DASS ASSIGNMENT - 3

BOWLING ALLEY MANAGEMENT SYSTEM

DATE OF SUBMISSION - 9th April 2020

MEMBERS NAME	EFFORT HOURS	ROLE
Vipul Chhabra	40 HOURS 4-5 Hours Getting started with JAVA 1-2 Hours Setting up the development environment 2-3 HOURS Understanding the code 17-18 HOURS Refactoring of the code 6-7 HOURS Implementing the new features 5-6 HOURS Documenting the whole work	Refactored the code, added 2 new features, Report
Bhuvanesh Sridharan	25 HOURS 1-2 Hours Setting up the environment 2-3 HOURS Understanding the code 9-10 HOURS Refactoring of the code 9-10 HOURS Implementing the new features	Refactored and completed the Third feature

Kushagra Agarwal	25 hours 5 hours Getting info about metrics 10 hours Refactoring which affected the working of the game 10 hours UML Diagrams	UML Diagrams

Project Overview

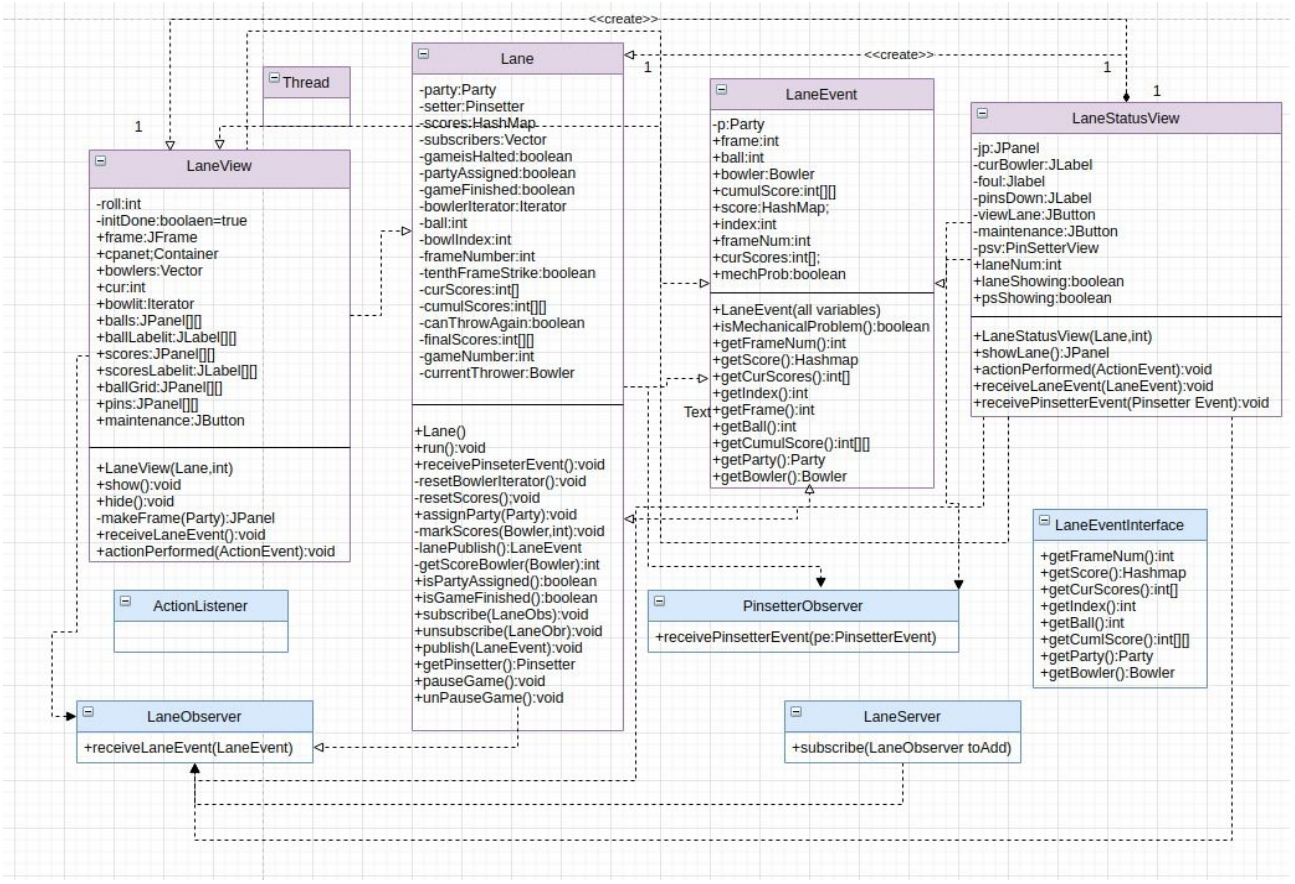
The project is useful for managing the bowling alley currently It simulates the bowling scenario and provides options for managing the Lanes for parties from Control Desk. The user can come and create a team of 5 persons. The vacant lane would be allotted to the team.

