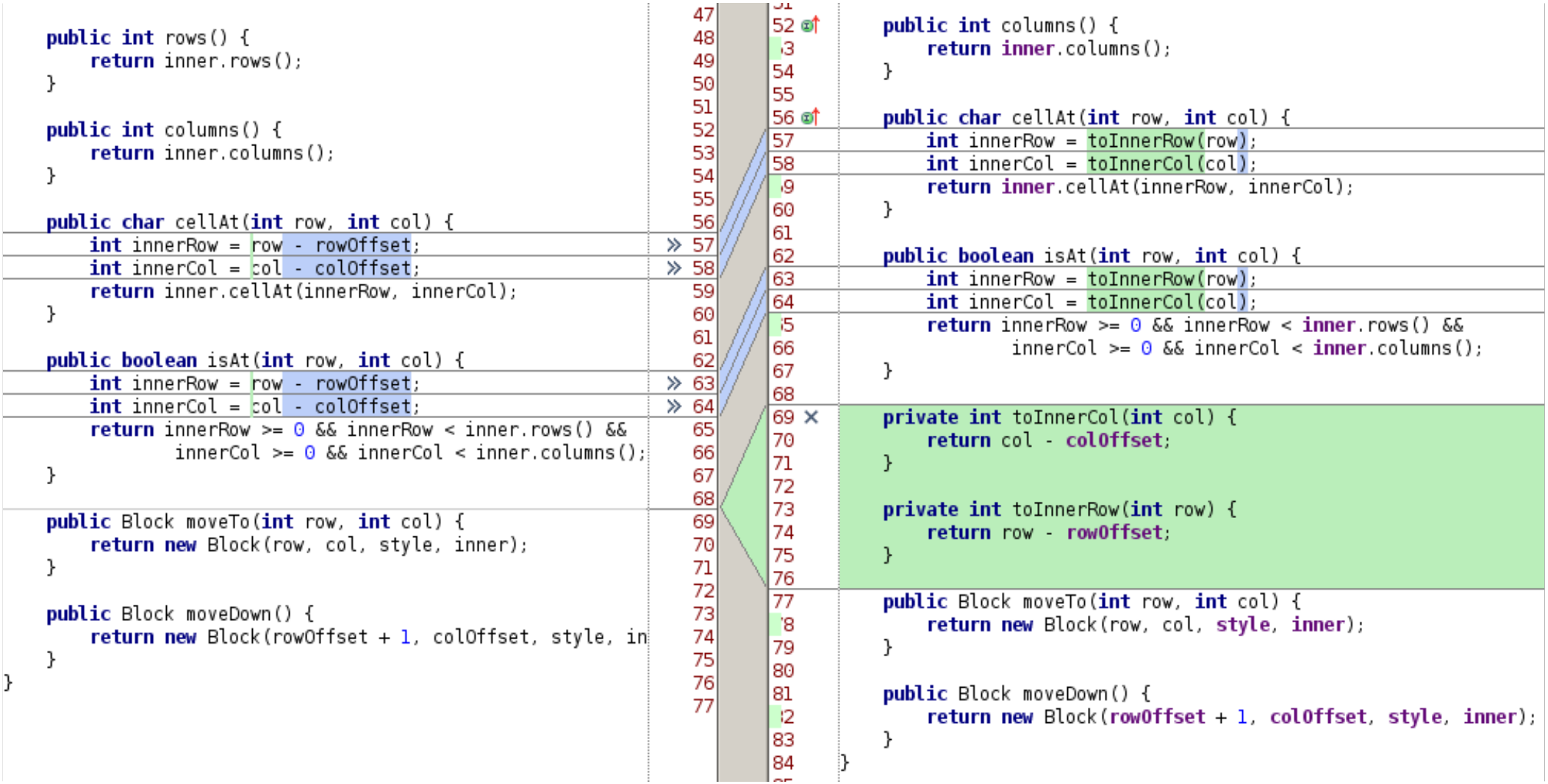
```
45  45      private boolean hitsAnotherBlock(Block block) {
46  46          return blocks[block.row()][block.col()] != EMPTY;
47  47      }
48  48
49  49      public void drop(Grid block) {
50  50          if (hasFalling()) {
51  51              throw new IllegalStateException("Another block may not be dropped when one is alr
52  52          }
53  53          fallingBlock = new Block(block).moveTo(0, columns() / 2 - block.columns() / 2);
54  54      }
55  55
56  56      public boolean hasFalling() {
57  57          return fallingBlock != null;
58  58      }
59  59
60  60      private void stopFallingBlock() {
61  61          assert hasFalling();
62  62          copyToBoard(fallingBlock);
63  63          fallingBlock = null;
64  64      }
```



