Design Document

SWE Project 1

Team-9

Table of contents

PROJECT INFORMATION

Team members and roles

Name	Roll number	Responsibilities
Abhinav Vaishya	2018121003	● **Files handled :** ○ Model/Bowler.java ○ Model/BowlerFile.java ○ Model/ControlDesk.java ○ Model/Lane.javaModel/Party.java ○ Model/Pinsetter.java ● **Final metrics calculation**
Meher Shashwat Nigam	20171062	● **Files handled :** ○ drive.java ○ ViewControl/AddPartyView.java ○ ViewControl/ControlDeskView.java ○ ViewControl/EndGamePrompt.java ○ ViewControl/EndGameReport.java ○ ViewControl/NewPatronView.java ● **Final UML class, sequence diagrams**
Sanchit Saini	20171191	● **Files handled:** ○ ViewControl/LaneStatusView.java ○ ViewControl/LaneView.java ○ ViewControl/PinsetterView.java ○ ViewControl/PrintableText.java ○ ViewControl/ScoreReport.java ● **Initial UML class, sequence diagrams**
Souptik Mondal	2019201090	● **Files handled :** ○ Model/PinsetterEvent.java ○ Model/Queue.java ○ Model/Score.java ○ Model/ScoreCalculator.java ○ Model/ScoreHistoryFile.java ○ Model/ShowScores.java ● **Initial metrics calculation**

Date of submission

February 12th, 2021

PROJECT OVERVIEW

The original project contains a java application that allows for the management of a bowling alley. Within the application, a collection of lanes can be monitored through a control desk, parties of bowlers can be assigned to the lane, the configuration of the pinsetter can be viewed, and the bowler's scores can be printed at the completion of the game.

This original source material contained designs, code, and documentation that exhibits poor software engineering design principles as well as anti-patterns.

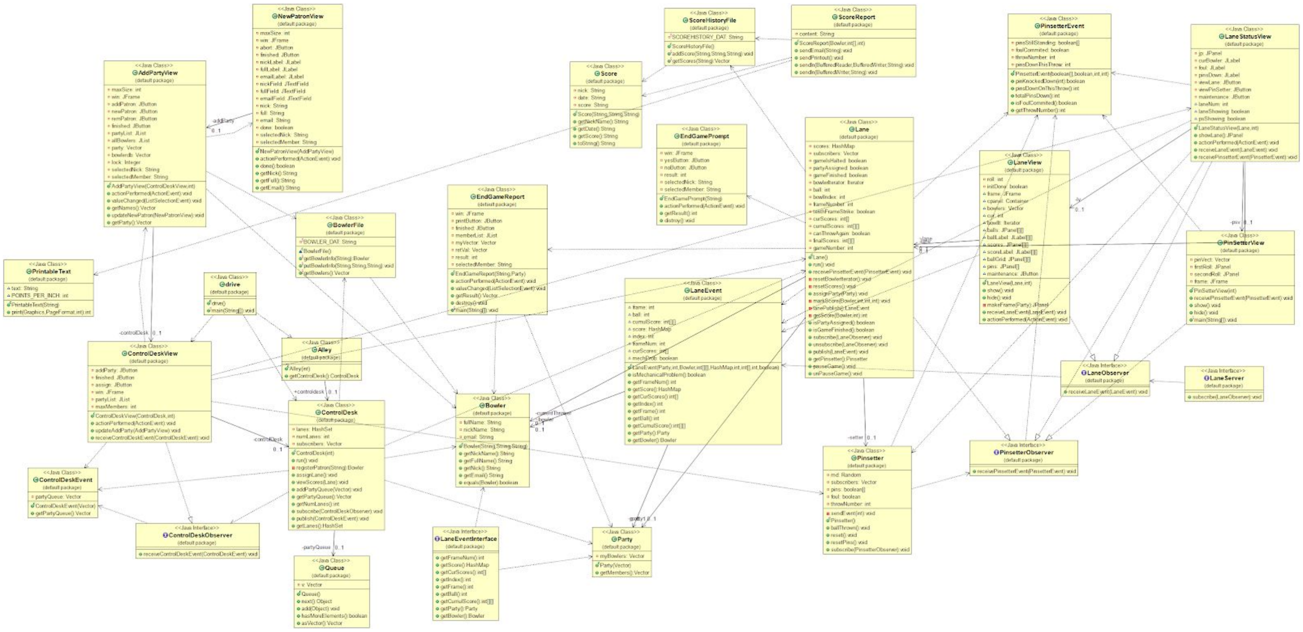
Our purpose was to improve the designs and source code of this application. Metrics were gathered for the project, and along with visual code and documentation inspection potential areas for refactoring were determined.

The features and models can be seen as per the UML Class and Sequence diagrams given below.

UML CLASS DIAGRAMS

Interfaces, inheritance, generalization, association, aggregation, composition, cardinality and role indicators have been shown via appropriate arrows and markings.

All classes



High resolution image link

Lane Classes

All the classes and interfaces are divided into major parts. Below is the UML class diagram for the lanes, lane display and lane events.

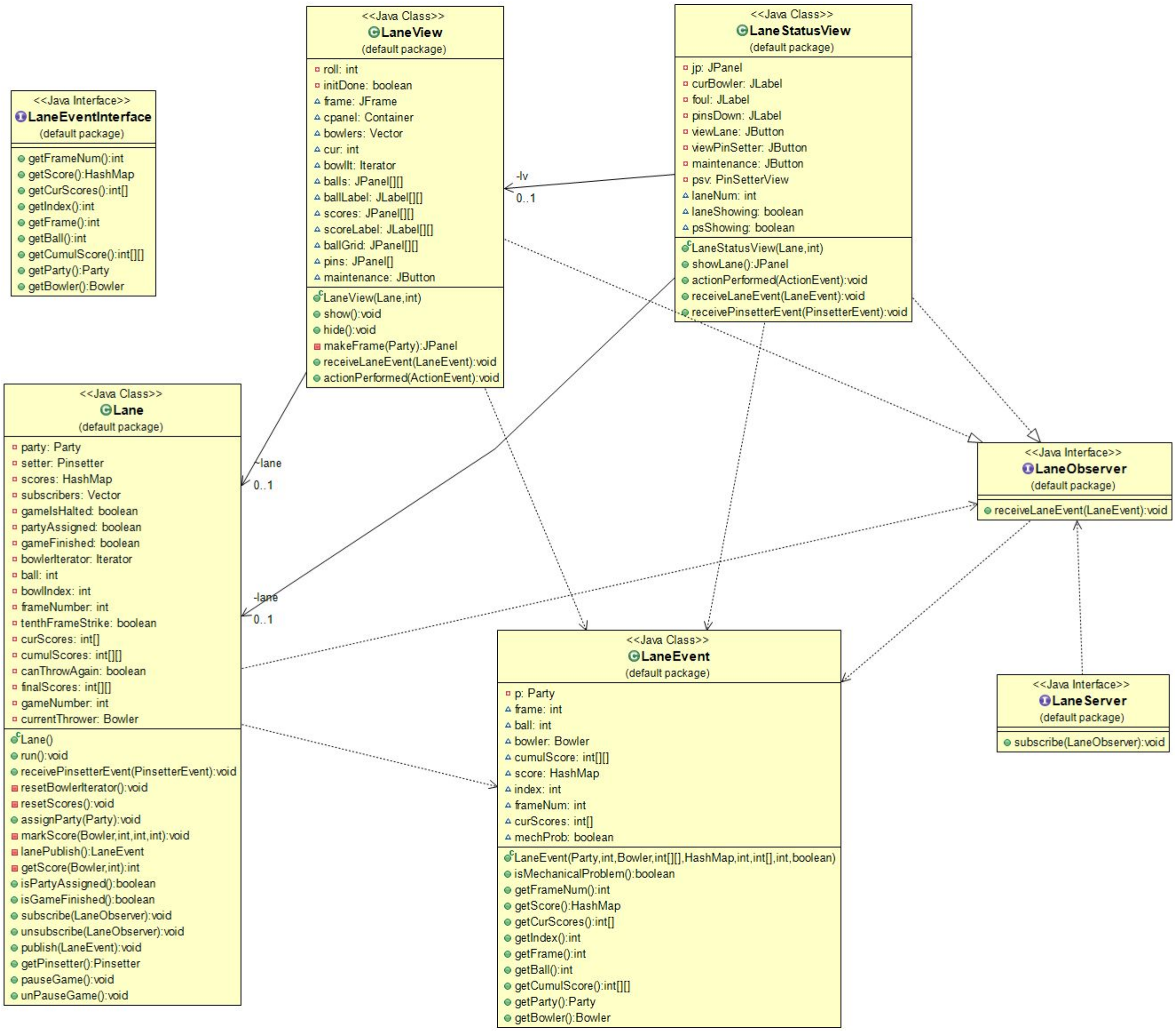
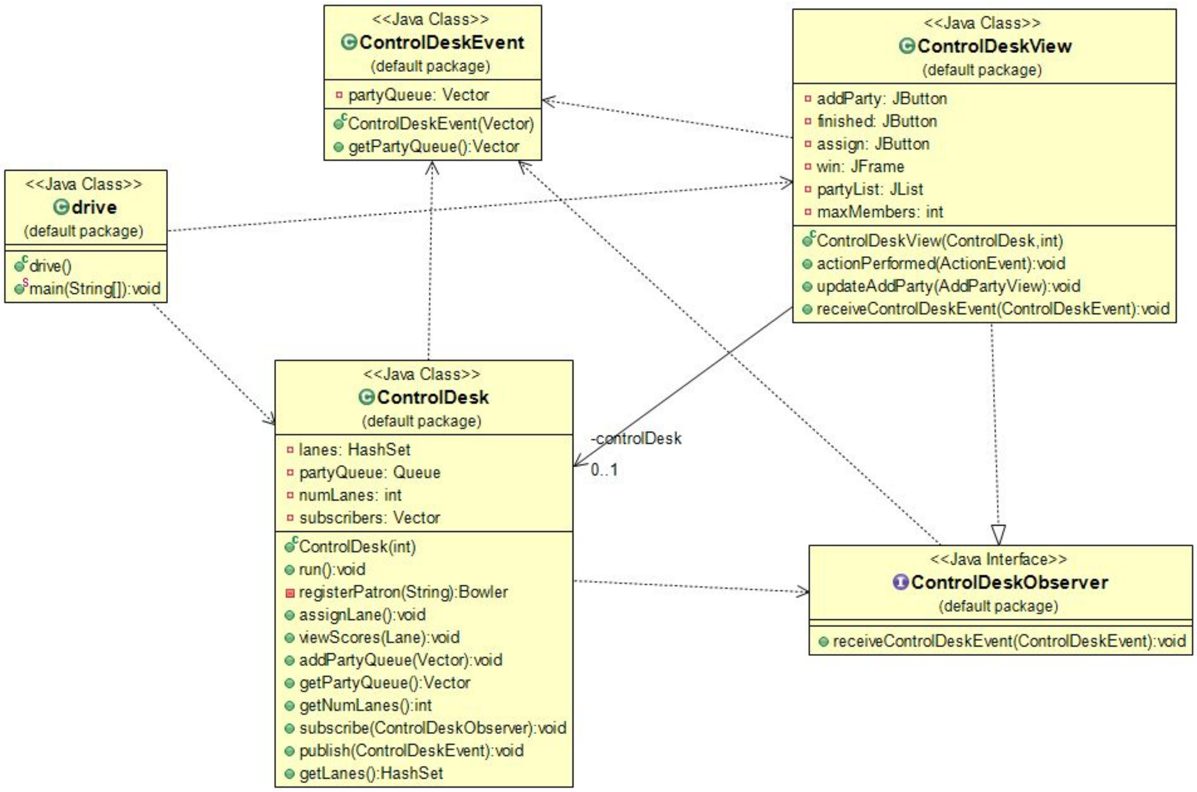


Fig. 1 : UML Class diagram for Lane Classes and Interfaces

Control Desk Classes

Another set of related classes and interfaces are Control Desk related. Following is the UML Class Diagram for the control desk events, control desk view and control desk.