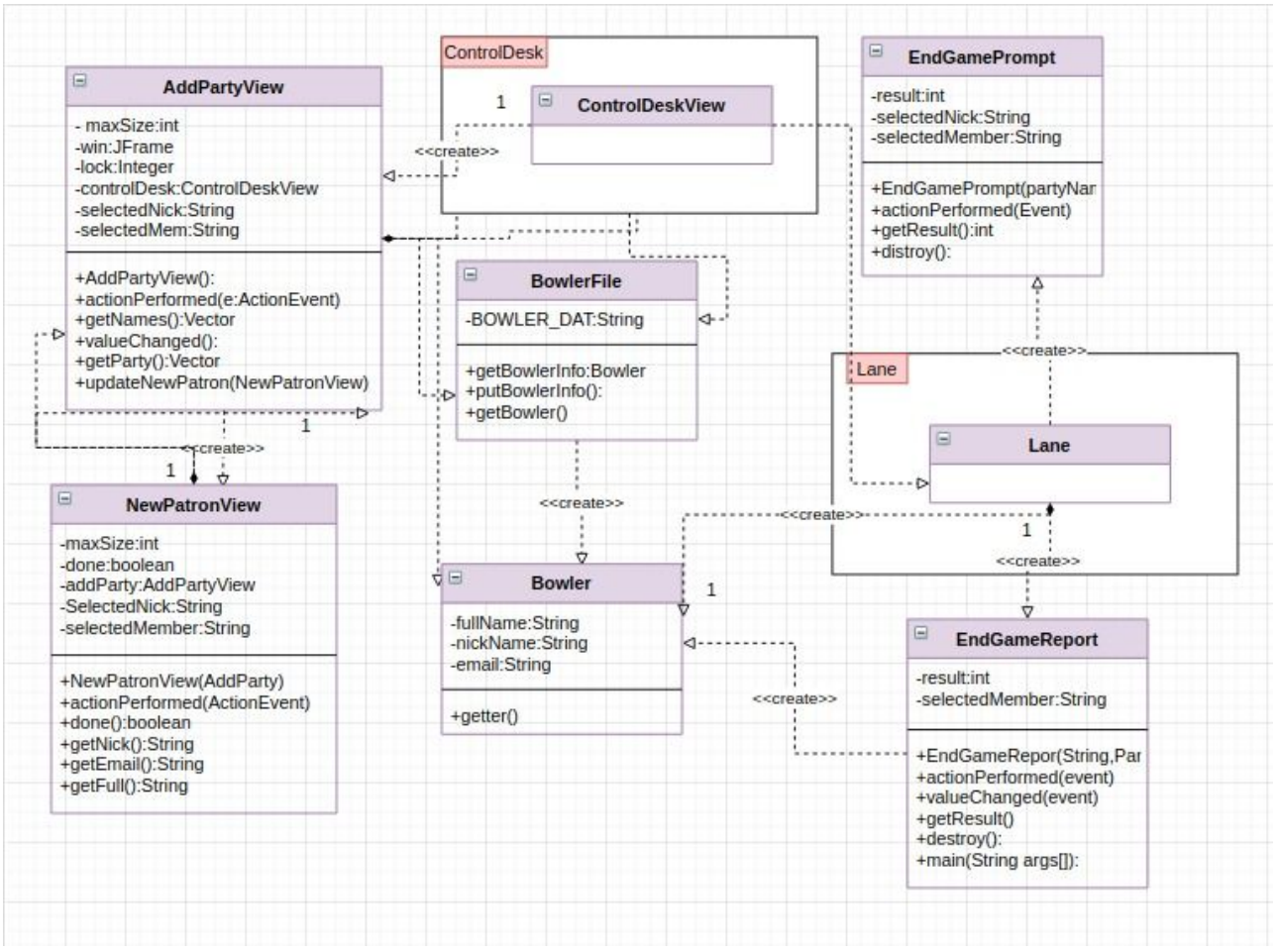
Features

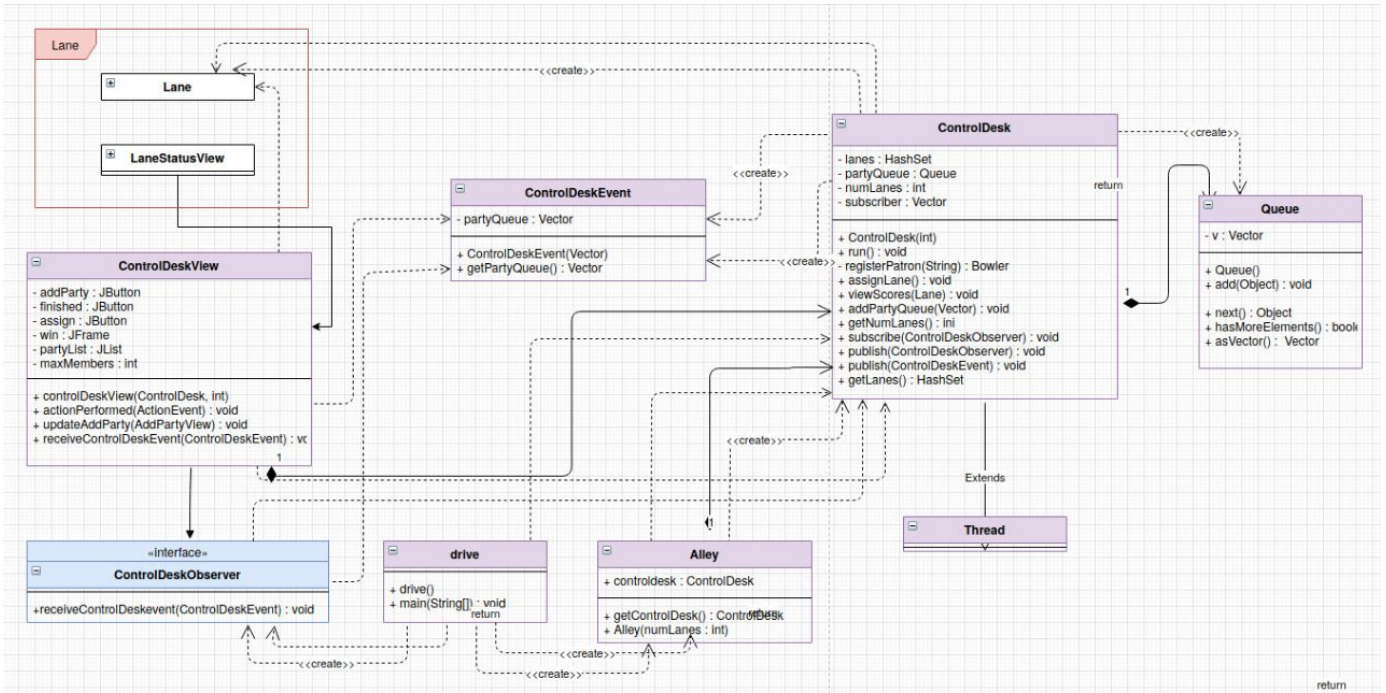
- Add new players in the database.
- Can observe the scoreboard for each lane.
- Can create teams and start a game in one of the lanes
- Can view the PinsetterView which is currently being simulated.
- Can pause the game for Maintenance and resume it back
- The report containing the previous score of each player can be viewed.
- **New -** More than 5 players can play the game at a time
- **New -** All the records of old scores can be viewed at the control desk and can be sorted in the desired order.
- **New -** The top players with their top scores can also be seen.
- **New -** The old game session can be stored
- New - The old game session can be loaded and the game can start from the same point.

