Refactoring:

1. Coupler Smell: We moved all the methods from our MakeMove class into our GameController class, and removed the MakeMove class all together because of Inappropriate Intimacy (All of the methods in MakeMove were being used by GameController).
2. Dispensable Smell: We removed the UniqueRandomNumbers class because it was dead code (code not being used). The Original intent of that class was to generate a unique random ID for our users, but the database handles the creation of unique IDs.
3. Coupler Smell: We removed our GameDriver class because all it was only acting as a middle man between the GameController class and the UI, so we deemed it unnecessary and connected the GameController class directly with the UI instead.
4. Dispensable Smell: We removed our DatabaseDriver class because it was dead code. We created the class to help communicate with the database, but we never used it.
5. Coupler Smell: We removed our Space class because it was acting as a middle man between Piece and Board. Piece and Board are now connected directly instead and a board consist of different types of pieces including Rook, King, or None.
6. Bloater smell: Our GameController class is still bloated, but to attempt to break it down, we converted a lot of the code to methods in a class called DatabaseTransalator. GameController now just must call the methods from that class.
7. Object-Oriented Abuser smell: We had a lot of temporary fields and in order to attempt to get rid of some of those we added some of those fields to the constructors of other classes instead.
8. Dispensable smell: We had some Data Classes (classes that did nothing more but act as containers for information that other classes used) in the Database side of our source code. We got rid of these by simplifying the way our database stores and retrieves information.

Design Patterns:

1. Interpreter behavioral design pattern: To translate objects pulled from the database to objects used in our backend code. This will allow us to call the builder methods when we need to translate objects from the database instead of having to code a translation algorithm in every class that we requires a translation.
2. Singleton creational design pattern: We made most of the methods in the classes (DatabaseTranslator and DatabaseManagerImp) static so that their methods can be used throughout the program instead of having to make an instance every time.