```java
import org.junit.* ;
import static org.junit.Assert.* ;

public class ModelTest {

   @Test
   /* Tests that alternating chips on a given row don't
    * constitute a win
    */
    public void test_nonHorizontalWin() {
        int playerOne = 1;
        int playerTwo = 2;
        FPModel model = new FPModel() ;
        model.setPiece(5,0,playerOne);
        model.setPiece(5,1,playerTwo);
        model.setPiece(5,2,playerOne);
        model.setPiece(5,3,playerTwo);
        assertFalse(model.winningLine(playerOne));
        assertFalse(model.winningLine(playerTwo));
    }

    @Test
    /* Tests that a horizontal row of 4 chips
    * constitute a win and 3 does not.
    */
    public void test_horizontalWin(){
        int playerOne = 1;
        int playerTwo = 2;
        FPModel model = new FPModel();
        model.setPiece(5,0,playerOne);
        model.setPiece(5,0,playerTwo);
        model.setPiece(5,1,playerOne);
        model.setPiece(5,1,playerTwo);
        model.setPiece(5,2,playerOne);
        model.setPiece(5,2,playerTwo);
        model.setPiece(5,3,playerOne);
        assertTrue(model.winningLine(playerOne));
        assertFalse(model.winningLine(playerTwo));
    }

    @Test
    /* Tests that a diagnal set of 4 chips
    * constitute a win and 3 does not.
    */
    public void test_diagonalWin(){
        int playerOne = 1;
        int playerTwo = 2;
        FPModel model = new FPModel();
        model.setPiece(5,0,playerOne);
        model.setPiece(5,1,playerTwo);
        assertFalse(model.winningLine(playerOne));
        assertFalse(model.winningLine(playerTwo));
        model.setPiece(5,1,playerOne);
        model.setPiece(5,2,playerTwo);
        assertFalse(model.winningLine(playerOne));
        assertFalse(model.winningLine(playerTwo));
        model.setPiece(5,3,playerOne);
        model.setPiece(5,2,playerTwo);
        assertFalse(model.winningLine(playerOne));
        assertFalse(model.winningLine(playerTwo));
        model.setPiece(5,2,playerOne);
        model.setPiece(5,3,playerTwo);
        assertFalse(model.winningLine(playerOne));
        assertFalse(model.winningLine(playerTwo));
        model.setPiece(5,4,playerOne);
        model.setPiece(5,3,playerTwo);
        assertFalse(model.winningLine(playerOne));
        assertFalse(model.winningLine(playerTwo));
        model.setPiece(5,3,playerOne);
        assertTrue(model.winningLine(playerOne));
        assertFalse(model.winningLine(playerTwo));
    }

    @Test
   /* Tests when the model is instantiated it is empty
    * Tests that when it is empty model.boardIsEmpty() returns false
    * Calls model.clearBoard() and check that all pieces are empty
    */
    public void test_resetBoard() {
        int playerOne = 1;
        int playerTwo = 2;
        int i,j;
        boolean pieceFound = false;

        //Clear the board and check the board is empty
        //check model.boardEmpty() returns true.
        FPModel model = new FPModel();
        int[][] boardStatus = model.getChipStatus();

        for(i = 0; i < model.getNoOfRows(); i++){
            for(j = 0; j < model.getNoOfColomns(); j++){
                if(boardStatus[i][j] == 1)
                    pieceFound = true;
            }
        }
        assertFalse(pieceFound);
        assertTrue(model.boardIsEmpty());

        //Set a piece and check model.boardIsEmpty()
        //returns false.
        model.setPiece(5,0,playerOne);
        assertFalse(model.boardIsEmpty());

        //Re-clear the board
        model.clearBoard();
        int[][] boardStatusReinit = model.getChipStatus();

        //Check the board is empty and model.boardIsEmpty()
        //returns true;
        for(i = 0; i < model.getNoOfRows(); i++){
            for(j = 0; j < model.getNoOfColomns(); j++){
                if(boardStatusReinit[i][j] == 1)
                    pieceFound = true;
            }
        }
        assertFalse(pieceFound);
        assertTrue(model.boardIsEmpty());
    }

}
```