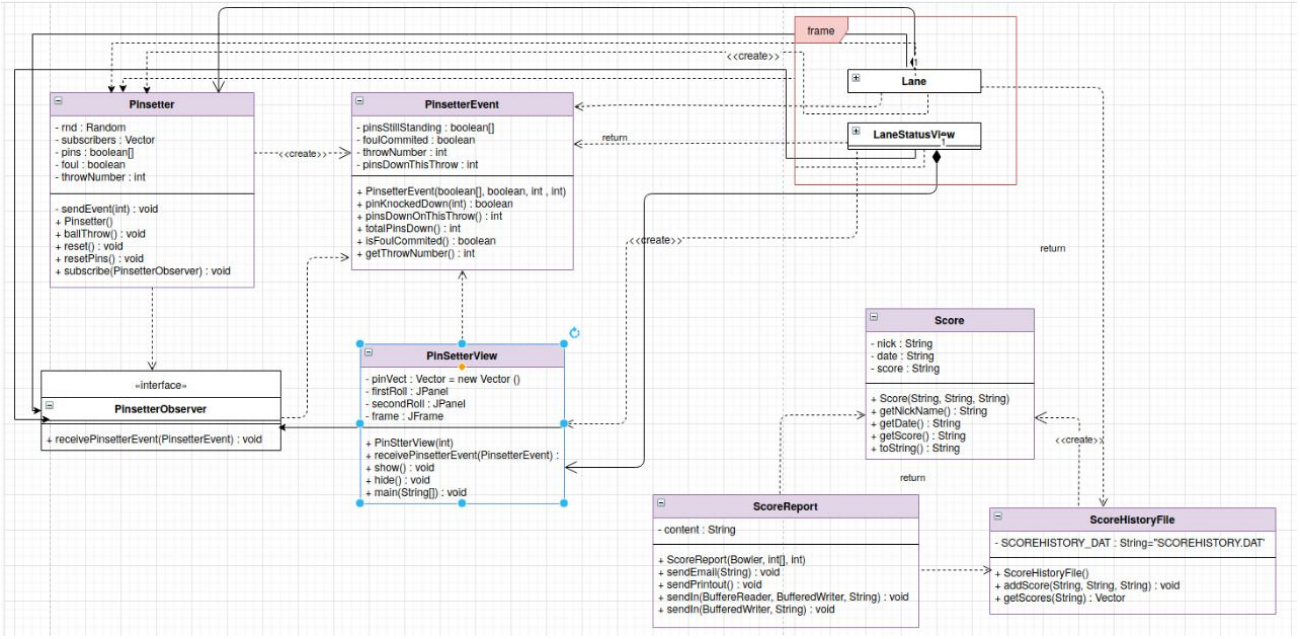
UML Class Diagrams

FOR OLD CODE

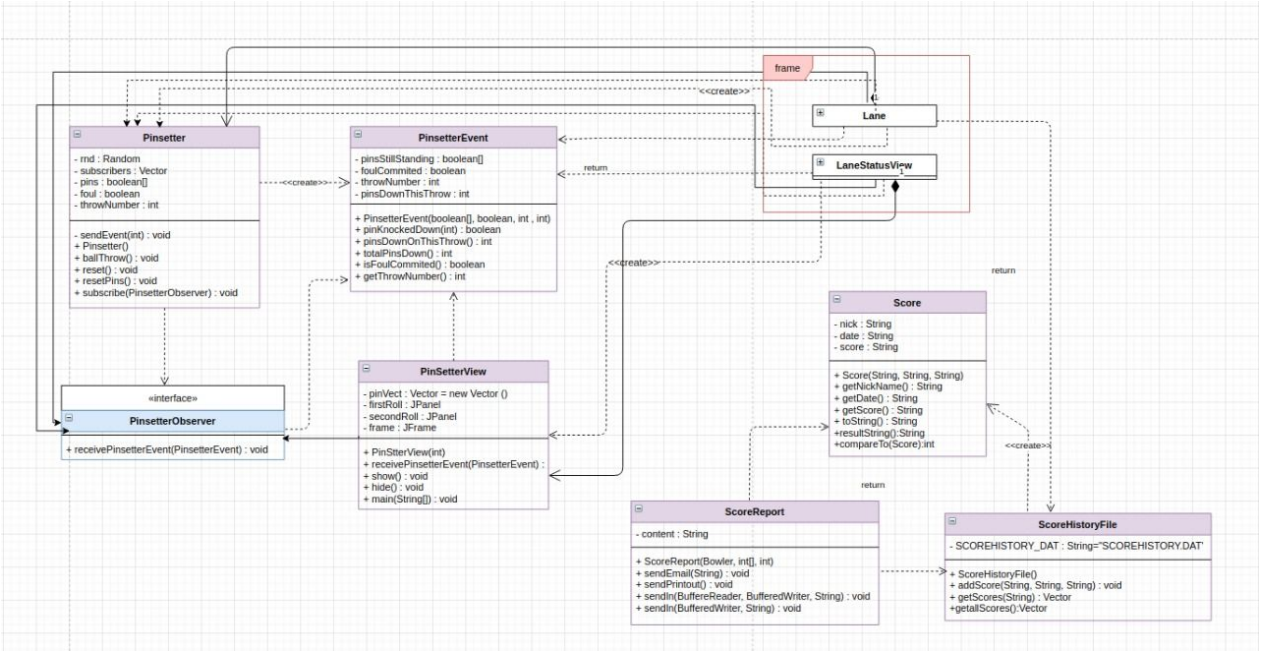


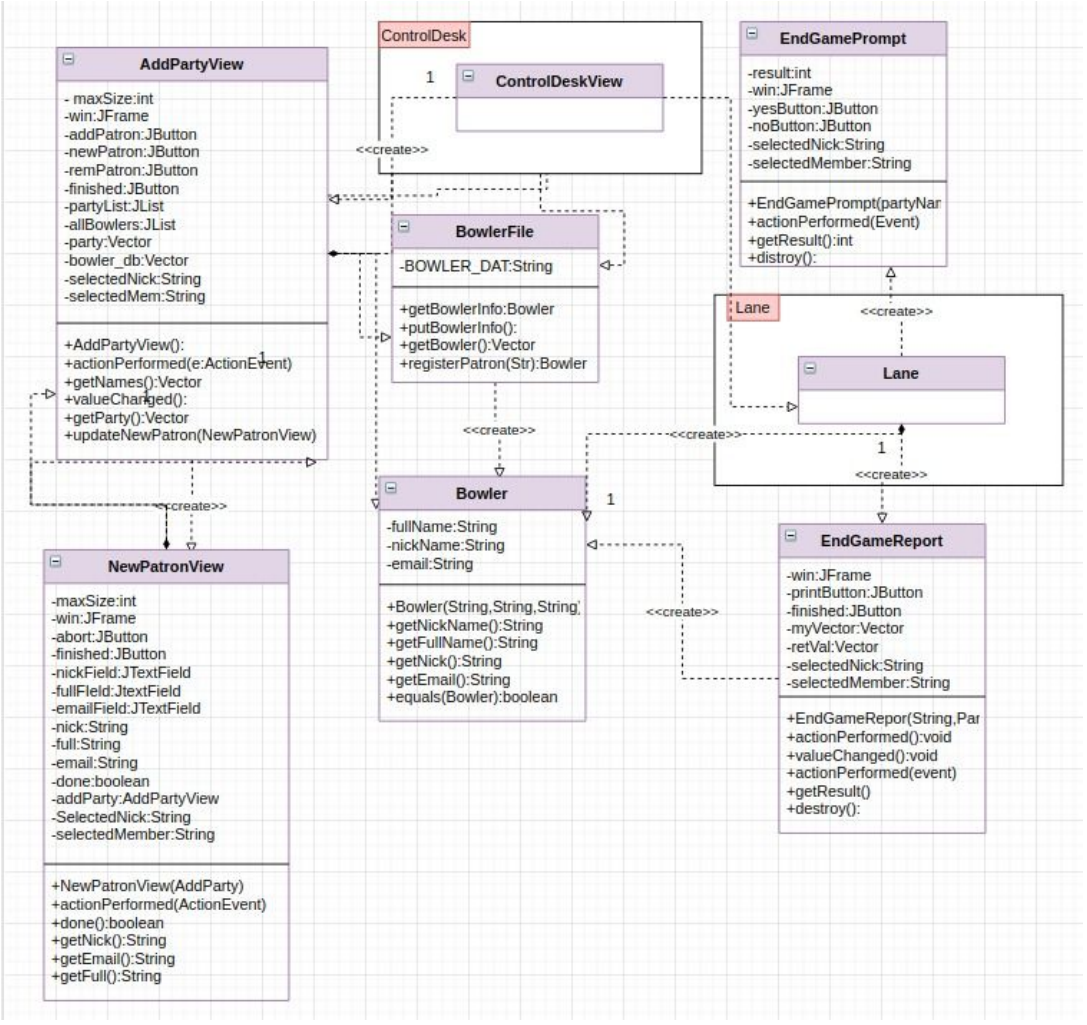

ControlDesk
AddPartyView
- maxSize:int
-win:JFrame
-lock:Integer
-controlDesk:ControlDeskView
-selectedNick:String
-selectedMem:String
+AddPartyView():
+actionPerformed(e:ActionEvent)
+getNames():Vector
+valueChanged():
+getParty():Vector
+updateNewPatron(NewPatronView)
1
<<create>>
ControlDeskView
EndGamePrompt
-result:int
-selectedNick:String
-selectedMember:String
+EndGamePrompt(partyNar
+actionPerformed(Event)
+getResult():int
+distroy():
BowlerFile
-BOWLER_DAT:String
+getBowlerInfo:Bowler
+putBowlerInfo():
+getBowler()
Lane
Lane
NewPatronView
-maxSize:int
-done:boolean
-addParty:AddPartyView
-SelectedNick:String
-selectedMember:String
+NewPatronView(AddParty)
+actionPerformed(ActionEvent)
+done():boolean
+getNick():String
+getEmail():String
+getFull():String
Bowler
-fullName:String
-nickName:String
-email:String
+getter()
EndGameReport
-result:int
-selectedMember:String
+EndGameRepor(String,Par
+actionPerformed(event)
+valueChanged(event)
+getResult()
+destroy():
+main(String args[]):