Block is a mess.

```
 * @author Esko Luontola
 */
public class Block implements Grid {

    private final int rowOffset;
    private final int colOffset;
    private final char style;
    private final Grid inner;

    public Block(char style) {
        this(0, 0, style, new Piece(style + "\n"));
    }

    public Block(Grid inner) {
        this(0, 0, 'z', inner);
    }

    private Block(int rowOffset, int colOffset, char style, Grid inner) {
        this.rowOffset = rowOffset;
        this.colOffset = colOffset;
        this.style = style;
        this.inner = inner;
    }

    public int row() {
        return rowOffset;
    }

    public int col() {
        return colOffset;
    }

    public char style() {
        return style;
    }

    public int rows() {
        return inner.rows();
    }

    public int columns() {
        return inner.columns();
    }

    public char cellAt(int row, int col) {
        int innerRow = row - rowOffset;
        int innerCol = col - colOffset;
        return inner.cellAt(innerRow, innerCol);
    }

    public boolean isAt(int row, int col) {
        int innerRow = row - rowOffset;
        int innerCol = col - colOffset;
        return innerRow >= 0 && innerRow < inner.rows() &&
                innerCol >= 0 && innerCol < inner.columns();
    }

    public Block moveTo(int row, int col) {
        return new Block(row, col, style, inner);
    }

    public Block moveDown() {
        return new Block(rowOffset + 1, colOffset, style, inner);
    }
}
```

Remove the obvious code duplication.



```
    public int rows() {
        return inner.rows();
    }

    public int columns() {
        return inner.columns();
    }

    public char cellAt(int row, int col) {
        int innerRow = row - rowOffset;
        int innerCol = col - colOffset;
        return inner.cellAt(innerRow, innerCol);
    }

    public boolean isAt(int row, int col) {
        int innerRow = row - rowOffset;
        int innerCol = col - colOffset;
        return innerRow >= 0 && innerRow < inner.rows() &&
                innerCol >= 0 && innerCol < inner.columns();
    }

    public Block moveTo(int row, int col) {
        return new Block(row, col, style, inner);
    }

    public Block moveDown() {
        return new Block(rowOffset + 1, colOffset, style, in
    }
}
```

```
    public int columns() {
        return inner.columns();
    }

    public char cellAt(int row, int col) {
        int innerRow = toInnerRow(row);
        int innerCol = toInnerCol(col);
        return inner.cellAt(innerRow, innerCol);
    }

    public boolean isAt(int row, int col) {
        int innerRow = toInnerRow(row);
        int innerCol = toInnerCol(col);
        return innerRow >= 0 && innerRow < inner.rows() &&
                innerCol >= 0 && innerCol < inner.columns();
    }

    private int toInnerCol(int col) {
        return col - colOffset;
    }

    private int toInnerRow(int row) {
        return row - rowOffset;
    }

    public Block moveTo(int row, int col) {
        return new Block(row, col, style, inner);
    }

    public Block moveDown() {
        return new Block(rowOffset + 1, colOffset, style, inner);
    }
}
```

Block has two responsibilities: moving the Grid and being a 1x1 Grid itself.
Copy Block into MovableGrid and remove everything having to do with the 1x1 Grid.

```
 * @author Esko Luontola
 */
public class Block implements Grid {

    private final int rowOffset;
    private final int colOffset;
    private final char style;
    private final Grid inner;

    public Block(char style) {
        this(0, 0, style, new Piece(style + "\n"));
    }

    public Block(Grid inner) {
        this(0, 0, 'z', inner);
    }

    private Block(int rowOffset, int colOffset, char style, Grid inner) {
        this.rowOffset = rowOffset;
        this.colOffset = colOffset;
        this.style = style;
        this.inner = inner;
    }

    public int row() {
        return rowOffset;
    }

    public int col() {
        return colOffset;
    }

    public char style() {
        return style;
    }

    public int rows() {
        return inner.rows();
    }
```

```
/**
 * @author Esko Luontola
 */
public class MovableGrid implements Grid {

    private final int rowOffset;
    private final int colOffset;
    private final Grid inner;

    public MovableGrid(Grid inner) {
        this(0, 0, inner);
    }

    private MovableGrid(int rowOffset, int colOffset, Grid inner) {
        this.rowOffset = rowOffset;
        this.colOffset = colOffset;
        this.inner = inner;
    }

    public int row() {
        return rowOffset;
    }

    public int col() {
        return colOffset;
    }

    public int rows() {
        return inner.rows();
    }

    public int columns() {
        return inner.columns();
    }

    public char cellAt(int row, int col) {
        int innerRow = toInnerRow(row);
        int innerCol = toInnerCol(col);
        return inner.cellAt(innerRow, innerCol);
    }
```

