Design Document

Group AO

Fanish Jain, T. Shiva Prasad, Shradha Sehgal

Bowling Game

Our final project is entitled **Bowling Game** and our team consists of **Fanish Jain, T. Shiva Prasad**, and **Shradha Sehgal**. Over the span of one month since this assignment's release, we divided our work efficiently and completed each of our responsibilities diligently. We combined all our efforts towards the end and integrated our modules. It was finally completed and submitted on **9th April 2020.** We now detail the work of each teammate.

Effort And Role of Each Teammate:

1. **Fanish Jain(2018101021)**
   - Hours put in: 33 hours
   - Roles: Implemented Features 1 and 2, UML Sequence Diagram, Code smells Identification.

2. **T.Shiva Prasad(2018101047)**
   - Hours put in: 30 hours
   - Roles: Responsible for all UML Class diagrams, detailed analysis of previous vs new code, Code smell Identification.

3. **Shradha Sehgal (2018101071)**
   - Hours put in: 32 hours
   - Roles: Responsible for refactoring of codebase, feature 3, and metrics Analysis.

Overview

Our project is a software system called **Bowling Game** that allows users to manage a virtual Bowling Alley. It has been written entirely in Java. We started with a basic code but refactored it so as to get rid of code smells and improve the code quality.

The game is completely modular as all functionalities are separated into various files.
We fixed various bugs in their code such as **incorrect score calculation** in the first frame and the **game arbitrarily stopping** on the press of too many buttons.

The already existing features that we refactored are as follows:

- Users can observe a game on 3 different lanes with parties of friends.
- The User can also add New Patrons with the feature in Add Party.
- The score can be checked via the View Lane feature, and the pins arrangement after each round via pinsetter feature.
- An ongoing game can be paused for maintenance calls via the button for the same.
- Upon termination of the game, the user can choose to play again or finish game (along with an option to get player reports).

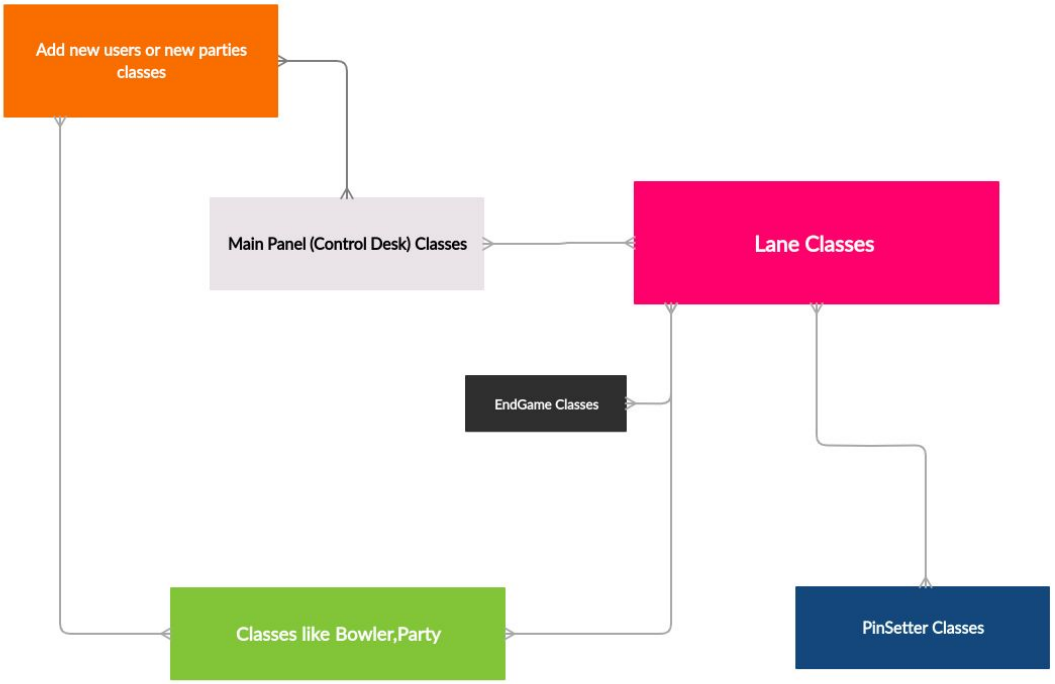
Apart from this, we implemented features such as

1. **Adding 6 or more members** to a party
2. A **Database Layer** to allow for queries such as Top Player, highest / lowest scores
3. **Pause and Resume** feature that enables you to load a game even upon closing the window.