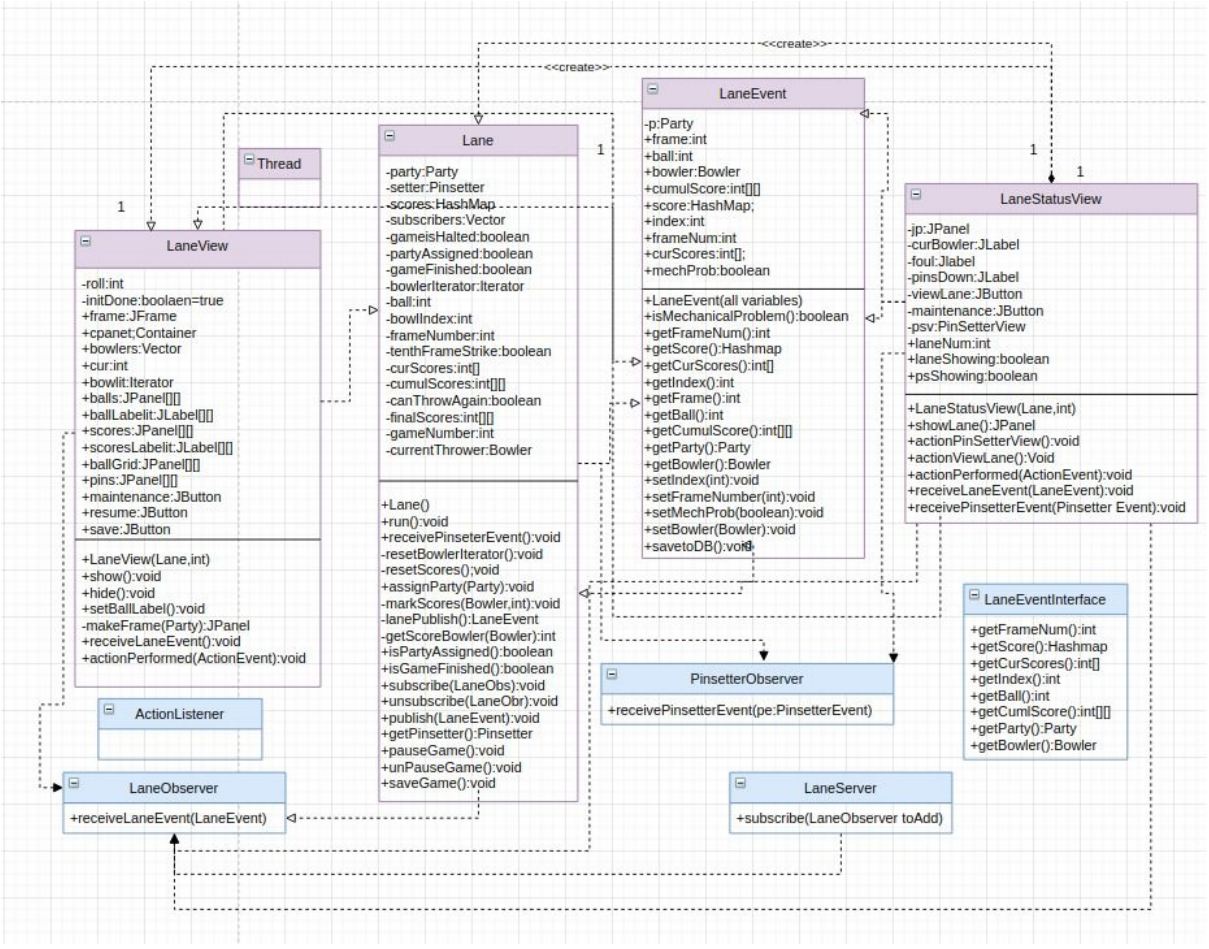

Lane
Lane
LaneStatusView
ControlDeskView
- addParty : JButton
- finished : JButton
- assign : JButton
- win : JFrame
- partyList : JList
- maxMembers : int
+ controlDeskView(ControlDesk, int)
+ actionPerformed(ActionEvent) : void
+ updateAddParty(AddPartyView) : void
+ receiveControlDeskEvent(ControlDeskEvent) : vc
ControlDeskEvent
- partyQueue : Vector
+ ControlDeskEvent(Vector)
+ getPartyQueue() : Vector
ControlDesk
- lanes : HashSet
- partyQueue : Queue
- numLanes : int
- subscriber : Vector
+ ControlDesk(int)
+ run() : void
- registerPatron(String) : Bowler
+ assignLane() : void
+ viewScores(Lane) : void
+ addPartyQueue(Vector) : void
+ getNumLanes() : ini
+ subscribe(ControlDeskObserver) : void
+ publish(ControlDeskObserver) : void
+ publish(ControlDeskEvent) : void
+ getLanes() : HashSet
Queue
- v : Vector
+ Queue()
+ add(Object) : void
+ next() : Object
+ hasMoreElements() : book
+ asVector() : Vector
«interface»
ControlDeskObserver
+receiveControlDeskevent(ControlDeskEvent) : void
drive
+ drive()
+ main(String[]) : void
Alley
+ controldesk : ControlDesk
+ getControlDesk() : ControlDesk
+ Alley(numLanes : Int)
Thread
Extends
<<create>>
return