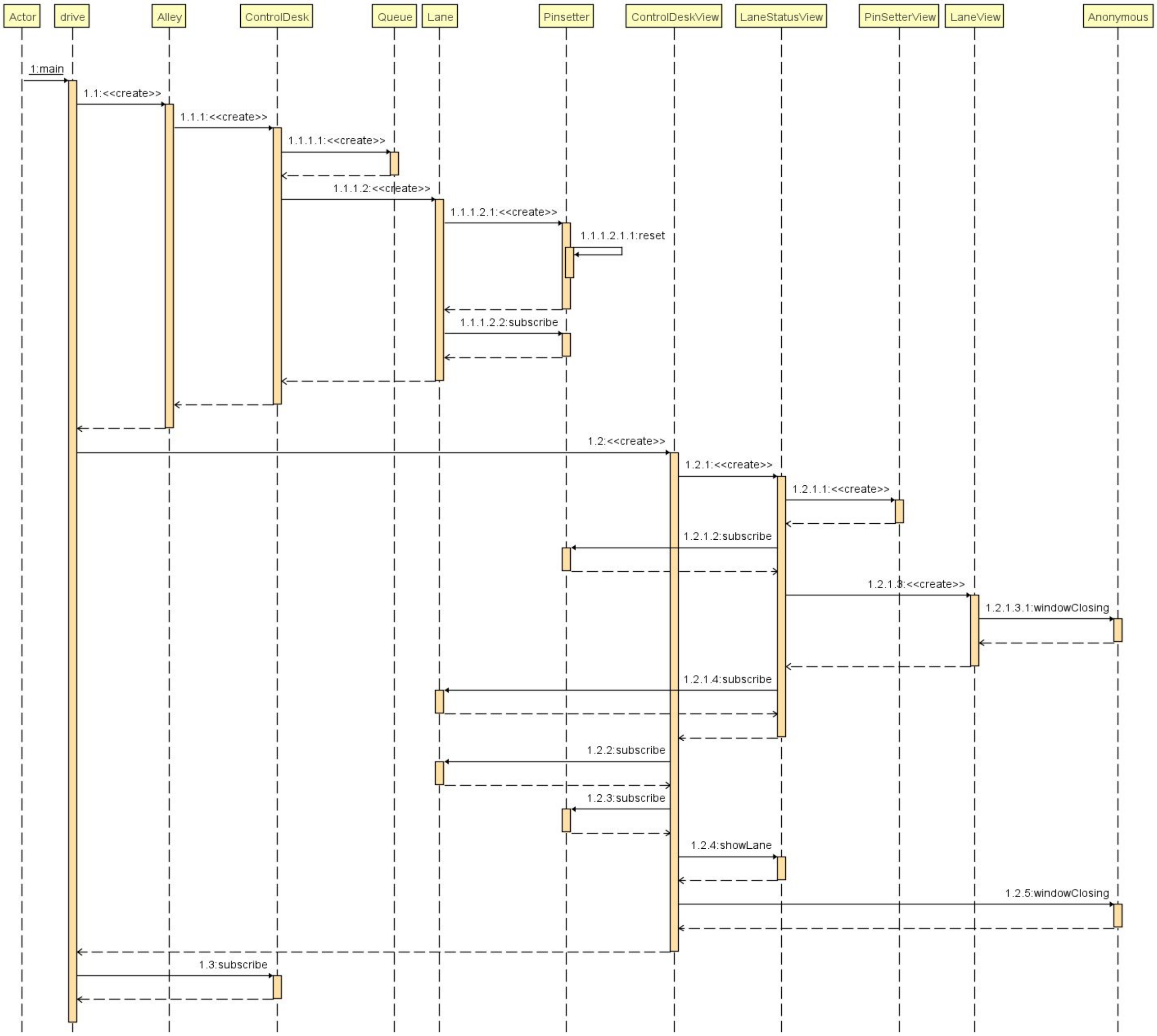


UML SEQUENCE DIAGRAMS

A sequence diagram is a type of interaction diagram because it describes how—and in what order—a group of objects works together. Below are some sequence diagrams for better understanding of the workflow of the whole code base.

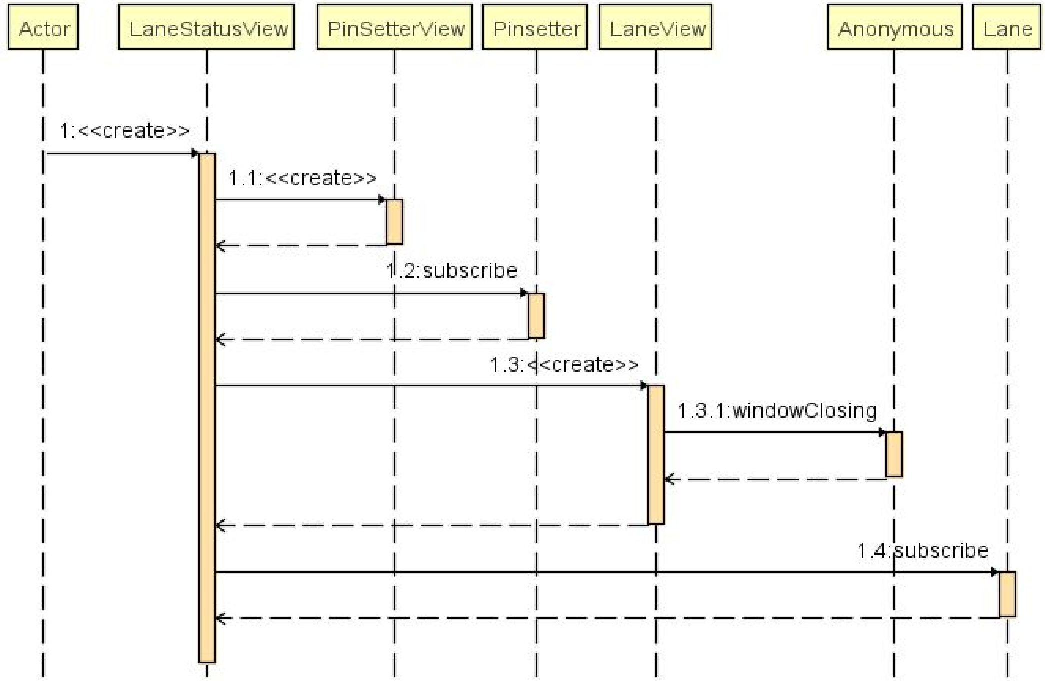
All classes

Below is the sequence diagram for the whole project.

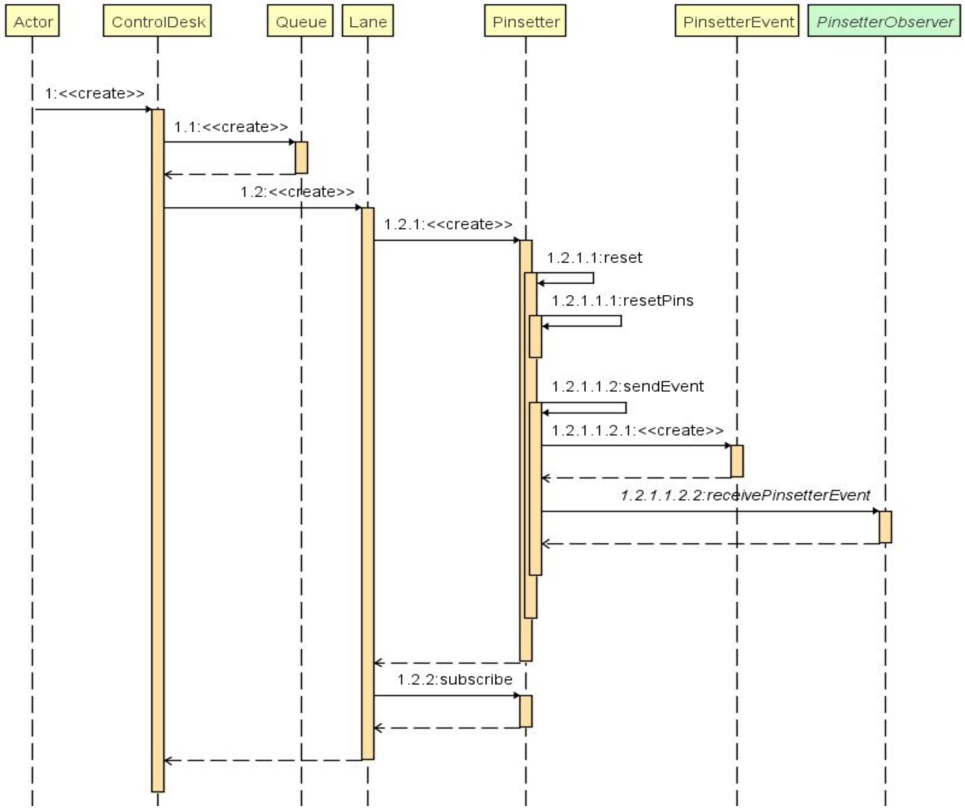


Similar to the previous section here also the whole code base is divided into some major parts and observed the sequence diagrams for them.