Also make the row/column coordinates explicit.

```
public int columns() {
    return inner.columns();
}

public char cellAt(int row, int col) {
    int innerRow = toInnerRow(row);
    int innerCol = toInnerCol(col);
    return inner.cellAt(innerRow, innerCol);
}

public boolean isAt(int row, int col) {
    int innerRow = toInnerRow(row);
    int innerCol = toInnerCol(col);
    return innerRow >= 0 && innerRow < inner.rows() &&
           innerCol >= 0 && innerCol < inner.columns();
}

private int toInnerCol(int col) {
    return col - colOffset;
}

private int toInnerRow(int row) {
    return row - rowOffset;
}

public MovableGrid moveTo(int row, int col) {
    return new MovableGrid(row, col, inner);
}

public MovableGrid moveDown() {
    return new MovableGrid(rowOffset + 1, colOffset, inner
}
```

```
    public int columns() {
        return inner.columns();
    }

    public char cellAt(int outerRow, int outerCol) {
        int innerRow = toInnerRow(outerRow);
        int innerCol = toInnerCol(outerCol);
        return inner.cellAt(innerRow, innerCol);
    }

    public boolean isAt(int outerRow, int outerCol) {
        int innerRow = toInnerRow(outerRow);
        int innerCol = toInnerCol(outerCol);
        return innerRow >= 0 && innerRow < inner.rows() &&
               innerCol >= 0 && innerCol < inner.columns();
    }

    private int toInnerCol(int outerCol) {
        return outerCol - colOffset;
    }

    private int toInnerRow(int outerRow) {
        return outerRow - rowOffset;
    }

    public MovableGrid moveTo(int rowOffset, int colOffset) {
        return new MovableGrid(rowOffset, colOffset, inner);
    }

    public MovableGrid moveDown() {
        return new MovableGrid(rowOffset + 1, colOffset, inner);
    }
}
```

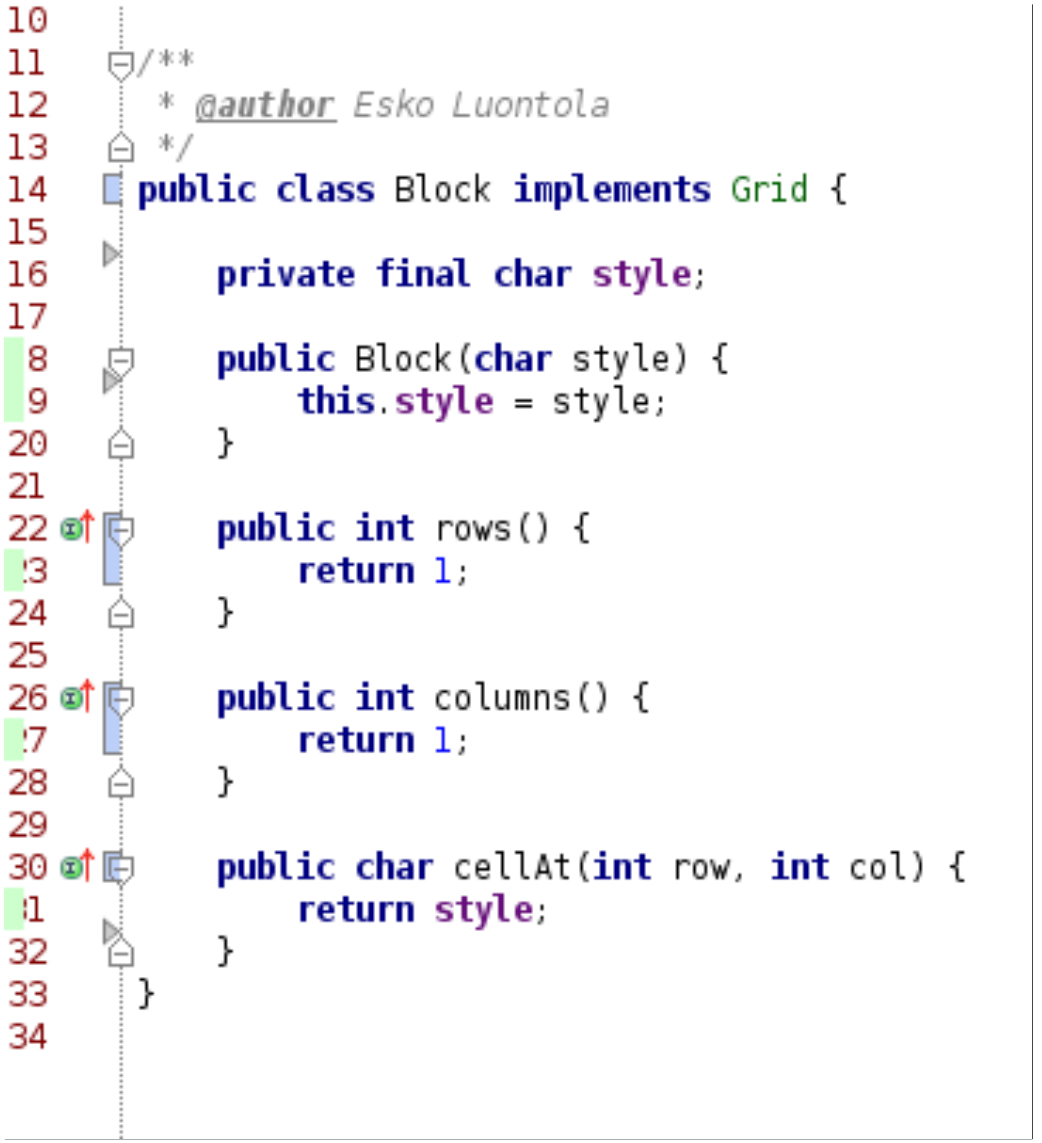
Change Board to use MovableGrid instead of Block.

```
 * @autho
 */
public class Board implements Grid {

    private MovableGrid fallingBlock;
    private char[][] blocks;

    public Board(int rows, int columns) {
        blocks = new char[rows][columns];
        for (char[] tmp : blocks) {
            Arrays.fill(tmp, EMPTY);
        }
    }

    public void tick() {
        MovableGrid test = fallingBlock.moveDown();
        if (conflictsWithBoard(test)) {
            stopFallingBlock();
        } else {
            fallingBlock = test;
        }
    }

    private boolean conflictsWithBoard(MovableGrid block) {
        return outsideBoard(block) || hitsAnotherBlock(block);
    }

    private boolean outsideBoard(MovableGrid block) {
        return block.row() >= rows();
    }

    private boolean hitsAnotherBlock(MovableGrid block) {
        return blocks[block.row()][block.col()] != EMPTY;
    }

    public void drop(Grid block) {
        if (hasFalling()) {
            throw new IllegalStateException("Another block may not be dropped when one is already falling");
        }
        fallingBlock = new MovableGrid(block).moveTo(0, columns() / 2 - block.columns() / 2);
    }

    public boolean hasFalling() {
        return fallingBlock != null;
    }

    private void stopFallingBlock() {
        assert hasFalling();
        copyToBoard(fallingBlock);
        fallingBlock = null;
    }

    private void copyToBoard(MovableGrid block) {
        for (int row = 0; row < blocks.length; row++) {
```