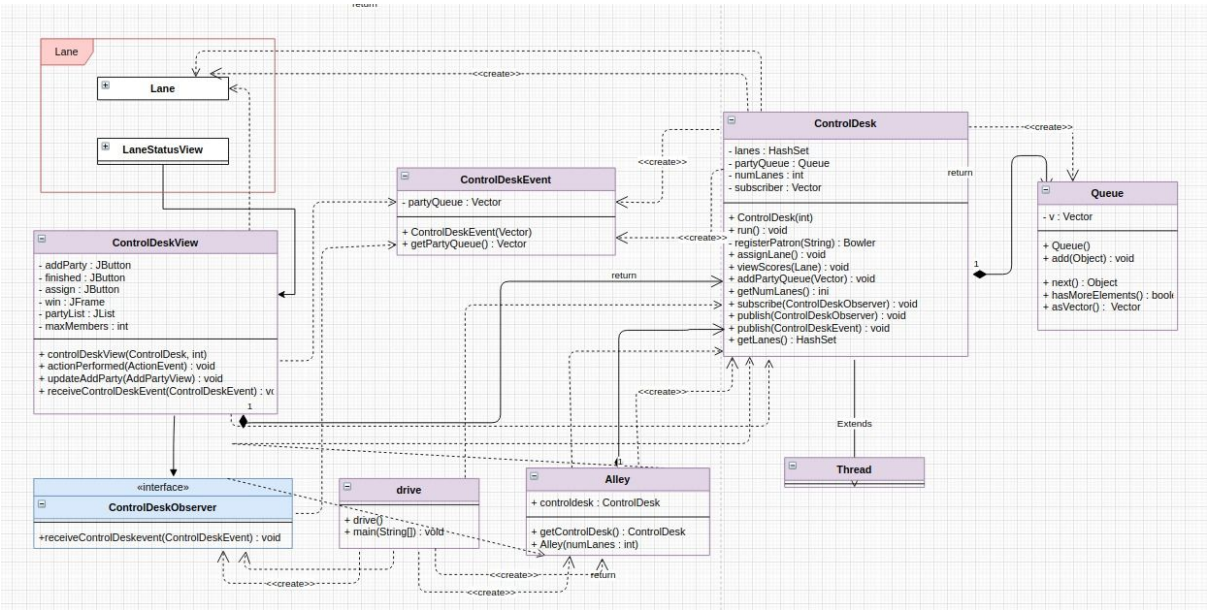


AFTER REFACTORING

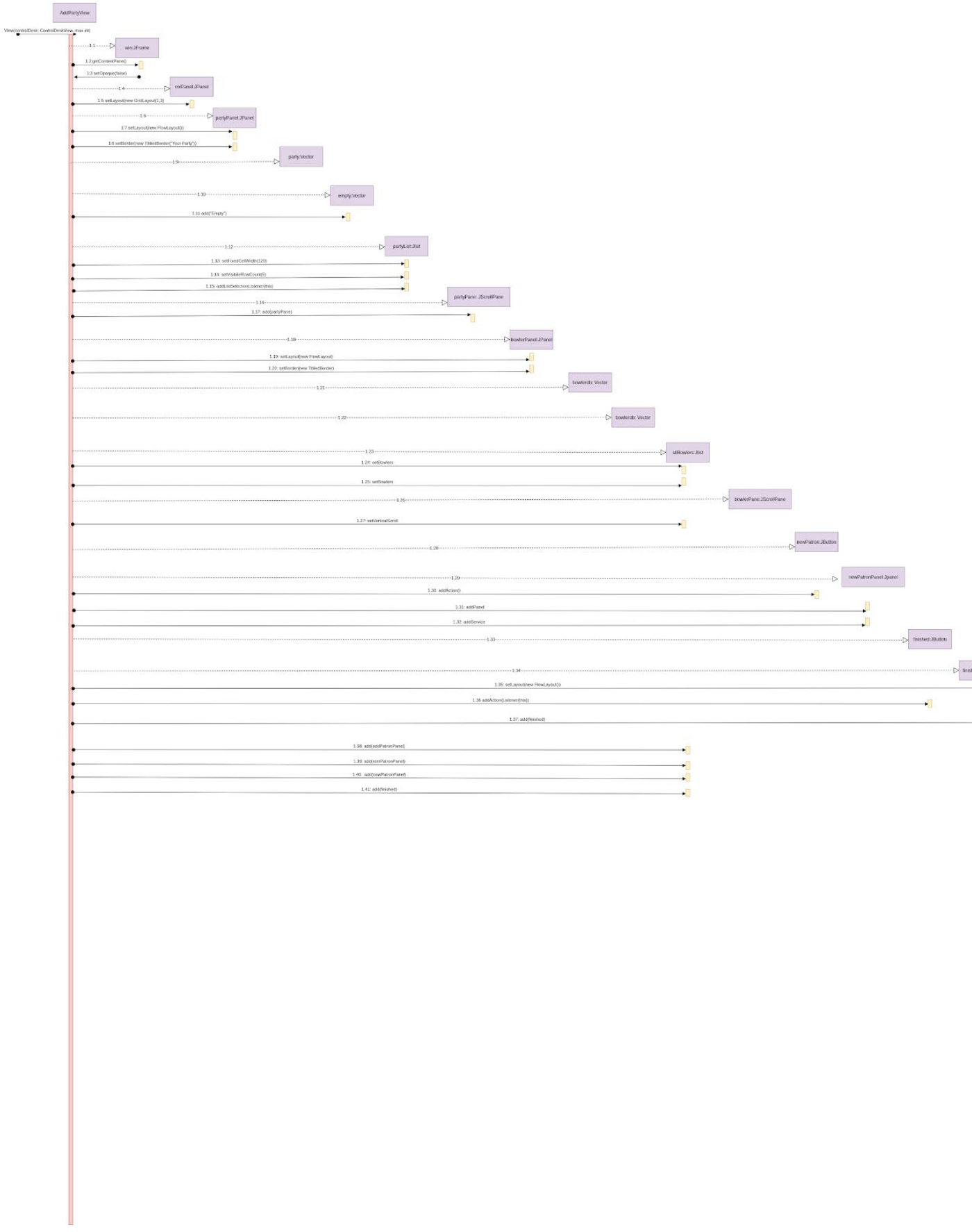



AddPartyView
- maxSize:int
-win:JFrame
-addPatron:JButton
-newPatron:JButton
-remPatron:JButton
-finished:JButton
-partyList:JList
-allBowlers:JList
-party:Vector
-bowler_db:Vector
-selectedNick:String
-selectedMem:String
+AddPartyView():
+actionPerformed(e:ActionEvent)
+getNames():Vector
+valueChanged():
+getParty():Vector
+updateNewPatron(NewPatronView)
ControlDesk
1
ControlDeskView
<<create>>
EndGamePrompt
-result:int
-win:JFrame
-yesButton:JButton
-noButton:JButton
-selectedNick:String
-selectedMember:String
+EndGamePrompt(partyNar
+actionPerformed(Event)
+getResult():int
+distroy():
BowlerFile
-BOWLER_DAT:String
+getBowlerInfo:Bowler
+putBowlerInfo():
+getBowler():Vector
+registerPatron(Str):Bowler
Lane
<<create>>
Lane
1
<<create>>
<<create>>
<<create>>
Bowler
-fullName:String
-nickName:String
-email:String
+Bowler(String,String,String
+getNickName():String
+getFullName():String
+getNick():String
+getEmail():String
+equals(Bowler):boolean
1
<<create>>
EndGameReport
-win:JFrame
-printButton:JButton
-finished:JButton
-myVector:Vector
-retVal:Vector
-selectedNick:String
-selectedMember:String
+EndGameRepor(String,Par
+actionPerformed():void
+valueChanged():void
+actionPerformed(event)
+getResult()
+destroy():
<<create>>
NewPatronView
-maxSize:int
-win:JFrame
-abort:JButton
-finished:JButton
-nickField:JTextField
-fullFleld:JtextField
-emailField:JTextField
-nick:String
-full:String
-email:String
-done:boolean
-addParty:AddPartyView
-SelectedNick:String
-selectedMember:String
+NewPatronView(AddParty)
+actionPerformed(ActionEvent)
+done():boolean
+getNick():String
+getEmail():String
+getFull():String


Thread
Lane
-party:Party
-setter:Pinsetter
-scores:HashMap
-subscribers:Vector
-gameisHalted:boolean
-partyAssigned:boolean
-gameFinished:boolean
-bowlerIterator:Iterator
-ball:int
-bowlIndex:int
-frameNumber:int
-tenthFrameStrike:boolean
-curScores:int[]
-cumulScores:int[][]
-canThrowAgain:boolean
-finalScores:int[][]
-gameNumber:int
-currentThrower:Bowler