Lane Status View



Control Desk



CLASS RESPONSIBILITY TABLE

The major classes and their responsibilities have been listed along with the methods they call upon.

	Name	Methods	Responsibility
1	**AddPartyView**	AddPartyView actionPerformed valueChanged getNames getParty updateNewPatron	Handles the display to add a new party.
2	**Alley**	Alley getControlDesk	A outer container class, for the bowling simulation
3	**Bowler**	Bowler getNickName getFullName getNick getEmail equals	Stores the person’s nickname, full name and email for basic retrieval.
4	**BowlerFile**	getBowlerInfo putBowlerInfo getBowlers	Keeps Bowler information in a file for : ● Getting a vector of all bowlers Inserting information and searching for a bowler by nickname
5	**ControlDesk**	ControlDesk Run registerPatron assignLane addPartyQueue getPartyQueue getNumLanes Subscribe Publish getLanes	● Initializes input number of lanes ● Assigns lanes ● Creates a party of vectors of nicknames ● Maintains wait queue

6	**ControlDeskEvent**	ControlDeskEvent getPartyQueue	Maintain a vector of strings containing the names of the parties in the wait queue
7	**ControlDeskView**	ControlDiskView actionPerformed updateAddParty receiveControlDeskEvent	Displays the Control Desk View and controls all the possible inputs
8	**drive**	main	This is the main class which initializes the game, by setting parameters (such as number of lanes and maximum patrons per party) and then initializing all the necessary objects.
9	**EndGamePrompt**	EndGamePrompt actionPerformed getResultdestroy	Display the prompt that shows up at the end of the game and handle all the different inputs to the prompt.
10	**EndGameReport**	EndGameReport actionPerformed	Display the report at the end of the game as per the user's request.

ORIGINAL DESIGN ANALYSIS

Weaknesses

The original code, design and documentation exhibited poor software engineering design principles as well as various antipatterns.

Antipatterns fixed

Lava Flow

	Description Of Weakness	**Fixed**
1	There was large commented-out code with no explanations	Removed dead code and gained a full understanding of any bugs introduced.
2	Lot's of "To Do" code	Wrote the code wherever necessary.

Cut-And-Paste Programming

	Description Of Weakness	**Fixed**
1	Duplicate Code in many .java files	Refactored and eliminated using the DRY principle

Golden Hammer

	Description Of Weakness	**Fixed**
1	Several deprecated functions such as show(), hide(), new Integer() used throughout the code.	Development via linting tools such as Intellij helped fix this issue.

Functional Decomposition

	Description Of Weakness	**Fixed**
	Classes with a single method in use, no real use of the class data structure	Fixed by moving into existing class and/or combining classes

The Blob

	Description Of Weakness	**Fixed**
1	Single controller class, multiple simple data classes seen throughout implementation in files such as Lane.java and ControllerView.java	Split into appropriate classes wherever necessary

Strengths

- One of the biggest strengths of the original design is the adherence it has to MVC. The model classes are highly independent. The control classes communicate between view and model classes as expected.
- The coupling amongst all the classes has been kept low.
- The size of the code files has been kept low and optimum.
- A strong cohesion amongst related classes exists.

Fidelity to Design Documentation

- The pinsetter interface communicates to the scoring station the pins that are left standing after a bowler has completed a throw.
- The original code has for the most part abided to the design documented listing the features it needs to be implemented.
- The control desk operator has the ability to monitor the scores of any active lane.
- A configurable display option will allow the operator to view the score of an individual scoring station or multiple scoring stations.
- A bowling alley is composed of a number of bowling lanes and parties bowl on these lanes
- This display feature is not seen on the window panel and hence brings down the fidelity to design.
- Each lane is equipped with a stand-alone scoring station that lists the bowlers' names and a graphic representation of their scores.

CODE SMELLS

Code Smells Within Classes

1. Conditional Complexity

	Files Involved	Problem	Action Taken
1	LaneStatusView.java	Redundant nested if statements	Nested if statements combined

2. Long Methods/ Parameters And Duplication

	Files Involved	Problem	Action Taken
1	All .java files	Many files with redundant code	Long methods refactored. Parameters excluded wherever necessary using DRY method.

3. Comments

	Files Involved	Problem	Action Taken
1	All .java files	Several Javadoc errors in terms of syntax and formatting	All comments corrected and new comments explaining the purpose of the code.

4. Combinatorial Explosion

	Files Involved	Problem	Action Taken
1	All .java files	Lots of code that does almost the same thing.. but with tiny variations in data or behavior.	Base classes made to inherit the functionality from using parameters to carry out the modification.

5. Large Classes

	Files Involved	Problem	Action Taken
1	Lane.java	Large class used by all subclasses to inherit from.	Restructured into a smaller class with inheritance only where required.
2	ControlDesk.java	Large class used by all subclasses to inherit from.	Decided to fix the ControLDesk Observer/ Observable pattern as well, for consistency. Removed the ControlDeskEvent.java file, now just passing the ControlDesk itself to Observers.

6. Uncommunicative/Inconsistent Name

	Files Involved	Problem	Action Taken

1	All .java files	Random non-descriptive single letter naming seen in many files.	Used standard terminology and stuck to it throughout the methods

7. Speculative Generality

	Files Involved	Problem	Action Taken
1	EndGamePrompt.java NewPatronView.java	Same line declaration for multiple variables.	Used one line for each declaration, it enhances code readability.

8. Dead Code

	Files Involved	Problem	Action Taken
1	LaneEvent.java	An empty statement (semicolon) not part of a loop.	Removed semicolon
2	LaneServer.java	An empty statement (semicolon) not part of a loop.	Removed semicolon
3	LaneStatusView.java	Commented code statements.	Removed multiple sections of commented code.
4	LaneStatusView.java LaneView.java	Deprecated function new Integer()	Changed to updated newer functionality Integer.valueOf()

5	LaneView.java
	NewPatronView.java
		Deprecated function
		
		show()
		hide()
			Changed to updated newer functionality
			setVisibility(true)
			setVisibility(false)

Code Smells Between Classes

9. Data Class

	Files Involved	Problem	Action Taken

1	LaneStatusView.java	Classe that passively stores only data.	Added a few getters within Lane so that LaneStatusView is able to generate the end-of-game routine originally found in Lane

10. Alternative Classes With Different Interfaces

	Files Involved	Problem	Action Taken
1	Lane.java	Two classes are similar on the inside, but different on the outside, they can be modified to share a common interface	Removed the implements LaneObserver class in the GUI elements, replaced with implements Observer. Changed the logic for adding an Observer to Lane's list of observers to be in line with the java implementation in the appropriate GUI classes.
2	ControlDesk.java	Two classes are similar on the inside, but different on the outside, they can be modified to share a common interface	Changed ControlDesk from extending Thread to extending Observable, and it now implements Runnable.

11. Inappropriate Intimacy

	Files Involved	Problem	Action Taken
1	BowlerFile.java ControlDesk.java	Classes that spend too much time together, or classes that interface in inappropriate ways.	Changed the class visibility of BowlerFile.java and ControlDesk.java so that the GUI components would have access to them being from a different package.

12. Data Clumps

	Classes Involved	Problem	Action Taken

1	package ViewControl	Same data hanging around together, then verify whether it belongs together.	Moved all of the View/Control elements of the program to a new ViewControl package.
2	package Model	Same data hanging around together, then verify whether it belongs together.	Moved all of the Model elements of the program to a new Model package.

13. Indecent Exposure

	Files Involved	Problem	Action Taken

1	LaneEvent.java	Data items that were unnecessarily public.
		HashMap score;
		int frameNum;
			Used explicit scoping instead of accidental usage of default package private level

2	LaneStatusView.java	Data items that were unnecessarily public. JLabel foul;	Replaced by a local variable.
3	LaneStatusView.java	Data items that were unnecessarily public. Boolean laneShowing;	Used explicit scoping instead of accidental usage of default package private level

4	LaneView.java	Data items that were unnecessarily public.
		Container cpanel;
		JPanel scores;
		JLabel scoreLabel;
		JButton maintenance;
			Used explicit scoping instead of accidental usage of default package private level

14. Refused Bequest

	Files Involved	Problem	Action Taken

1	Lane.java	Inherit from a class but never use any of the inherited functionality	In Lane, got rid of the recievePinsetterEvent method and added the standard Observer.update method. Decided to keep the PinsetterEvent since it encapsulates data and does not inherit from Pinsetter.

15. Feature Envy

	Files Involved	Problem	Action Taken
1	Lane.java	Methods that make extensive use of another class.	The Lane.java file had some GUI code in it that needed migrating to the correct classes.

16. Lazy Class

	Files Involved	Problem	Action Taken
1	PinSetterView.java	Classes should pull their weight	Changed the constructor of the PinSetterView.java class to accept a Pinsetter so that it has the responsibility of adding itself as an observer.
2	Pinsetter.java	If a class isn't doing enough to pay for itself, it should be collapsed or combined into another class.	Removed the PinsetterObserver class.
3	drive.java	If a class isn't doing enough to pay for itself, it should be collapsed or combined into another class.	Removed the redundant alley class. Instead, Drive simply creates the ControlDesk directly and an entire class is able to be removed. These changes are clear in drive.java.

17. Shotgun Surgery

	Files Involved	Problem	Action Taken
1	Lane.java	A score change in one class requires cascading changes in several related classes.	Added a publish() call in Lane, once it is determined the game is finished. This allows the GUI to finalize the score display one last time.

18. Message Chains

	Files Involved	Problem	Action Taken
1	Lane.java	Long sequences of method calls to get routine data.	Deleted the redundant lanePublish() method within Lane.java. Replaced the publish() method with the proper Observable notify operations within Lane.java.

19. Middle Man

	Files Involved	Problem	Action Taken
1	LaneEvent.java	Class that is merely wrappers over other classes or existing functionality in the framework	Removed the LaneEvent imports in the GUI.

20. Divergent Change

	Files Involved	Problem	Action Taken
1	LaneStatusView.java LaneView.java	Changes to a class that touch completely different parts of the class.	Fixed the issues with migrating the receiveLaneEvent code to the update method in LaneStatusView.java.

			Fixed the issues with migrating the receiveLaneEvent code to the update method in LaneView.java.

21. Solution Sprawl

	Classes Involved	Problem	Action Taken
1	ScoreCalculator class	Simplifying and consolidating your design to ensure no more than one class is needed to make	Added the ScoreCalculator class and migrated all of the getScore functionality originally belonging to main into that class.

