## Unit 2 project – Bowling Alley (Enhancement)

## (Due 20th March, 11:55 pm)

This Unit 2 project is an extension of Unit 1 with newer requirements. Hence you will continue with the same team as Unit 1. You are required to use patterns as needed. This project will require you to use your previous Bowling Alley Code and make the following enhancements implementing Gang of Four Design patterns and other UI patterns.

Principles such as coupling, cohesion, delegation vs. inheritance, assignment of responsibilities, elimination of bad code smells, metric values, etc. should continue to be in your consideration while implementing the patterns. The enhancements should be included in your refactored code and the intention is not to throw away the design and start with a clean sheet of paper. As part of the new requirements, you may have to further refactor your code. Each team should document their analysis of the existing design patterns including its strengths and weaknesses, implementation of design patterns to incorporate the new requirements in the dimensions of the design principles.

### New Requirements

1) **Make the bowling alley game interactive using possible UI patterns.**
   This is your chance to show creativity in making it interactive. If it means that simulation is replaced by an actual set of players playing the game... so be it!
2) Make the code extensible and working for multi-player, let the maximum number of players be 6. Provide an option to add and store players names
3) Add database layer to implement persistence of the scores and players. Provide a searchable view to make ad-hoc queries on the stored data. Some possible queries - highest/lowest scores, Top Player, etc.
4) Implement penalty for Gutters – On bowling two consecutive gutters, the player should be penalized 1/2 points of the highest score obtained. If the first 2 consecutive times are at the start of the game, player would be penalized 1/2 of the points that is scored in the next frame.
5) At the end of 10 frames, provide a chance to the 2nd highest player to bowl. If the player becomes highest, continue with 3 additional frames between 1st and 2nd highest till the winner is finalized. If there is a tie-break, declare the winner that had most strikes.
6) Implement an emoticon for appreciate, envy, embarrass and other possible actions based on player score.
7) All the numbers quoted and existing in the code should be made configurable.

### Submission Instructions

1. Create a zip file named **TeamMembers_Unit2.zip** (where ***TeamMembers*** are your team members first names) containing three directories name src (for source code), doc (for the major document), misc (for all miscellaneous documents).
2. The src directory should contain all the source code (appropriately packaged – if necessary)

3. The doc directory should contain a single design document in either PDF format. It should contain the information specified below. Note it would not be an effective presentation of your design to take each item below, stack them one after the other and staple it together. You need to weave this information through your document in a manner that tells a cohesive story with prose guiding the reader and tying the sections together.
    I. Title information, including the name of the project, the date of submission, a list of all the team members, effort (number of hours) put in by each team member, role played by each team member.
    II. A short overview section describing the product and the features included. UML class diagrams showing the main classes and interfaces in your design, along with inheritance (generalization), association, aggregation, and composition relationships.
    III. Include cardinality and role indicators as you deem appropriate to make the diagram clear. Indicate the role a class plays in any patterns in which it participates. You may decide on the appropriate level of abstraction with respect to state or method information.
    IV. You may need several class diagrams at different levels of abstraction and for different subsystems to completely document your design in a way that the reader can physically see and intelligently understand.
    V. One or more other UML diagrams (e.g., sequence diagrams) to provide insight into the key static and dynamic characteristics of the program both before and after refactoring.
    VI. A table summarizing the responsibility(ies) of each major class.
    VII. A narrative analyzing the original design (note: the "original" design is what is in the code you reverse engineered), its weaknesses and strengths, fidelity to the design documentation, and use of design patterns.
    VIII. A narrative outlining how the design reflects a balance among competing criteria such as low coupling, high cohesion, separation of concerns, information hiding, the Law of Demeter, extensibility, reusability, etc. This should include a discussion of the design patterns used to achieve this balance.
    IX. Discussion of your metrics analysis of up to eight metric values including answers to the following questions:
        a. What were the metrics for the code base? What did these initial measurements tell you about the system before implementing new requirements?
        b. How did you use these measurements and your understanding to guide implementation of new requirements?
        c. In the misc document, you may put in all the miscellaneous management related documents. For example, task trackers, meeting minutes, etc.
4. Push your code into GitLab and provide access to TAs.