+Lane()
+run():void
+receivePinseterEvent():void
-resetBowlerIterator():void
-resetScores():void
+assignParty(Party):void
-markScores(Bowler,int):void
-lanePublish():LaneEvent
-getScoreBowler(Bowler):int
+isPartyAssigned():boolean
+isGameFinished():boolean
+subscribe(LaneObs):void
+unsubscribe(LaneObr):void
+publish(LaneEvent):void
+getPinsetter():Pinsetter
+pauseGame():void
+unPauseGame():void
+saveGame():void
LaneEvent
-p:Party
+frame:int
+ball:int
+bowler:Bowler
+cumulScore:int[][]
+score:HashMap;
+index:int
+frameNum:int
+curScores:int[];
+mechProb:boolean
+LaneEvent(all variables)
+isMechanicalProblem():boolean
+getFrameNum():int
+getScore():Hashmap
+getCurScores():int[]
+getIndex():int
+getFrame():int
+getBall():int
+getCumulScore():int[][]
+getParty():Party
+getBowler():Bowler
+setIndex(int):void
+setFrameNumber(int):void
+setMechProb(boolean):void
+setBowler(Bowler):void
+savetoDB():void
LaneStatusView
-jp:JPanel
-curBowler:JLabel
-foul:JLabel
-pinsDown:JLabel
-viewLane:JButton
-maintenance:JButton
-psv:PinSetterView
+laneNum:int
+laneShowing:boolean
+psShowing:boolean
+LaneStatusView(Lane,int)
+showLane():JPanel
+actionPinSetterView():void
+actionViewLane():Void
+actionPerformed(ActionEvent):void
+receiveLaneEvent(LaneEvent):void
+receivePinsetterEvent(Pinsetter Event):void
LaneView
-roll:int
-initDone:boolaen=true
+frame:JFrame
+cpanet;Container
+bowlers:Vector
+cur:int
+bowlit:Iterator
+balls:JPanel[][]
+ballLabelit:JLabel[][]
+scores:JPanel[][]
+scoresLabelit:JLabel[][]
+ballGrid:JPanel[][]
+pins:JPanel[][]
+maintenance:JButton
+resume:JButton
+save:JButton
+LaneView(Lane,int)
+show():void
+hide():void
+setBallLabel():void
-makeFrame(Party):JPanel
+receiveLaneEvent():void
+actionPerformed(ActionEvent):void
PinsetterObserver
+receivePinsetterEvent(pe:PinsetterEvent)
LaneEventInterface
+getFrameNum():int
+getScore():Hashmap
+getCurScores():int[]
+getIndex():int
+getBall():int
+getCumlScore():int[][]
+getParty():Party
+getBowler():Bowler
ActionListener
LaneObserver
+receiveLaneEvent(LaneEvent)
LaneServer
+subscribe(LaneObserver toAdd)
<<create>>
1