REFACTORED DESIGN

Balance Among Competing Criteria

1. Low Coupling

Tightly coupled systems tend to exhibit the following characteristics:

- A change in a class usually forces a ripple effect of changes in other classes.
- Require more effort and/or time due to the increased dependency.
- Might be harder to reuse a class because dependent classes must be included.

Hence, the coupling was reduced as much as possible.

Code Example:

Transferred the end-of-game code in Lane to LaneStatusView, decoupling the Model from the GUI.

2. High Cohesion

High cohesion tends to be preferable, because high cohesion is associated with several desirable traits of software including robustness, reliability, reusability, and understandability. In contrast, low cohesion is associated with undesirable traits such as being difficult to maintain, test, reuse, or even understand.

Code Example:

Replaced the extended Thread call in Lane to an implemented Runnable. Then in the constructor, created a new thread object, passing itself (as a Runnable object) as a parameter), and then started it. This effectively allows for us to extend the Observable class in Lane.

3. Separation Of Concerns

   Code Example:

   Moved the drive.java file outside of the ViewControl and Model packages and added it to the main package. Hence the code is split into three main packages which address 3 different issues.

4. Information Hiding

   Code Example: Effective use of classes.

5. Law Of Demeter

   It states that-

   - Each unit should have only limited knowledge about other units: only units "closely" related to the current unit.
   - Each unit should only talk to its friends; don't talk to strangers.
   - Only talk to your immediate friends.

   Code Example:

   Separation of the Lane and Observer functionality classes.

   Removed the implements LaneObserver class in the GUI elements, replaced with implements Observer. Changed the logic for adding an Observer to Lane's list of observers to be in line with the java implementation in the appropriate GUI classes.

OVERVIEW OF REFACTORING

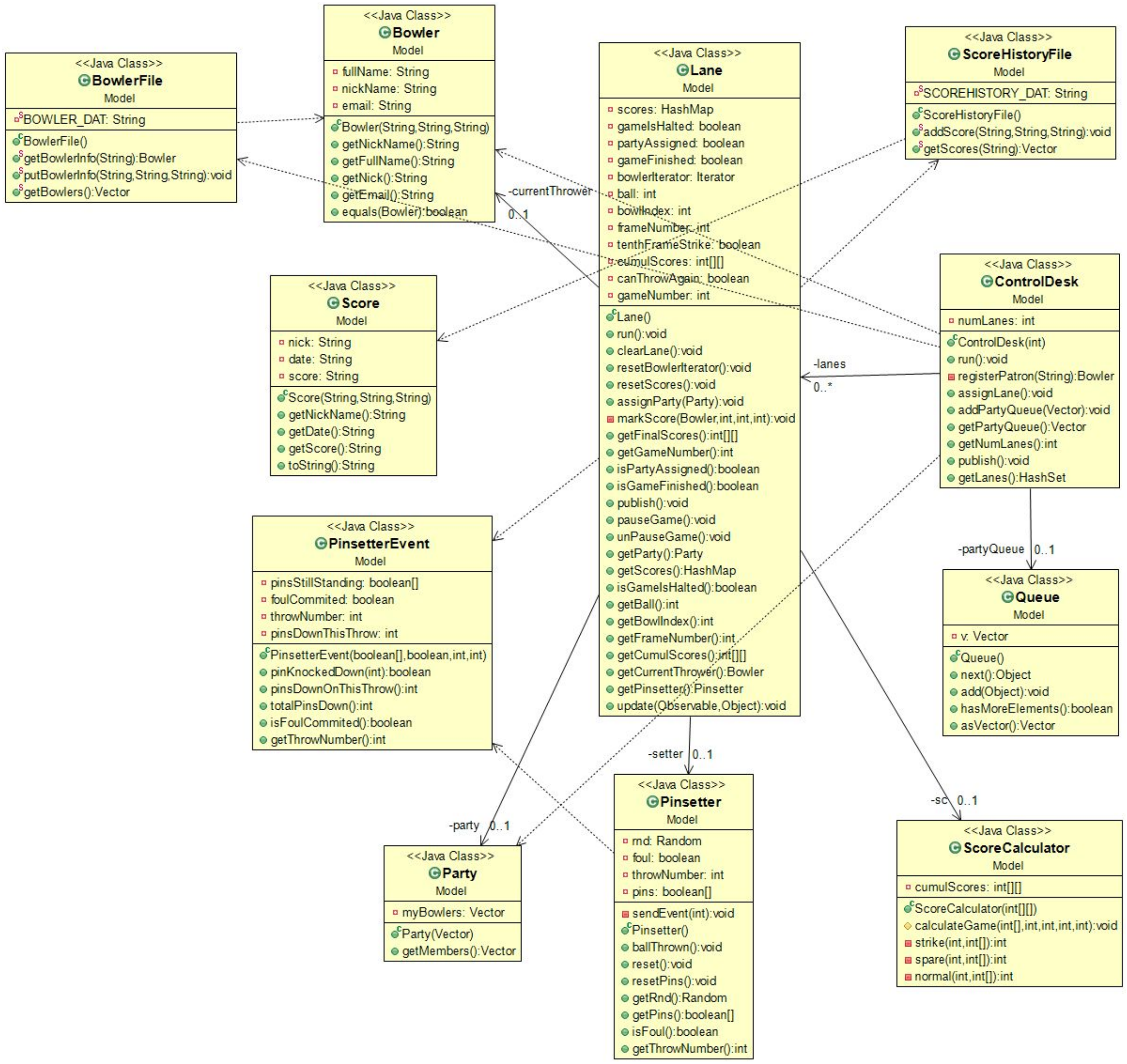
Sometimes code refactoring can take too much time as the code base may be very complex. So, it should be checked if refactoring the code actually benefits with respect to time consumed or not. It might be a better option to rewrite the code.

UML CLASS DIAGRAMS (REFACTORED DESIGN)

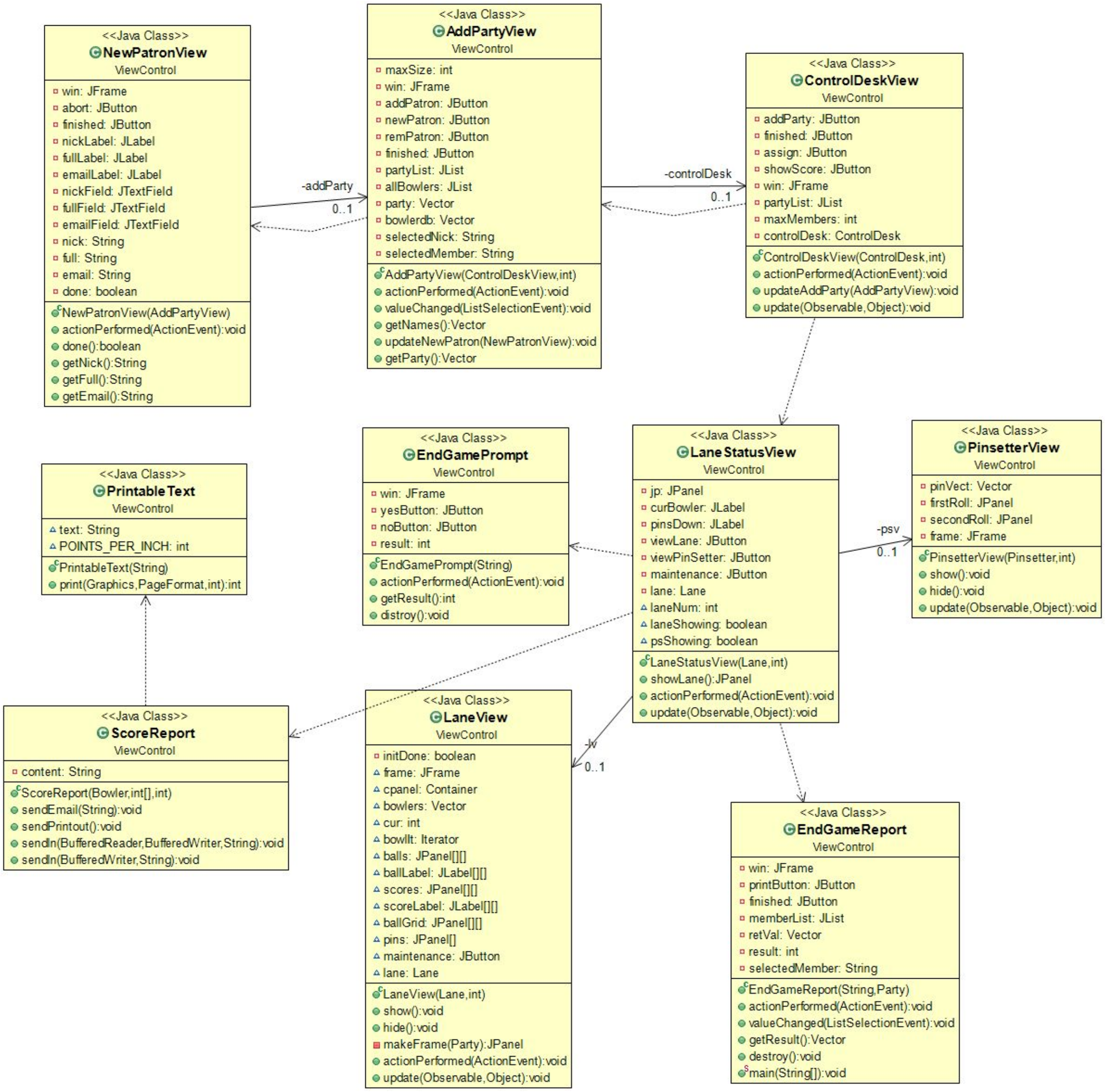
Interfaces, inheritance, generalization, association, aggregation, composition, cardinality and role indicators are indicated using appropriate arrows.

There are two major modules : **Models module** and **View Control module**.

