Front Range Game Designers

Scenario

Congratulations! Your hard work in Object Oriented Analysis and Design has resulted in a skill set that impressed a recruiter for your favorite company, *Front Range Game Designers*, and you've just started a great summer internship! *Front Range Game Designers* is working on some really exciting games, including *Code Bandicoot 2*, a platformer game, and *Code Bug Crush Saga*, a puzzle game.

The development team for both of these games have made some great games. Unfortunately, they're really lazy and only made games to be played on an NES and controlled by a game pad. Now, all these games need to be ported to PC and be controlled using a keyboard. There are way too many games to make serious modifications, but during your interview, you pointed out your wonderful skills in creating flexible, reusable code.

Problem

Every game is controlled through a gamepad. The software interface involves creating a *GamePad* object, and checking the controller state using the *getControllerState* method. The state is represented as a map from strings ("UP", "DOWN", "LEFT", "RIGHT", "A", "B") to boolean variables indicating if the button is pressed or not.

Keyboards behave very differently. Interfacing with a keyboard involved getting a keyboard instance from the Keyboard class via the *Keyboard.getInstance()* method. Once this is obtained, a listener is registered using the *setListener* method. The listener has two callback methods - *onKeyPressed* and *onKeyReleased*, which get called when something on the keyboard is pressed or released.

Deliverables

1. Identify the design pattern you used to solve this problem, and the participants (i.e., the roles each class takes).
2. An implementation in a language of your choice. You should only need to create one or two new classes / interfaces. You may not modify the Keyboard, GamePad or KeyboardListener classes / interfaces. You can change the type of *gamePad* in the Game class, and for testing, swap which class gets instantiated in the constructor.

3. A class diagram of your solution (including existing classes), so future developers can easily see how to work with your solution.