Lane
Lane
LaneStatusView
ControlDesk
- lanes : HashSet
- partyQueue : Queue
- numLanes : int
- subscriber : Vector
+ ControlDesk(int)
+ run() : void
- registerPatron(String) : Bowler
+ assignLane() : void
+ viewScores(Lane) : void
+ addPartyQueue(Vector) : void
+ getNumLanes() : ini
+ subscribe(ControlDeskObserver) : void
+ publish(ControlDeskObserver) : void
+ publish(ControlDeskEvent) : void
+ getLanes() : HashSet
ControlDeskEvent
- partyQueue : Vector
+ ControlDeskEvent(Vector)
+ getPartyQueue() : Vector
Queue
- v : Vector
+ Queue()
+ add(Object) : void
+ next() : Object
+ hasMoreElements() : bool
+ asVector() : Vector
ControlDeskView
- addParty : JButton
- finished : JButton
- assign : JButton
- win : JFrame
- partyList : JList
- maxMembers : int
+ controlDeskView(ControlDesk, int)
+ actionPerformed(ActionEvent) : void
+ updateAddParty(AddPartyView) : void
+ receiveControlDeskEvent(ControlDeskEvent) : v
«interface»
ControlDeskObserver
+receiveControlDeskevent(ControlDeskEvent) : void
drive
+ drive()
+ main(String[]) : void
Alley
+ controldesk : ControlDesk
+ getControlDesk() : ControlDesk
+ Alley(numLanes : int)
Thread
Extends
<<create>>
return
1

UML Sequence Diagrams



Code Overview

Class Name	Responsibility
AddpartyView	This class is responsible for creating a view to add a new party. This class implements ActionListener and ListSelectionListener to perform functions in accordance with each button pressed or item selected in JList.
Alley	This class is used to simulate the alley and store related information about the Control Desk, Number of lanes and provide methods to access them.
Bowler	This class is used to simulate Bowler and store information of the nickname chosen, full name and email and provides methods to access them.
BowlerFile	This class contains the methods related to the storage of Bowlers related information. This class creates a BOWLERS.DAT file and stores the information of the Bowlers and also provides a method to access the information.
ControlDesk	This class simulates the behavior of the control desk at the bowling alley and contains methods of managing the game like assigning the empty lane to the party, create a waiting queue and methods to access control desk related information
ControlDeskEvent	This class contains the waiting queue and methods to set and get the Vector
ControlDeskView	This class is responsible for creating a view of the control desk. This class implements ActionListener to perform functions in accordance with each button pressed. The control desk is the main starting window with all options
Drive	This is the main class file it contains the definition of the number of lanes and the number of maxPatrons per party and creates the objects
EndGamePrompt	This class implements a view when the game is

	completed. It provides the option to play another game or to leave the game
EndGameReport	This class implements and presents the view only if the user chooses no to play another game it presents with the option to print the report of the game and the previous score report of the members in the game.
Lane	This is the most significant class managing the whole lane. It contains methods to manage the lane related functionalities. It also contains variables to stop and halt the current game so all the functions related to the Running of the game are closely implemented with this class. Also, it implements PinSetterObserver to make PinSetterView coupled with this class and provide synchronization between them
LaneEvent	This class is the class to simulate the Event and store all related information to Lane Events like Current Bowler, Current Scores, etc. This class provides getters and setters for the information Stored. This class is helpful for getting the current state of the Lane if required.
LaneStatusView	This class is used in showing the lane status view of each lane. Each of its objects is embedded on the ControlDeskView, It provides options such as current bowler, no of pins down and the option to resume if in case its paused, etc. It implements LaneObserver and PinSetterObserver to provide the required information on each event.
LaneView	This class implements the detailed View for each lane like a detailed score of each bowler and displays the current state of each game
NewPatronView	This class implements a view for adding NewPatron to the system. Its also implements the action listener for the response to the buttons and implements getter and setter for getting information about the Patron

Party	This class holds for details for Bowlers and provide getters and setters for them.
Pinsetter	This class is used to simulate PinSetter it holds the information's about the pins and methods to control them like resetting and send notification on event etc.
PinSetterEvent	This class contains methods for events related to pins like the number of pins down and check if foul is committed etc.
PinSetterView	This class implements the view for the Pinsetter and methods to change the view once the status of pins changes during the bowling event.
PrintableText	This class provides methods for converting the text to a printable format.
Queue	This class helps to provide a queue and related functions for managing the queues.
Score	This class is helpful in storing the scores of the players and related information like nickname, date and the score and contains related methods which have the relevance to score processing
ScoreHistoryFile	This class provides methods for storing and getting the score from the file for the given nickname.
ScoreReport	This class provides methods for generating the reports for the given person and formats them in a presentable format

CODE ANALYSIS

During the analysis of code following points were observed

Strengths

1. The code has good comments which help in understanding the code.
2. The code was well divided into Classes in a meaningful way.
3. The Classes have only important class-variables.
4. Appropriate logging is being done at appropriate places to simulate real-world behavior.
5. Some of the methods are being reused appropriately to reduce the code.

Weaknesses

1. The complexity of a few of the functions was really high which decreases the readability of the code of those functions.
2. The fairly high number of blocks of if and else were nested which produce hindrance in reading the actual conditions. The branching could have been reduced by joining the conditions
3. Some of the functions required a large number of parameters which is bad coding practice since it becomes difficult to remember the order of the parameters that need to be passed.
4. There were few unused variables which are also the code-smell.
5. There were a few functions that had repetitive implementation with different names.
6. Some of the buffers read, file access, socket connections as not enclosed in a try-catch block and hence may lead to giving unreadable errors
7. There were few variables in the methods that have the same name as of the class variables.
8. There were few points where conditions always evaluate to true but still condition check is being done.
9. There were few catch blocks with empty implementation.

