Class Diagram Description:

Our GizmoBall implementation incorporates the MVC design pattern which splits our design to three parts: Model, View, and Controller.

GizmoModel:

Contains all back-end functionality. It includes the Gizmo interface which is implemented by different gizmos such as Bumpers, Absorbers, and Flippers. Each Gizmo has its individual length, trigger, and collision details. Additionally, the class GizmoModel includes methods to move, get, and add balls. Furthermore, it extends the observable class to communicate with View and Controller.

GizmoView:

Contains all front-end functionality. The user-interface including the board, buttons, and bars are created and displayed in this class. It uses 2 main interfaces, GBallGui and Board. The former will has methods to create buttons, menu bar, and message to display to the user and is implemented by the two classes RunGui and BuildGui which are the two possible modes of the game Run and Build modes. The latter is a Board interface which is implemented by RunBoard and BuildBoard classes. These classes are used by GizmoView to create the GUI board. Board classes call update() to notify other classes of any changes.

Controller:

The controller side of our implementation includes different listeners which handle user actions. The class diagram includes three listener interfaces: GizmoListener, KeyboardListener, and MouseListener. GizmoListener is implemented by RunListener and BuildListener which include the methods actionPerformed(), setListener(), and update(). RunListener calls moveBall() from GizmoModel whenever actionPerformed() is called. KeyboardListener classes RunKeyboardListener and BuildKeyboard listener handle keyboard presses in both run modes. They will be responsible for connecting gizmos and some functionality to keyboard shortcuts. Finally, addGizmoListener() will notify the view whenever the user clicks on the addGizmo button.

Observable:

Extends the Observable class from Java’s API. Contains the methods addObserver() which adds an Observer to a list of observers, notifyObservers() to notify observers of any changes, and setChanged() to set a particular observer as changed.

Observer:

This interface is implemented by Observers and contains a single method: update(). This method notifies other classes of any changes in observers. Listeners and the Board implement Observer to notify other parts of the program of user input and any changes that occur.