Models Module

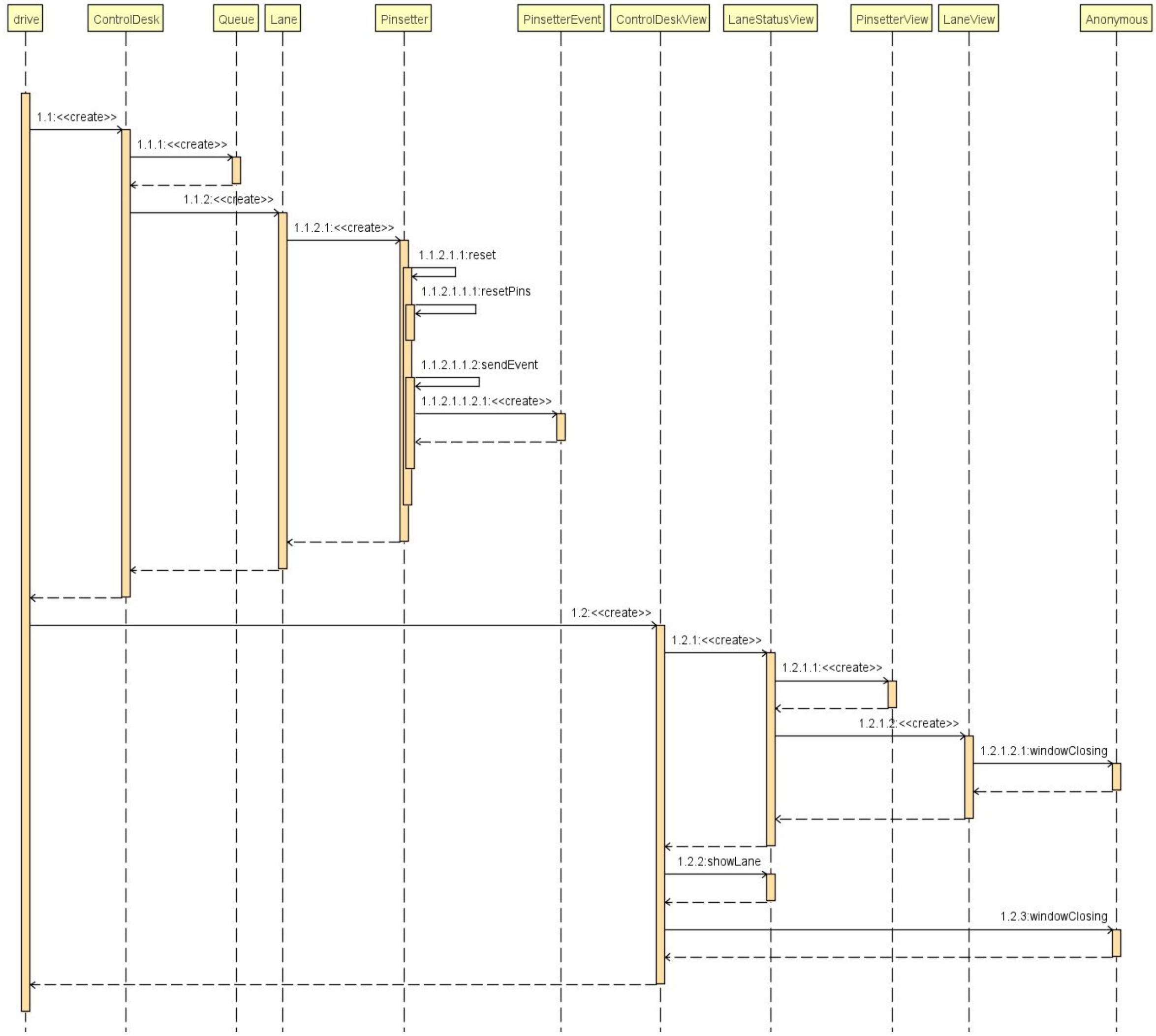


View Control Module

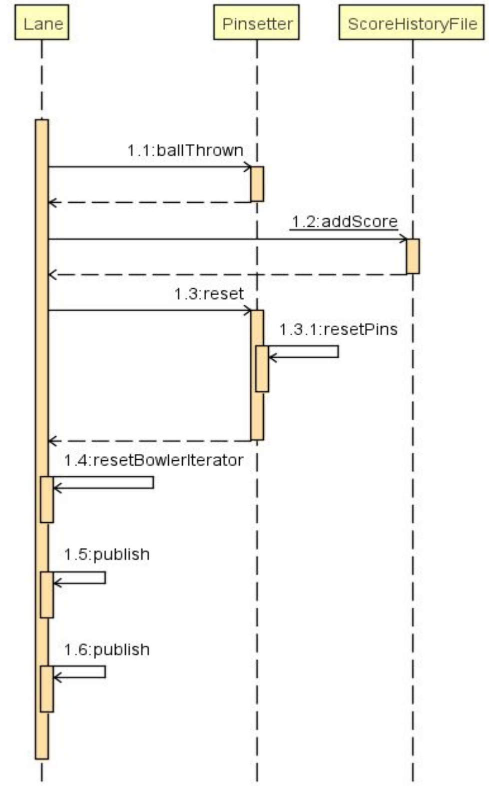


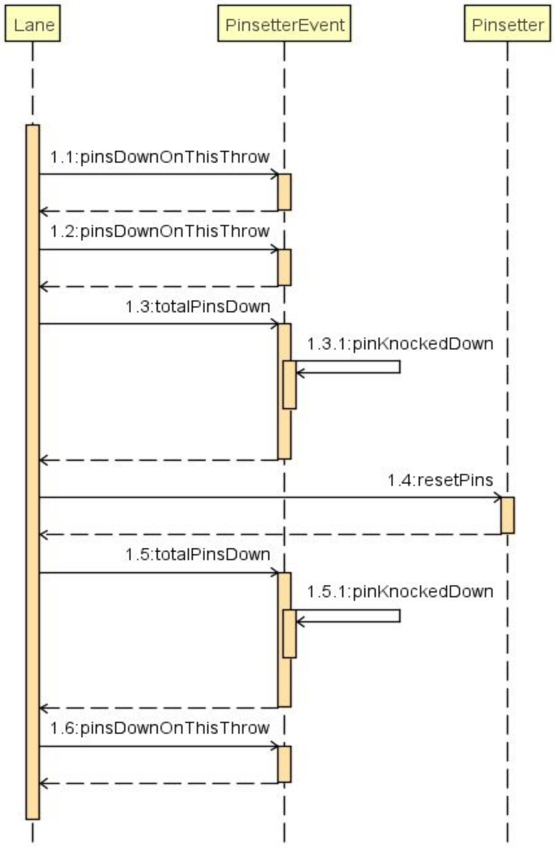
UML SEQUENCE DIAGRAMS (REFACTORED DESIGN)

All classes



Lane classes





CLASS RESPONSIBILITY TABLE (REFACTORED)

New Classes	Responsibilities
ScoreCalculated	To universally store the calculated score
NewPatronView	To get the view of different windows in the GUI
Observer	A general observer class that can be called upon to observe Lane and other entities

DESIGN PATTERNS ADOPTED

Adapter Pattern

- It is often used to make existing classes work with others without modifying their source code by allowing the interface of an existing class to be used as another interface
- For example, **ScoreCalculator class**: All the functionalities of getScore that belonged to the main class were migrated into the ScoreCalculator class.

Proxy Pattern

- Proxy is a structural design pattern that lets you provide a substitute or placeholder for another object. A proxy controls access to the original object, allowing you to perform something either before or after the request gets through to the original object.
- For example, **Lane.java**: To make LaneStatusView able to generate the end-of-game routine that was originally present in Lane.

Singleton Pattern

- The singleton pattern is a software design pattern that restricts the instantiation of a class to one "single" instance.
- For example, the **drive.java** class that acts as the main function of this game is instantiated only once in its entire lifetime.

Controller Facade Pattern

- The controller routes the request to your "model" and interacts with the View. The what is the same (route a simple interface to a more complex set behind the scenes). A controller in an application acts just like a Facade does in that it acts as a "gateway" to a more complex application for a specific use-case.

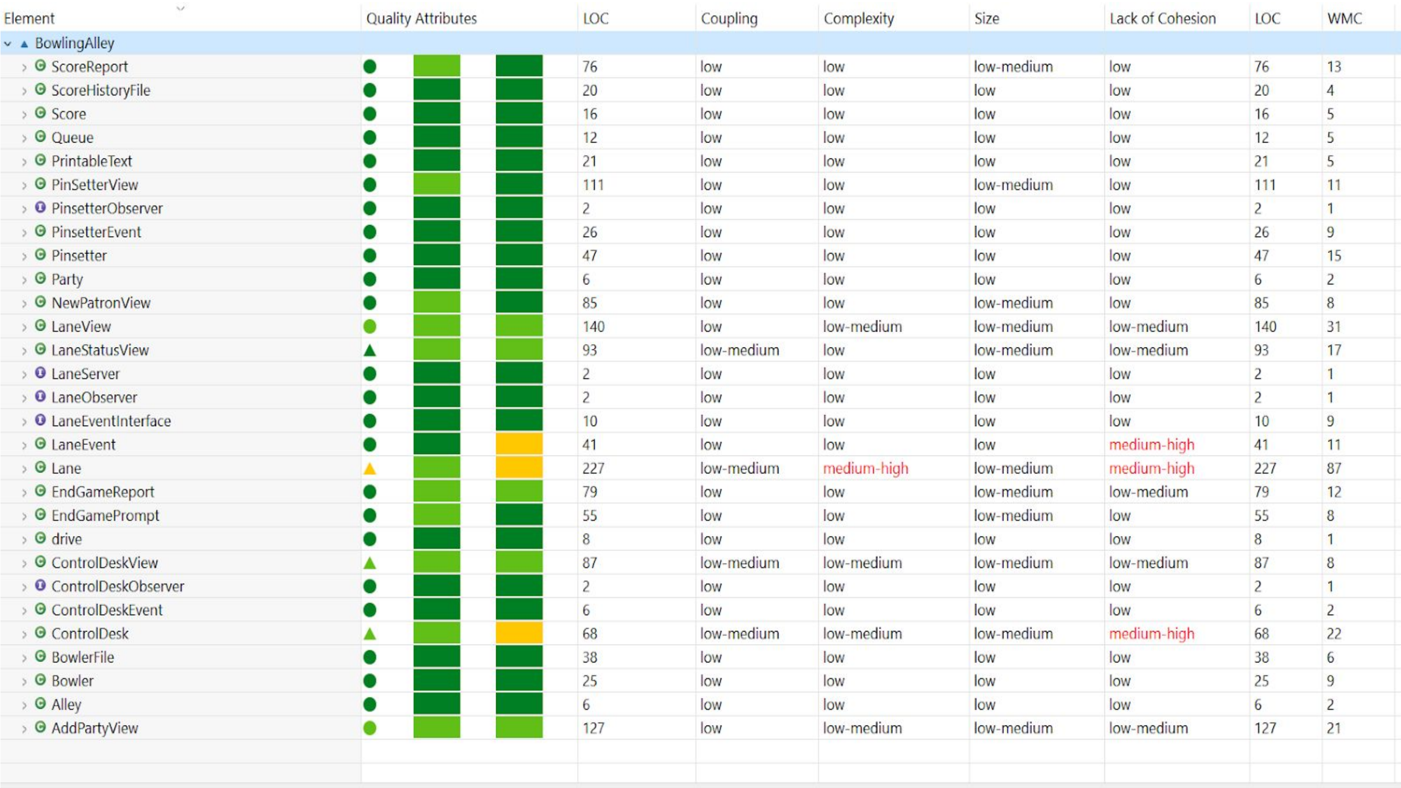
- Code Base: The code was split into Model and ViewControl packages.

