Code Review Template

This template is to be completed and submitted by the *Reviewer*.

Names of the Reviewer: **Mike Lasby**

Name of the developer being reviewed: **Davis Allan**

Category	Comments /questions about of the reviewing group about the design documents	Responses by the developer (if any)
Spelling Mistakes		
Naming issues	**SmartPlayer.java** • Line 13: Rename Boolean "block" to "winner" for better "intension-revealing names".	Good idea, this would make it easier to understand
SOLID Principle Violations	**BlockingPlayer.java** • Consider using composition to generate a new RandomGenerator instance for each RandomPlayer. This way, there is no need for classes derived from RandomPlayer to aggregate a RandomGenerator instance. (Single Responsibility Principal) • Method testForBlocking should be refactored into several functions to ensure single responsibility principal. For example, consider adding checkVertical, checkHorizontal, and checkDiagonals. **RandomPlayer.java** • Same point as above, consider composing a RandomGenerator with the RandomPlayer class. Think of it as the RandomPlayer's "brain". **Board.java** • Similar to BlockingPlayer, method checkWinner should be refactored into several functions to ensure single responsibility principal. For example,	I certainly need to focus on following the Single Responsibility and focus on method brevity where needed, this would make it easier to maintain in the future

Category	Comments /questions about of the reviewing group about the design documents	Responses by the developer (if any)
	consider adding checkVertical, checkHorizontal, and checkDiagonals. **Game.java** • Main function is doing lots of heavy lifting. Consider splitting the game state initialization into several functions, for example getPlayerName(char mark). This will help maintain the program if the initialization requirements change and improve code reuse (Single responsibility + DRY). • create_player function also have several responsibilities. I recommend using some primitive getInt() or similar functions when prompting the user for input. This will improve code reuse and declutter the create_player function. **HumanPlayer.java** • makeAMove() has repeated code in checking bounds of user input. This logic should be moved into a separate function to create a more open and maintainable function. This will also ensure that the makeAMove() function has a single responsibility. • Great job using the isValidMove() function to improve readability and code reuse. **Player.java** • Consider moving the if statement for checkWinner and board state checks into a new function. For example, the checkWinner function could call a displayResult function directly if the game has been won. This will ensure that play() simply changes the active player and and logical checks on game state are the responsibility of separate functions. **SmartPlayer.java** • Similar to BlockingPlayer, method checkWinner should be refactored into several functions to ensure single responsibility principal. For example, consider adding checkVertical, checkHorizontal, and checkDiagonals.	
Lack of documentation	**BlockingPlayer.java** • No class or method javaDocs in BlockingPlayer.java **Board.java** • Consider adding type information to Javadoc strings eg., "@param row (int) the row…"	I missed adding docs in a few of the files, good reminder to be more diligent!

Category	Comments /questions about of the reviewing group about the design documents	Responses by the developer (if any)
	Game.java • No class javaDocs **HumanPlayer.java** • No class or method javadocs **Player.java** • Missing Javadoc for play() method **RandomPlayer.java** • No Javadocs **Referee.java** • Missing class header Javadoc decorators (@author, etc.)_ **SmartPlayer.java** • Missing class and method Javadoc	
Clean Code	**Game.java:** • Catch and deal with IOException here to avoid throwing in client facing code. • In function create_player, split mult-line string across several string objects to improve readability.	
Security	**BlockingPlayer.java** • Make helper method testForBlocking private. **Player.java** • Consider making data members private and adding getters/setters	Good reminders
Performance	**BlockingPlayer.java** • Line 20: Consider adding a return statement to bypass the if statement below. If move is made, no need to check Boolean flag, simply return. If move is not made, call super.makeAMove(). • Line 29: Add immediate check if space is occupied. If occupied, you may immediately return that no blocking / winning position is possible and skip all other conditional checks. **Player.java** • Line 67: "Beware the cost of string concatenation" https://www.corejavaguru.com/effective-java/items/51 . Consider using	Those changes to the logic make a lot of sense!

Category	Comments /questions about of the reviewing group about the design documents	Responses by the developer (if any)
	String.format, StringBuffer, or similar methods to avoid building multiple String objects. **SmartPlayer.java** • Similar to BlockingPlayer, consider checking if move is valid prior to checking for winner/blocker. If move is not valid, we immediately can conclude it's not a possible option.	
General		

Review aided by the following checklist: https://dzone.com/articles/java-code-review-checklist