10. There were few places where implementation was repeated they could be enclosed in the separate methods.

CODE SMELLS

CODE SMELLS	DESCRIPTION
AddPartyView	• There were few unused local variables • The commented out code could be removed to provide better readability • GetNames is similar to GetParty function so one of the function could be removed and wherever the function was used could be changed • actionPerformed function has slightly more complexity higher than metrics value.
Bowler	• getNick and getNickName have identical implementations so one of the getters can be removed. • The equal function can be renamed to prevent confusion. • The naming of the variables can be improved to comply with standard regex.
BowlerFile	• The Bowler File doesn't have the private constructor which is nice to implemented under Singleton Design Pattern • The Buffer read and file read are implemented without try-catch block
Control Desk	• Override annotation can be added to provide better readability. • The registerPatron can be moved to Bowler file since to improve cohesion
ControlDeskView	• The commented out code can be removed. • The @override annotation can be added at windowClosing function • addPartyWin seems to be a useless local

	variable
EndGamePrompt	• The EndGame report has some unused private variables. • thread in catch block can be restarted if the interrupt was received by a thread. • Printing of errors could be done through Logger rather than using System.out.println()
EndGameReport	• myVector is used as a class variable as well as a local variable in the method. • Useless local variable e can be removed.
Lane	• The complexity of the get score function is very high and hence needs to reduced by altering the conditions and dividing them further into meaningful functions. • The comparison using == for strings needs to be removed because sometimes it gives wrong results. • The depth of the nested if-else conditions was reduced by merging the unnecessary if conditions. • The number of if-else conditions was reduced if two conditions performed the same functions.
LaneEvent	• The lane event lacks cohesion very much because all the variables have separate getters and setters. • Few more setters could be implemented to have an implementation in building design patterns.
LaneStatusView	• The Lane status views have a similar implementation of all the buttons which can be enclosed in a method to reduce repetitive code. • Few unused local variables can be removed.

LaneView	• recieveLaneEvent has high complexity. The complexity can be reduced by breaking them in meaningful functions further. • The catch block needs to have some implementation or appropriate logging.
NewPatronView	• It has a similar implementation of all the buttons which can be enclosed in a method to reduce repetitive code.
ScoreHistory	• Implementation of file reading can be done in try and catch block for better error handling • Private Constructor needs to be initialized under the Singleton Design Pattern.

OUTLINE FOR REFACTORING

The code was refactored on the above-noticed points. The eclipse metric was used for the refactoring. The metric showed the point where the code had the errors and mostly It presented the complexity errors, parameters errors, and depth of the block. Refactoring on the above points mostly helped to align the metric score to the standard metrics but few places where The design pattern could be implemented but It was affecting the metric score. Also implementing the design pattern helps in better design of the code. For example, Implementing the building design pattern on LaneEvent helps in setting the parameters in the better and easier way and help to stand in accordance with parametric constraint in metrics but at the same time including more methods for getter and setter mack the class less-cohesive or the other way could be so opted to implement the builder design pattern the other way was to put those variables and fit them in some pre-existing classes but at the same time this could increase the complexity of the functions and will also make the code less meaningful. The other tradeoff observed was during the attempts made to reduce the complexity of the Lane class as a whole. The lane class has multiple methods with complexity higher than

the standard value; they were broken into meaningful functions but breaking them leads to making the class less-cohesive so we balanced both the parameters at the same time and tried implementing them in a balanced way. We also tried implementing multiple design patterns but some of them made the codebase deviate from the standard values so we avoided them and implemented those which seemed to be important or it didn't make the metric score away from the desired ranges. There are classes that help in reducing the reusability of the code but at the same time lacks cohesion. So we refactored the code in a way so that It stands along with all the parameters.

METRIC USED FOR CODE ANALYSIS

For Old Code

Metric	Total	Mean	Std. Dev.	Maximur	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/max per method)		2.319	4.062	38	/new/src/Lane.java	getScore
Number of Parameters (avg/max per method)		0.723	1.131	9	/new/src/LaneEvent.java	LaneEvent
Nested Block Depth (avg/max per method)		1.511	1.177	7	/new/src/Lane.java	run
Afferent Coupling (avg/max per packageFragment)		0	0	0	/new/src	
Efferent Coupling (avg/max per packageFragment)		0	0	0	/new/src	
Instability (avg/max per packageFragment)		1	0	1	/new/src	
Abstractness (avg/max per packageFragment)		0.172	0	0.172	/new/src	
Normalized Distance (avg/max per packageFragment)		0.172	0	0.172	/new/src	
Depth of Inheritance Tree (avg/max per type)		0.897	0.48	2	/new/src/Lane.java	
Weighted methods per Class (avg/max per type)	327	11.276	15.991	87	/new/src/Lane.java	

For Refactored Code

Metric	Total	Mean	Std. Dev.	Maximur	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/max per method)		2.144	2.156	11	/CHECKING/src/Lane.java	getScore
Number of Parameters (avg/max per method)		0.744	0.944	5	/CHECKING/src/LaneEvent.java	LaneEvent
Nested Block Depth (avg/max per method)		1.562	1.082	5	/CHECKING/src/Lane.java	gameFinished
Afferent Coupling (avg/max per packageFragment)		0	0	0	/CHECKING/src	
Efferent Coupling (avg/max per packageFragment)		0	0	0	/CHECKING/src	
Instability (avg/max per packageFragment)		1	0	1	/CHECKING/src	
Abstractness (avg/max per packageFragment)		0.172	0	0.172	/CHECKING/src	
Normalized Distance (avg/max per packageFragment)		0.172	0	0.172	/CHECKING/src	
Depth of Inheritance Tree (avg/max per type)		0.897	0.48	2	/CHECKING/src/ControlDesk.java	
Weighted methods per Class (avg/max per type)	343	11.828	16.756	92	/CHECKING/src/Lane.java	

a) The initial values can tell that there were some issues in many files that are affecting the code and code is complex enough and formed a scary image in a mind but It seemed fun analyzing the code and getting the overview. The steps to reduce nested block depth seemed a little pane in starting but could be resolved.

b) The metric value for Nested block Depth and Cyclomatic complexity sometimes go hand in hand reducing the one parameter and sometimes reducing the other automatically. The values guided me to reach the respective functions and then either combined the statements or broke them into functions. The values tell a lot about the codebase

c) Most of the refactoring either reduced the values or didn't affect the metrics only some affected them in a negative way. Few of the functions with very high complexity were broken down into functions but into meaningful functions, most of them could be made to desired ranges but the getScore method couldn't be broken down into more meaningful form hence we are slightly above the range in the metric. Implementing the builder design pattern affected the metric in a negative way and led to a lack of cohesiveness. Similarly reducing the complexity led to a lack of cohesiveness and sometimes improving the cohesiveness led to less sensible methods in the class so there were a lot of the tradeoffs that we dealt with.