METRICS ANALYSIS AND COMPARISON

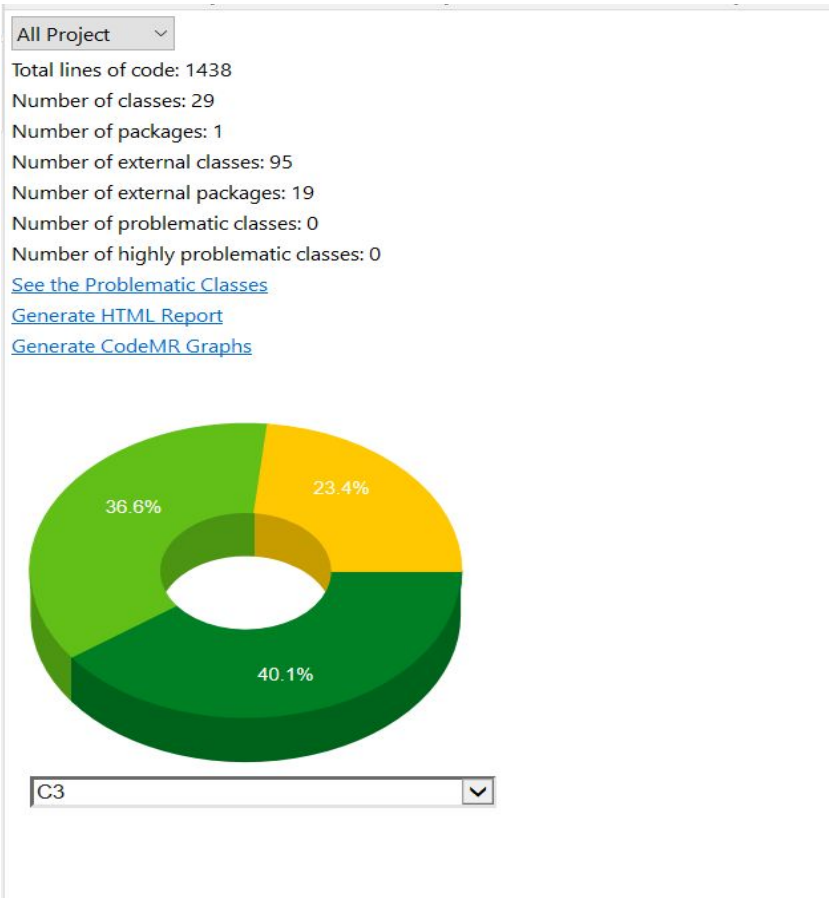
There are different metrics to compare the code base before and after modification. So, both the analyses are shown below in order to see the improvement.

Initial Analysis

There are different tools/plugins to measure code base. Here CodeMR is used to measure the code base. Below is the table generated by this tool.



Element	Quality Attributes	LOC	Coupling	Complexity	Size	Lack of Cohesion	LOC	WMC
BowlingAlley								
ScoreReport	● 🟩 🟩	76	low	low	low-medium	low	76	13
ScoreHistoryFile	● 🟩 🟩	20	low	low	low	low	20	4
Score	● 🟩 🟩	16	low	low	low	low	16	5
Queue	● 🟩 🟩	12	low	low	low	low	12	5
PrintableText	● 🟩 🟩	21	low	low	low	low	21	5
PinSetterView	● 🟩 🟩	111	low	low	low-medium	low	111	11
PinsetterObserver	● 🟩 🟩	2	low	low	low	low	2	1
PinsetterEvent	● 🟩 🟩	26	low	low	low	low	26	9
Pinsetter	● 🟩 🟩	47	low	low	low	low	47	15
Party	● 🟩 🟩	6	low	low	low	low	6	2
NewPatronView	● 🟩 🟩	85	low	low	low-medium	low	85	8
LaneView	● 🟩 🟩	140	low	low-medium	low-medium	low-medium	140	31
LaneStatusView	▲ 🟩 🟩	93	low-medium	low	low-medium	low-medium	93	17
LaneServer	● 🟩 🟩	2	low	low	low	low	2	1
LaneObserver	● 🟩 🟩	2	low	low	low	low	2	1
LaneEventInterface	● 🟩 🟩	10	low	low	low	low	10	9
LaneEvent	● 🟩 🟨	41	low	low	low	medium-high	41	11
Lane	▲ 🟩 🟨	227	low-medium	medium-high	low-medium	medium-high	227	87
EndGameReport	● 🟩 🟩	79	low	low	low-medium	low-medium	79	12
EndGamePrompt	● 🟩 🟩	55	low	low	low-medium	low	55	8
drive	● 🟩 🟩	8	low	low	low	low	8	1
ControlDeskView	▲ 🟩 🟩	87	low-medium	low-medium	low-medium	low-medium	87	8
ControlDeskObserver	● 🟩 🟩	2	low	low	low	low	2	1
ControlDeskEvent	● 🟩 🟩	6	low	low	low	low	6	2
ControlDesk	▲ 🟩 🟨	68	low-medium	low-medium	low-medium	medium-high	68	22
BowlerFile	● 🟩 🟩	38	low	low	low	low	38	6
Bowler	● 🟩 🟩	25	low	low	low	low	25	9
Alley	● 🟩 🟩	6	low	low	low	low	6	2
AddPartyView	● 🟩 🟩	127	low	low-medium	low-medium	low-medium	127	21



All Project

Total lines of code: 1438

Number of classes: 29

Number of packages: 1

Number of external classes: 95

Number of external packages: 19

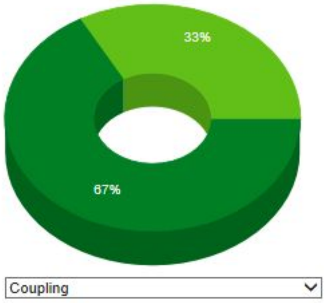
Number of problematic classes: 0

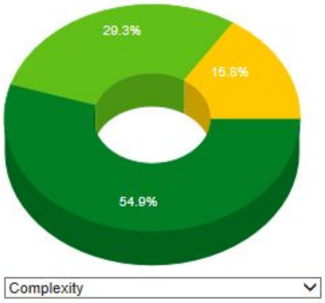
Number of highly problematic classes: 0

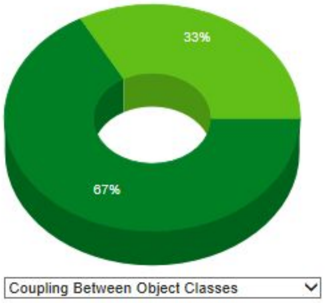
See the Problematic Classes

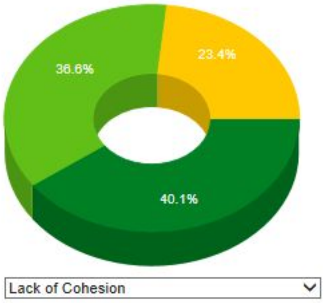
Generate HTML Report

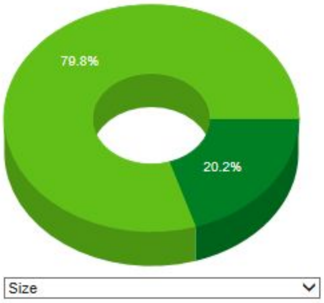
Generate CodeMR Graphs

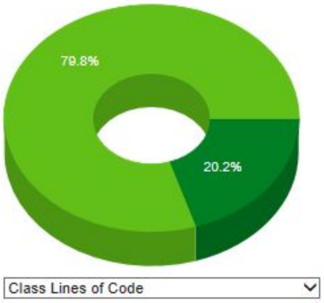




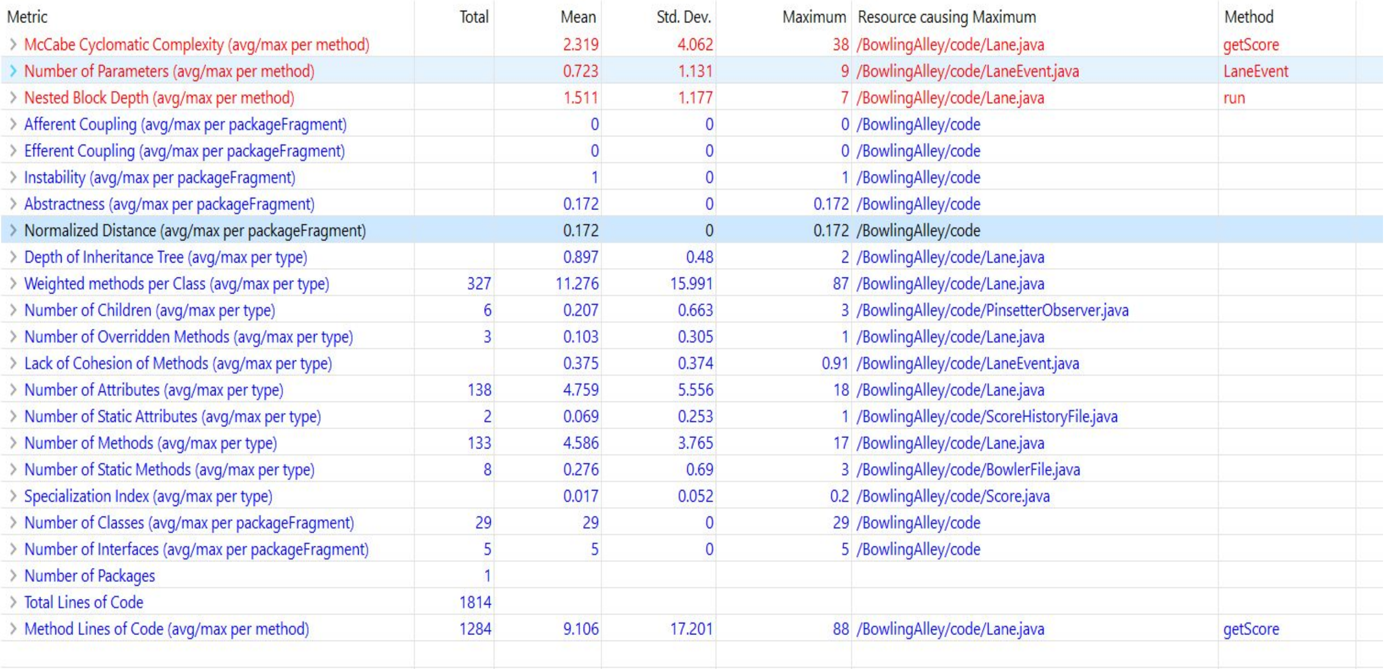








The initial metrics like cyclomatic complexity, no of parameters(avg/max per method) etc. are measured using the metrics plugin in eclipse ide is as follows :



Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/max per method)		2.319	4.062	38	/BowlingAlley/code/Lane.java	getScore
Number of Parameters (avg/max per method)		0.723	1.131	9	/BowlingAlley/code/LaneEvent.java	LaneEvent
Nested Block Depth (avg/max per method)		1.511	1.177	7	/BowlingAlley/code/Lane.java	run
Afferent Coupling (avg/max per packageFragment)		0	0	0	/BowlingAlley/code	
Efferent Coupling (avg/max per packageFragment)		0	0	0	/BowlingAlley/code	
Instability (avg/max per packageFragment)		1	0	1	/BowlingAlley/code	
Abstractness (avg/max per packageFragment)		0.172	0	0.172	/BowlingAlley/code	
Normalized Distance (avg/max per packageFragment)		0.172	0	0.172	/BowlingAlley/code	
Depth of Inheritance Tree (avg/max per type)		0.897	0.48	2	/BowlingAlley/code/Lane.java	
Weighted methods per Class (avg/max per type)	327	11.276	15.991	87	/BowlingAlley/code/Lane.java	
Number of Children (avg/max per type)	6	0.207	0.663	3	/BowlingAlley/code/PinsetterObserver.java	
Number of Overridden Methods (avg/max per type)	3	0.103	0.305	1	/BowlingAlley/code/Lane.java	
Lack of Cohesion of Methods (avg/max per type)		0.375	0.374	0.91	/BowlingAlley/code/LaneEvent.java	
Number of Attributes (avg/max per type)	138	4.759	5.556	18	/BowlingAlley/code/Lane.java	
Number of Static Attributes (avg/max per type)	2	0.069	0.253	1	/BowlingAlley/code/ScoreHistoryFile.java	
Number of Methods (avg/max per type)	133	4.586	3.765	17	/BowlingAlley/code/Lane.java	
Number of Static Methods (avg/max per type)	8	0.276	0.69	3	/BowlingAlley/code/BowlerFile.java	
Specialization Index (avg/max per type)		0.017	0.052	0.2	/BowlingAlley/code/Score.java	
Number of Classes (avg/max per packageFragment)	29	29	0	29	/BowlingAlley/code	
Number of Interfaces (avg/max per packageFragment)	5	5	0	5	/BowlingAlley/code	
Number of Packages	1					
Total Lines of Code	1814					
Method Lines of Code (avg/max per method)	1284	9.106	17.201	88	/BowlingAlley/code/Lane.java	getScore

Desired Measurements

After analysing the metrics shown above it is evident that the complexity, coupling and lack of cohesion in the files with large number of lines of code need to be reduced.

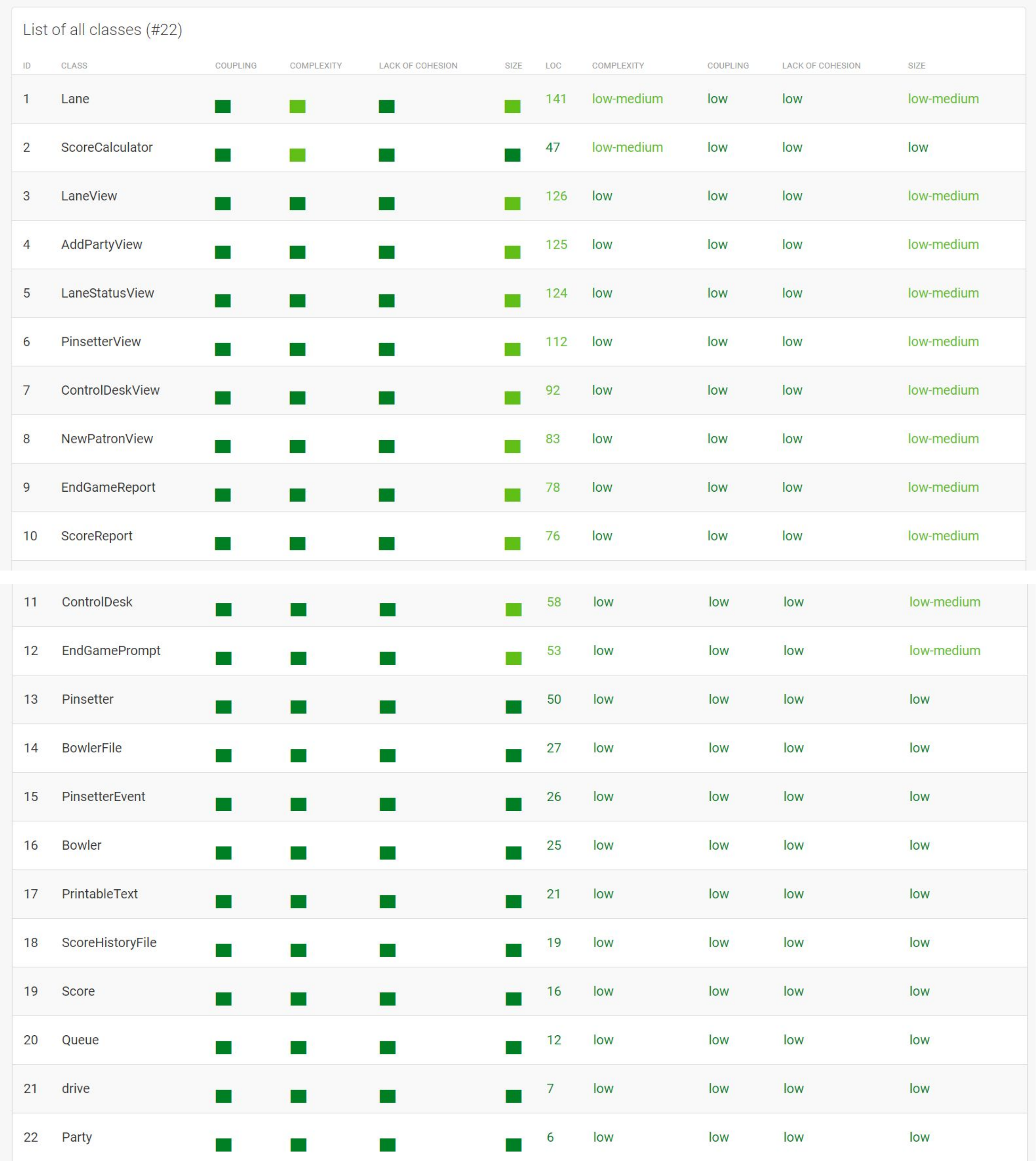
We applied several design patterns and removed code smells as documented above in order to achieve the following final results.

From the above analysis it was clear that many metrics were out of range. We tried to refactor the classes with large numbers of lines of code.

We also tried to reduce the total no of classes and successfully reduced the numbers of classes down to 22.

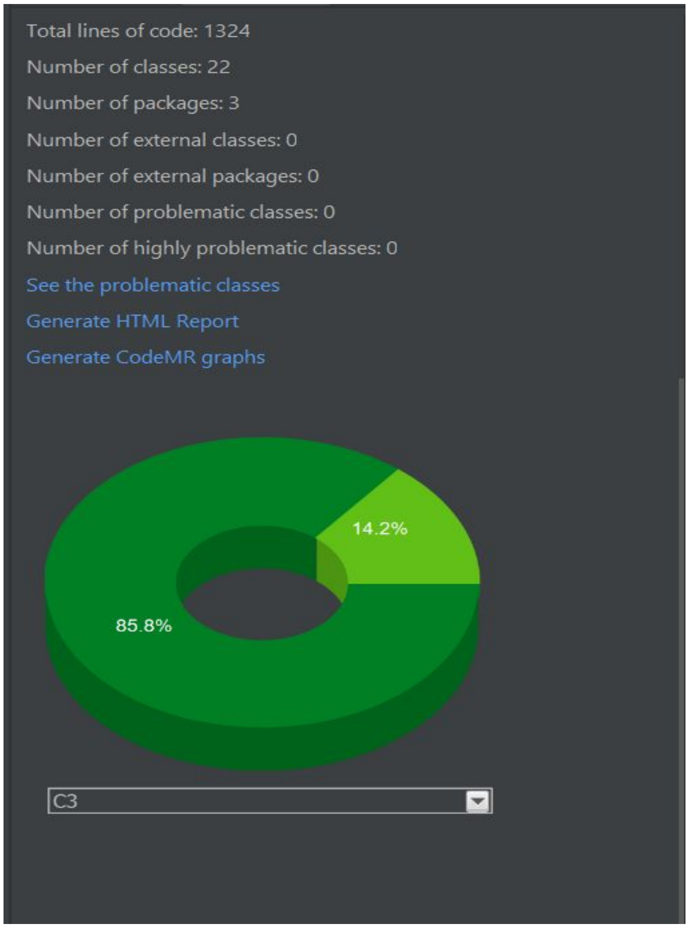
Final Analysis

After modification of the code base it can be seen that the coupling, lack of cohesion is reduced to low for each and every class. The complexity of every class is reduced from low to low-medium also the number of lines of code is reduced without affecting the functionality.