Improve the names in Board.

```
14  * @author Esko Luontola
15  */
16 public class Board implements Grid {
17
18     private MovableGrid falling;
19     private char[][] blocks;
20
21     public Board(int rows, int columns) {
22         blocks = new char[rows][columns];
23         for (char[] tmp : blocks) {
24             Arrays.fill(tmp, EMPTY);
25         }
26     }
27
28     public void tick() {
29         MovableGrid test = falling.moveDown();
30         if (conflictsWithBoard(test)) {
31             stopFalling();
32         } else {
33             falling = test;
34         }
35     }
36
37     private boolean conflictsWithBoard(MovableGrid piece) {
38         return outsideBoard(piece) || hitsStationaryBlock(piece);
39     }
40
41     private boolean outsideBoard(MovableGrid piece) {
42         return piece.row() >= rows();
43     }
44
45     private boolean hitsStationaryBlock(MovableGrid piece) {
46         return blocks[piece.row()][piece.col()] != EMPTY;
47     }
48
49     public void drop(Grid piece) {
50         if (hasFalling()) {
51             throw new IllegalStateException("Another piece may not be dropped when one is already falling");
52         }
53         falling = new MovableGrid(piece).moveTo(0, columns() / 2 - piece.columns() / 2);
54     }
55
56     public boolean hasFalling() {
57         return falling != null;
58     }
59
60     private void stopFalling() {
61         assert hasFalling();
62         copyToBoard(falling);
63         falling = null;
64     }
65
66     private void copyToBoard(MovableGrid block) {
67         for (int row = 0; row < blocks.length; row++) {
```

Remove from Block everything that is already in MovableGrid.
Now Block is used only in FallingBlocksTest, so it can be moved to the test source folder.



```
10
11  /**
12   * @author Esko Luontola
13   */
14  public class Block implements Grid {
15
16      private final char style;
17
 8      public Block(char style) {
 9          this.style = style;
20      }
21
22      public int rows() {
:3          return 1;
24      }
25
26      public int columns() {
:7          return 1;
28      }
29
30      public char cellAt(int row, int col) {
:1          return style;
32      }
33  }
34
```