OOP Design Explanation

In this program, four main classes are used to structure the game: Line, Block, Ball, and Level (excluding App.java).

1. **Line.java**: This class represents a line drawn by a player on the map. Each line corresponds to a separate instance of the Line class. It manages line-specific properties, such as storing and adding points. The method for adding points allows the line to extend as the player continues drawing.
2. **Block.java**: This class represents various objects on the game board, including walls, holes, tiles, and spawners. Instead of creating subclasses for each type of block, a single class (Block) was used to simplify array management, as the board is represented by a Block[][] array. Using subclasses would limit the flexibility of managing the different block types within the array. The class stores important attributes such as coordinates, block type, and color. It ensures that all block types can be handled uniformly within the game's logic.
3. **Ball.java**: This class encapsulates the properties of a ball in the game, where each ball has its own instance of the class. The Ball class contains attributes such as radius, speed, and current coordinates, ensuring that each ball behaves independently on the game map.
4. **Level.java**: This class represents a game level. Instead of having separate instances for each level, a single Level instance is reused, with the setup being loaded from different files as the player progresses. The class manages information about the current level, including the score, lines drawn, balls in play, and the level status (e.g., paused). This design allows for easy extension and flexibility in level creation without needing significant changes to the code structure.

How the Extension Has Been Implemented

For the extension, I implemented the feature "Bricks – become progressively more damaged when balls hit them and then disappear after being hit 3 times." To achieve this:

- I added a new variable in the Block class to track the number of collisions between the ball and any wall-type block. Each time a corresponding ball hits a wall, the counter increments.
- Once the wall block is hit three times, it changes its appearance by displaying a tile instead of the wall image, effectively simulating the wall being destroyed.

- Additionally, after three hits, the wall block becomes unbouncy, allowing balls to pass through it, thereby affecting the game dynamics.

This extension leverages the existing Block structure, minimizing changes to the overall game logic while adding the desired functionality.