List of all classes (#22)

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	Lane	🟩	🟩	🟩	🟩	141	low-medium	low	low	low-medium
2	ScoreCalculator	🟩	🟩	🟩	🟩	47	low-medium	low	low	low
3	LaneView	🟩	🟩	🟩	🟩	126	low	low	low	low-medium
4	AddPartyView	🟩	🟩	🟩	🟩	125	low	low	low	low-medium
5	LaneStatusView	🟩	🟩	🟩	🟩	124	low	low	low	low-medium
6	PinsetterView	🟩	🟩	🟩	🟩	112	low	low	low	low-medium
7	ControlDeskView	🟩	🟩	🟩	🟩	92	low	low	low	low-medium
8	NewPatronView	🟩	🟩	🟩	🟩	83	low	low	low	low-medium
9	EndGameReport	🟩	🟩	🟩	🟩	78	low	low	low	low-medium
10	ScoreReport	🟩	🟩	🟩	🟩	76	low	low	low	low-medium
11	ControlDesk	🟩	🟩	🟩	🟩	58	low	low	low	low-medium
12	EndGamePrompt	🟩	🟩	🟩	🟩	53	low	low	low	low-medium
13	Pinsetter	🟩	🟩	🟩	🟩	50	low	low	low	low
14	BowlerFile	🟩	🟩	🟩	🟩	27	low	low	low	low
15	PinsetterEvent	🟩	🟩	🟩	🟩	26	low	low	low	low
16	Bowler	🟩	🟩	🟩	🟩	25	low	low	low	low
17	PrintableText	🟩	🟩	🟩	🟩	21	low	low	low	low
18	ScoreHistoryFile	🟩	🟩	🟩	🟩	19	low	low	low	low
19	Score	🟩	🟩	🟩	🟩	16	low	low	low	low
20	Queue	🟩	🟩	🟩	🟩	12	low	low	low	low
21	drive	🟩	🟩	🟩	🟩	7	low	low	low	low
22	Party	🟩	🟩	🟩	🟩	6	low	low	low	low



Total lines of code: 1324

Number of classes: 22

Number of packages: 3

Number of external classes: 0

Number of external packages: 0

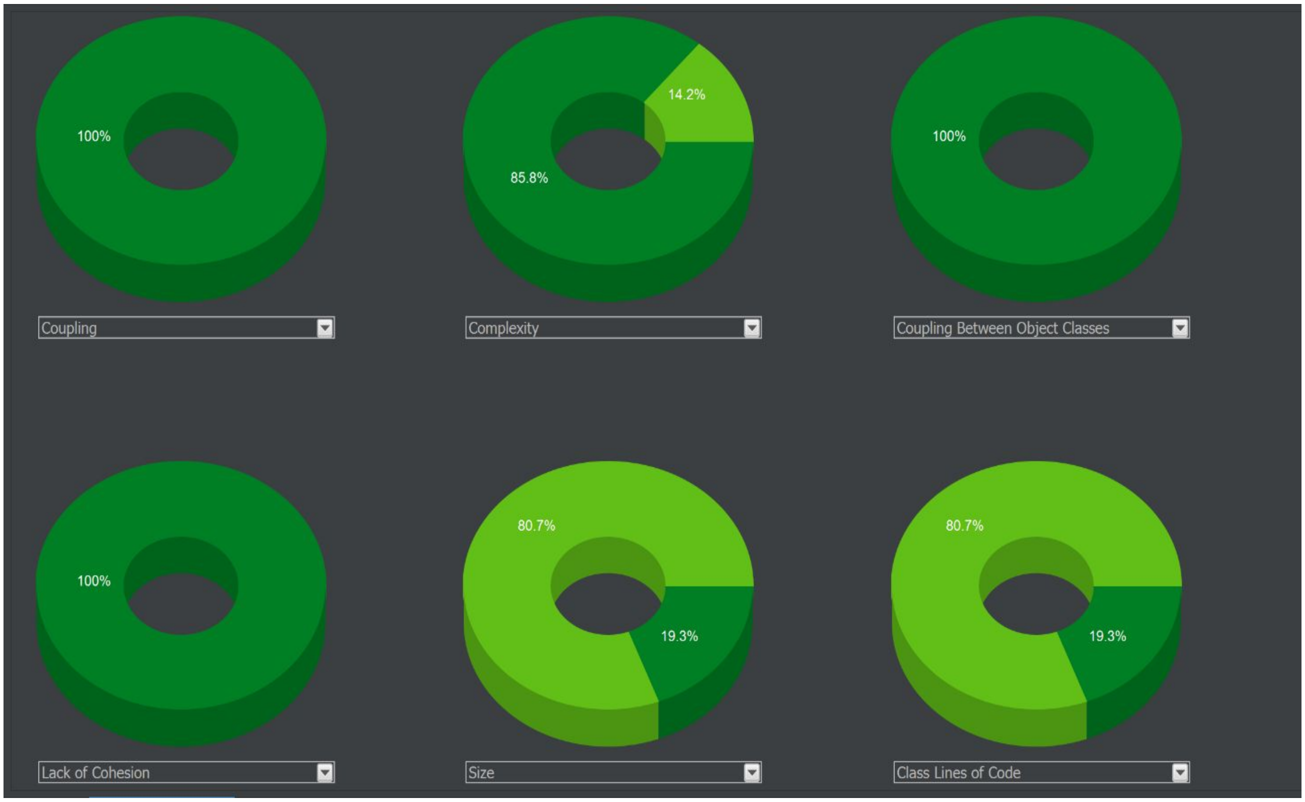
Number of problematic classes: 0

Number of highly problematic classes: 0

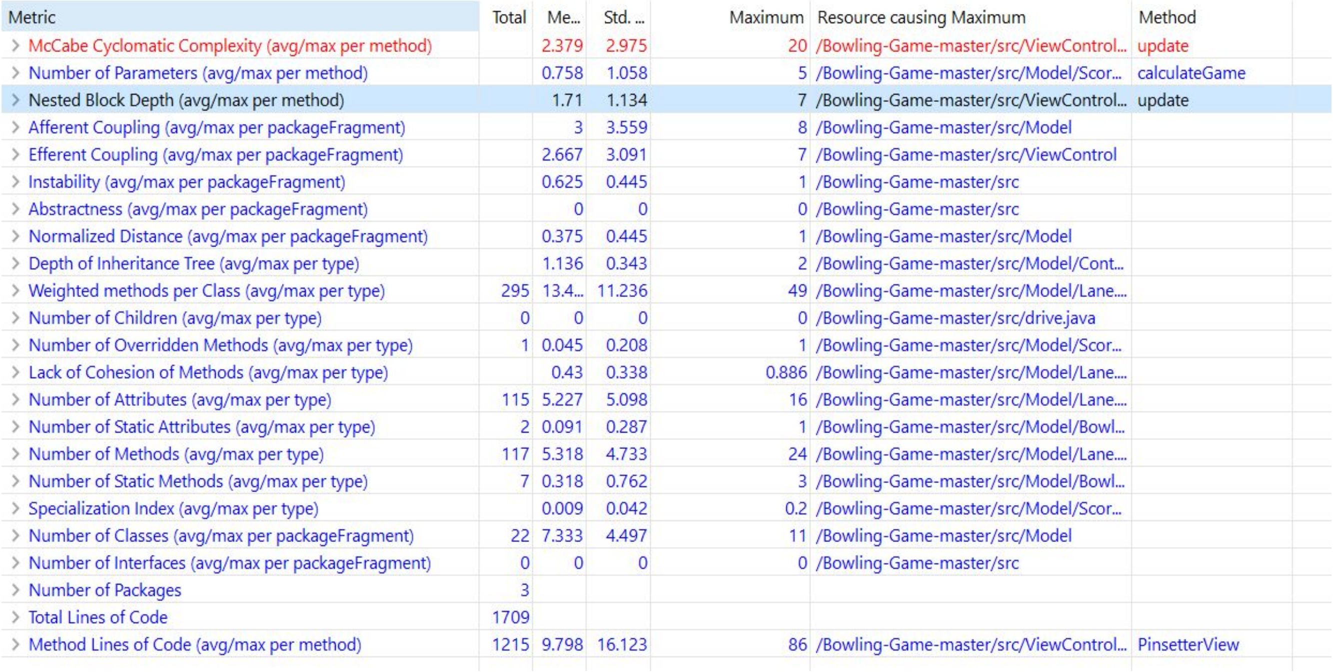
See the problematic classes

Generate HTML Report

Generate CodeMR graphs



The final metrics like cyclomatic complexity, no of parameters (avg/max per method) etc. are measured using the metrics plugin in eclipse ide is as follows :



Metric	Total	Me...	Std. ...	Maximum	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/max per method)		2.379	2.975	20	/Bowling-Game-master/src/ViewControl...	update
Number of Parameters (avg/max per method)		0.758	1.058	5	/Bowling-Game-master/src/Model/Scor...	calculateGame
Nested Block Depth (avg/max per method)		1.71	1.134	7	/Bowling-Game-master/src/ViewControl...	update
Afferent Coupling (avg/max per packageFragment)		3	3.559	8	/Bowling-Game-master/src/Model	
Efferent Coupling (avg/max per packageFragment)		2.667	3.091	7	/Bowling-Game-master/src/ViewControl	
Instability (avg/max per packageFragment)		0.625	0.445	1	/Bowling-Game-master/src	
Abstractness (avg/max per packageFragment)		0	0	0	/Bowling-Game-master/src	
Normalized Distance (avg/max per packageFragment)		0.375	0.445	1	/Bowling-Game-master/src/Model	
Depth of Inheritance Tree (avg/max per type)		1.136	0.343	2	/Bowling-Game-master/src/Model/Cont...	
Weighted methods per Class (avg/max per type)	295	13.4...	11.236	49	/Bowling-Game-master/src/Model/Lane....	
Number of Children (avg/max per type)	0	0	0	0	/Bowling-Game-master/src/drive.java	
Number of Overridden Methods (avg/max per type)	1	0.045	0.208	1	/Bowling-Game-master/src/Model/Scor...	
Lack of Cohesion of Methods (avg/max per type)		0.43	0.338	0.886	/Bowling-Game-master/src/Model/Lane....	
Number of Attributes (avg/max per type)	115	5.227	5.098	16	/Bowling-Game-master/src/Model/Lane....	
Number of Static Attributes (avg/max per type)	2	0.091	0.287	1	/Bowling-Game-master/src/Model/Bowl...	
Number of Methods (avg/max per type)	117	5.318	4.733	24	/Bowling-Game-master/src/Model/Lane....	
Number of Static Methods (avg/max per type)	7	0.318	0.762	3	/Bowling-Game-master/src/Model/Bowl...	
Specialization Index (avg/max per type)		0.009	0.042	0.2	/Bowling-Game-master/src/Model/Scor...	
Number of Classes (avg/max per packageFragment)	22	7.333	4.497	11	/Bowling-Game-master/src/Model	
Number of Interfaces (avg/max per packageFragment)	0	0	0	0	/Bowling-Game-master/src	
Number of Packages	3					
Total Lines of Code	1709					
Method Lines of Code (avg/max per method)	1215	9.798	16.123	86	/Bowling-Game-master/src/ViewControl...	PinsetterView

CONCLUSION

It can be seen that the code base has significantly improved in terms of code quality due to the refactoring. This has been achieved by creating a balance among competing criteria such as low coupling and high cohesion so as to develop efficient classes with no dead or duplicate code.