**Discussion about the Project**

* The files which do not need refactoring
  1. ControlDeskObserver.java
  2. LaneObserver.java
  3. LaneEventInterface.java
  4. LaneServer.java
  5. PinsetterObserver.java
  6. BowlerFile.java
  7. ScoreHistoryFile.java
  8. Queue.java
  9. Score.java
  10. Party.java
  11. ControlDeskEvent.java
  12. Alley.java (check)
  13. Drive.java (check)
  14. PinsetterEvent.java (check)
  15. LaneEvent.java (check whether it can be divided into subclasses or not as it is a comparatively long class)
* The files which need refactoring (the list is not complete)
  1. EndGamePrompt.java
  2. EndGameReport.java
  3. LaneStatusView.java
  4. Bowler.java
  5. LaneEvent.java (check)
  6. PrintableText.java
* Bowler.java has two methods which are doing the same thing basically. ‘getNickName’ and ‘getNick’. So, it’s a redundancy. Only keeping one method will suffice.
* Bowler.java has a method called ‘equals’ which has no instance of invocation from any of the files the in the codebase. So, we can remove and check whether the method is actually required or not.
* Alley.java is the outer container for the bowling sim. It’s just returning the object of ‘ControlDesk’ class. So, we should either keep it or remove it. No refactoring required.
* LaneEvent.java (check whether it can be divided into subclasses or not as it is a comparatively long class)
* PrintableText.java has a long method ‘print’. We can split it into a helper sub-class. For eg. From line 35 to line 44 we can shift it to a helper class.
* In some files, the naming convention for the variables are not that much understandable. We can give more meaningful names. For eg. In Drive.java ‘Alley a = **new** Alley( numLanes );’ . We can rename the variable ‘a’ to some other meaningful name.
* Drive.java also does not need no refactoring. In case we remove the Alley.java file, then we need some cosmetic change.
* EndGamePrompt.java has a long constructor. Mainly it’s setting a variable and rest all are UI changes. So, we can shift all the other changes to a method and call that method from the constructor itself.
* EndGameReport.java has a long constructor. Mainly it’s setting a variable and rest all are UI changes. So, we can shift all the other changes to a method and call that method from the constructor itself.
* In some files there are deprecated methods like ‘win.~~show~~()’ and ‘win.~~hide~~()’. We need to update these methods with more recent methods.
* Also, check for dead ends in the codebase (where the control does not reach). For eg. Bowler.java has a method called ‘equals’ which has no instance of invocation from any of the files the in the codebase. So, we can remove and check whether the method is actually required or not.
* LaneStatusView.java has a long constructor which is using two variables called ‘lane’ and ‘laneNum’ in just the starting few lines of the code (line 33-44). Rest all the portion of the constructor is untouched from these two variables. So, we can shift the rest of the changes in other methods. Another observation is line 74-79 is maintenance code. So, we can actually make 2 classes. One for the maintenance one and another for rest of the things.
* Also, lot of classes are sharing the same UI codes, just check whether these can be invoked from one place only and thus reused.