We now move on to (briefly) explain the classes and important functionalities in the game. We have coupled this with UML class and sequence diagrams for greater clarity.

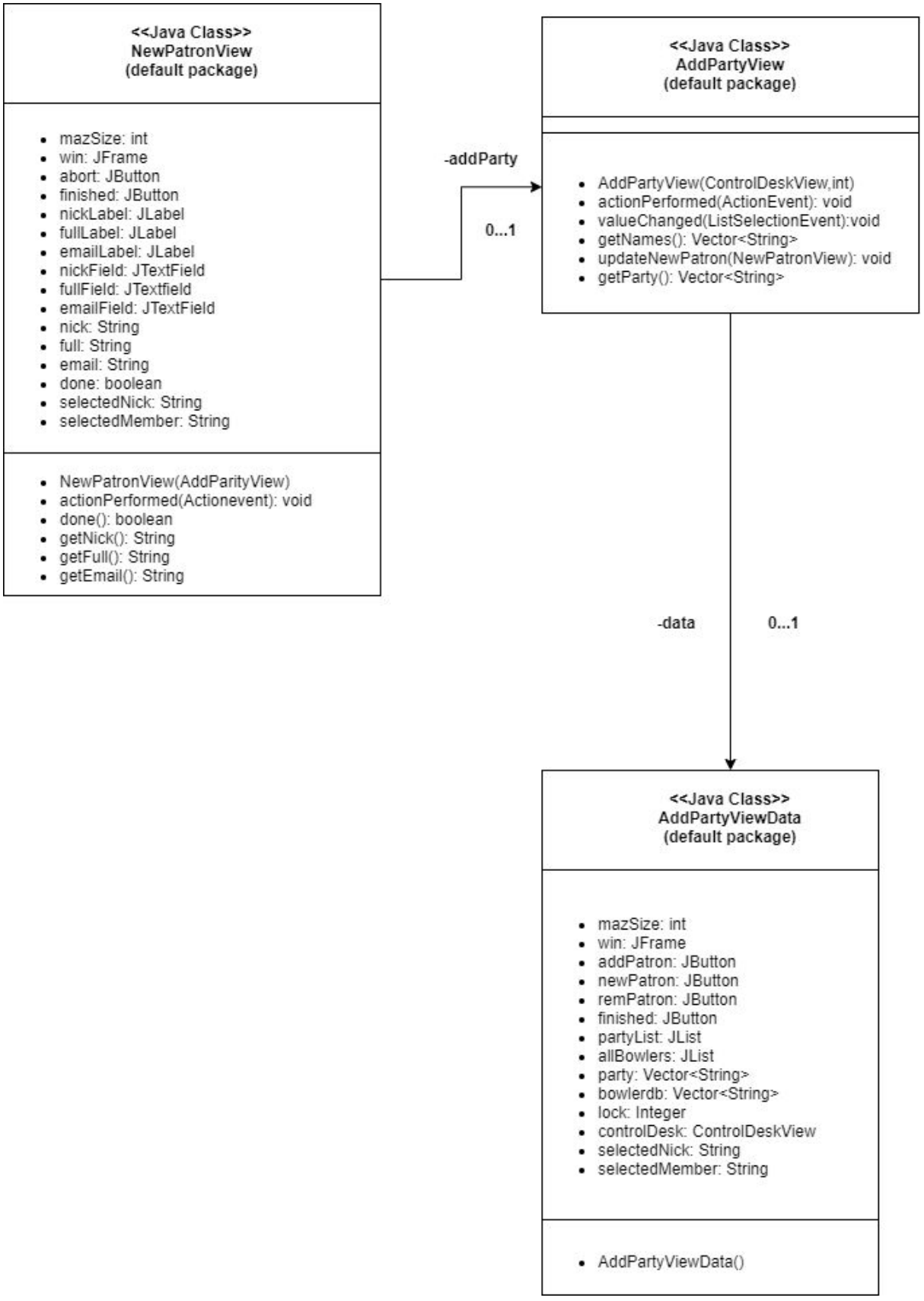
UML Class Diagram

We have broken down the project into various features and represented these for clarity and ease of readability. An overview is offered here:

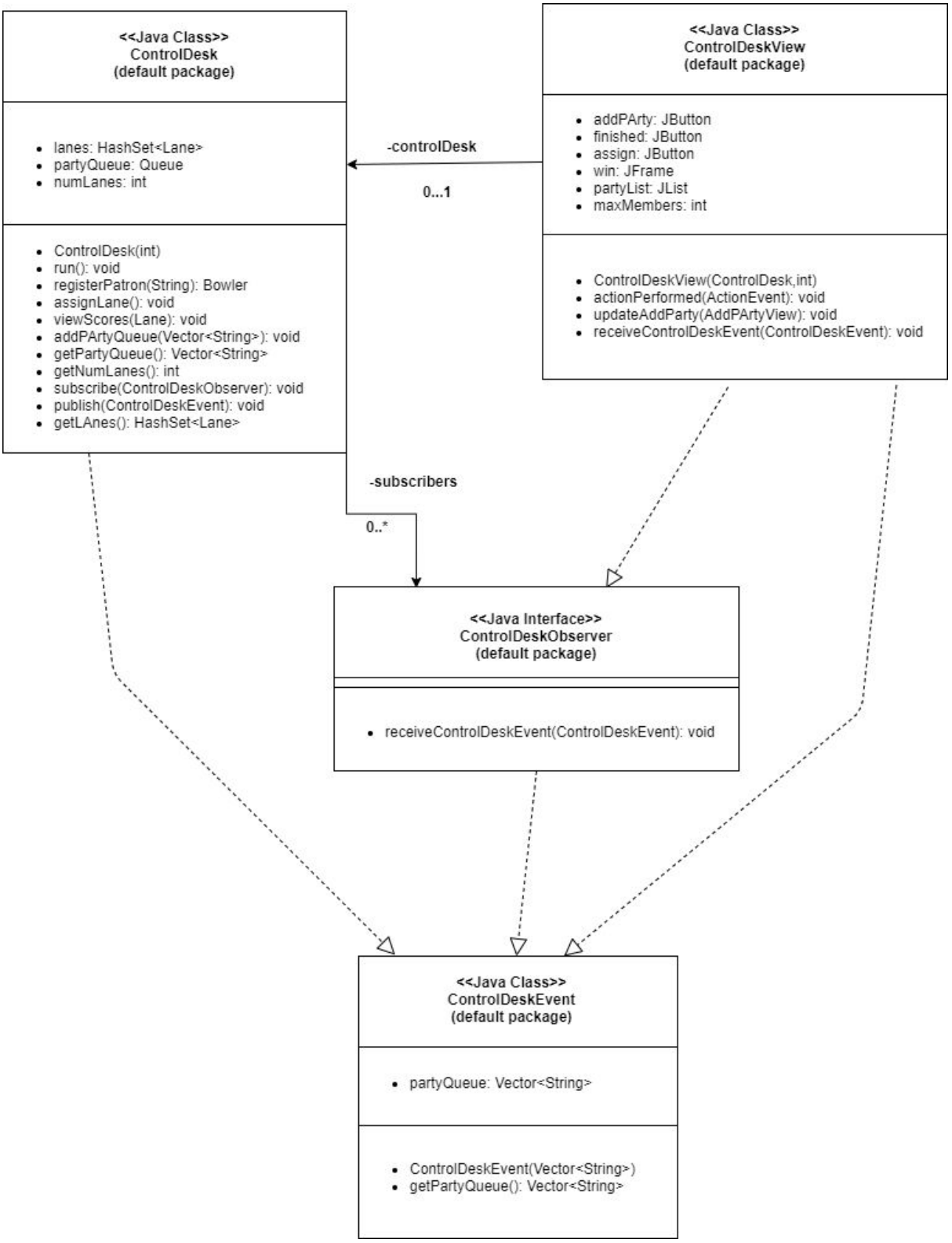


Now the detailed classes drawn in draw.io are here:

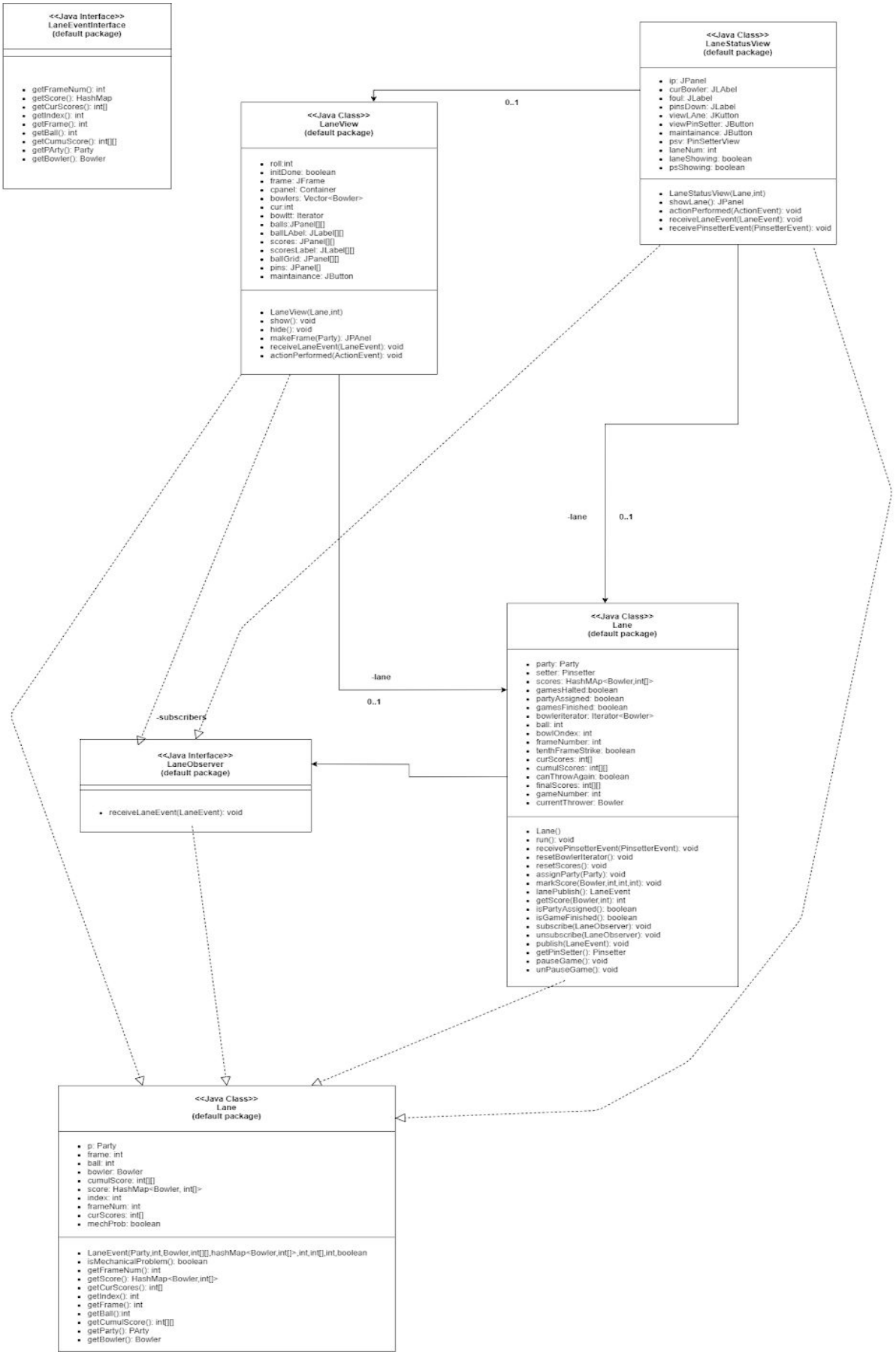
Add new Parties or Patrons classes:



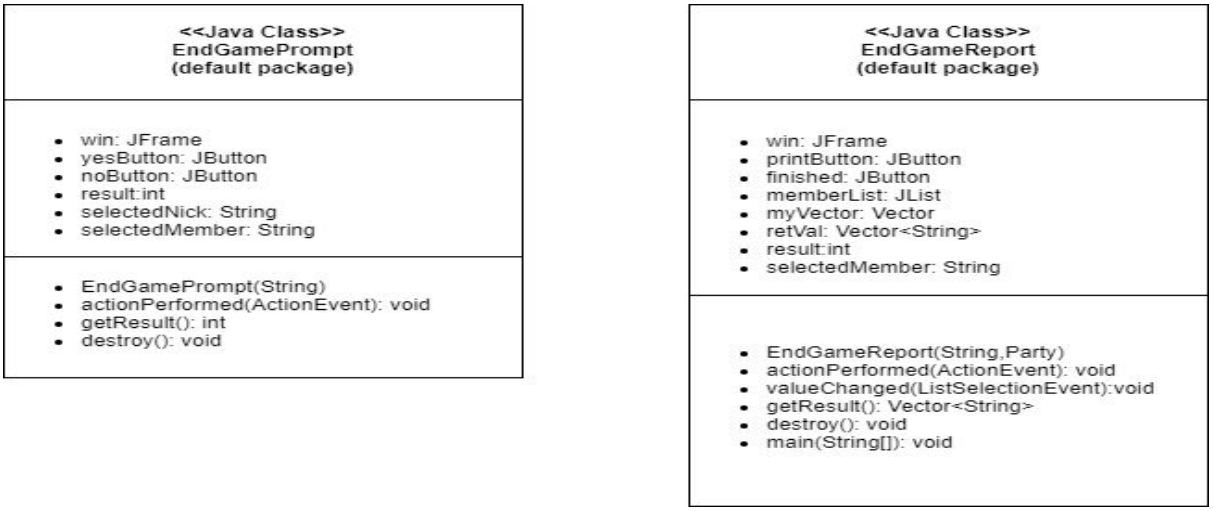
Main Panel(Control Desk) Classes:



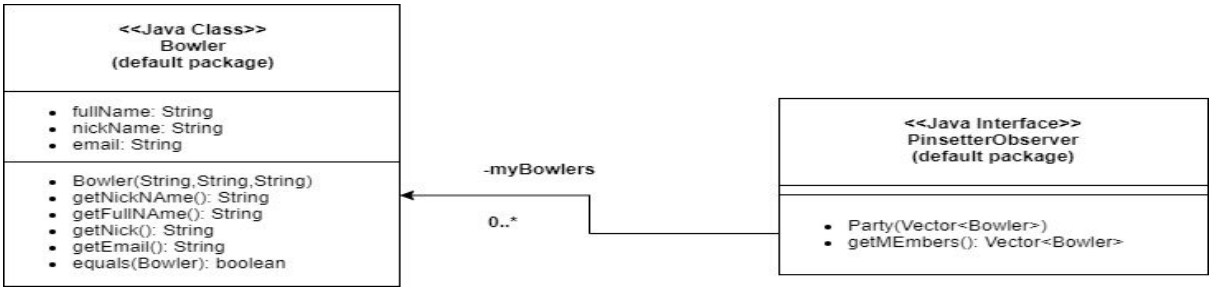
Lane Classes:



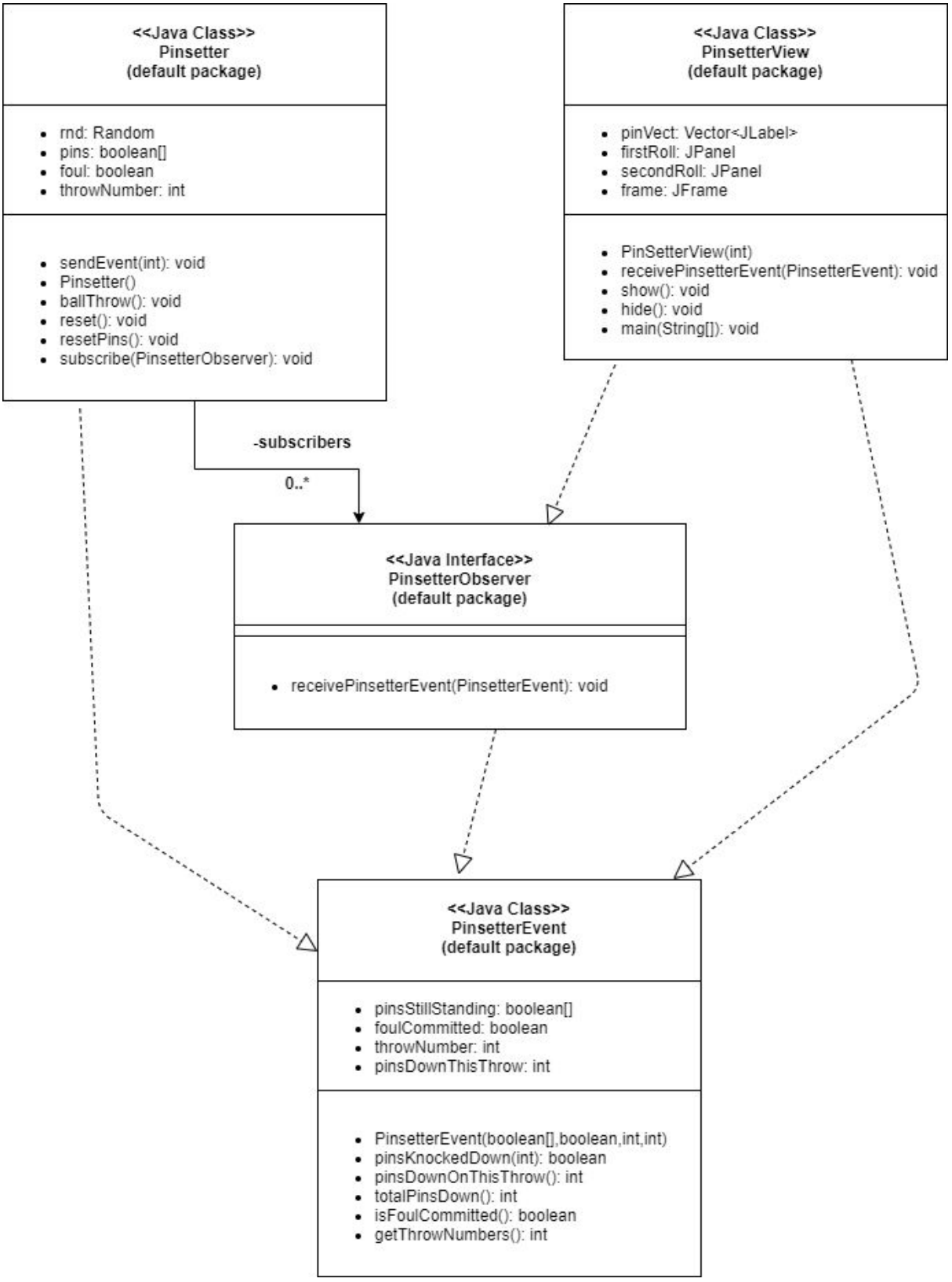
End Game Classes:



Bowler Classes:



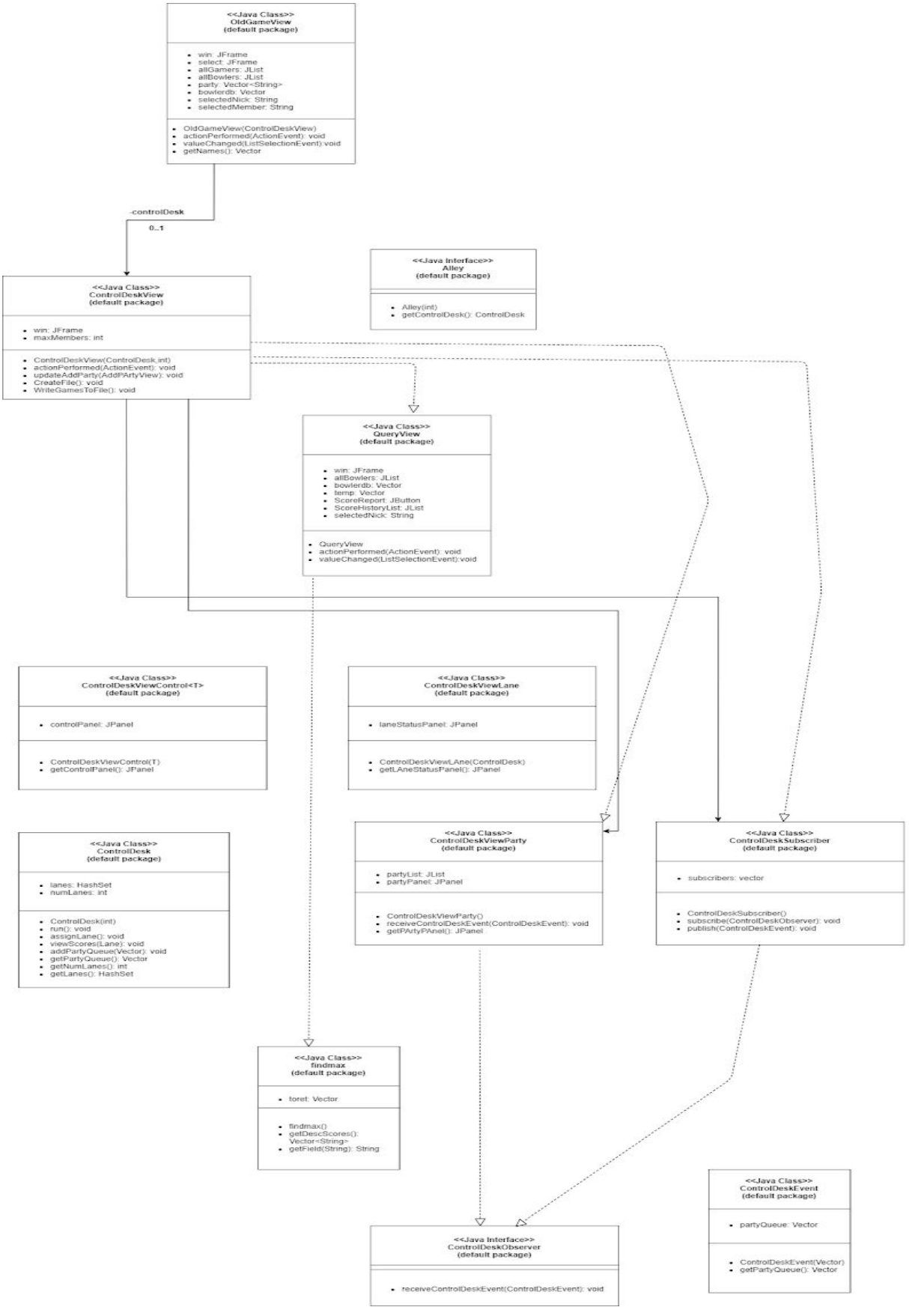
Pinsetter Classes:



New UML Class Diagram

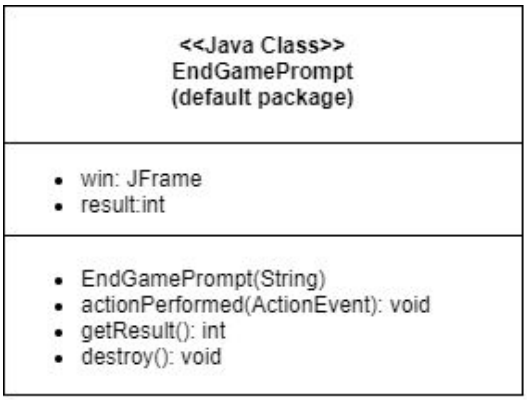
We tried to show new classes and new associations :

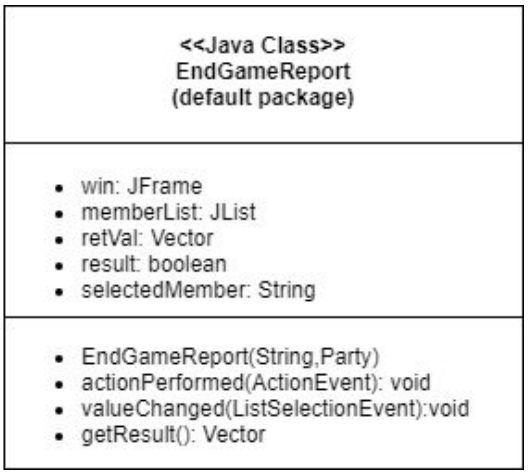
Main Panel(Control desk) Classes:



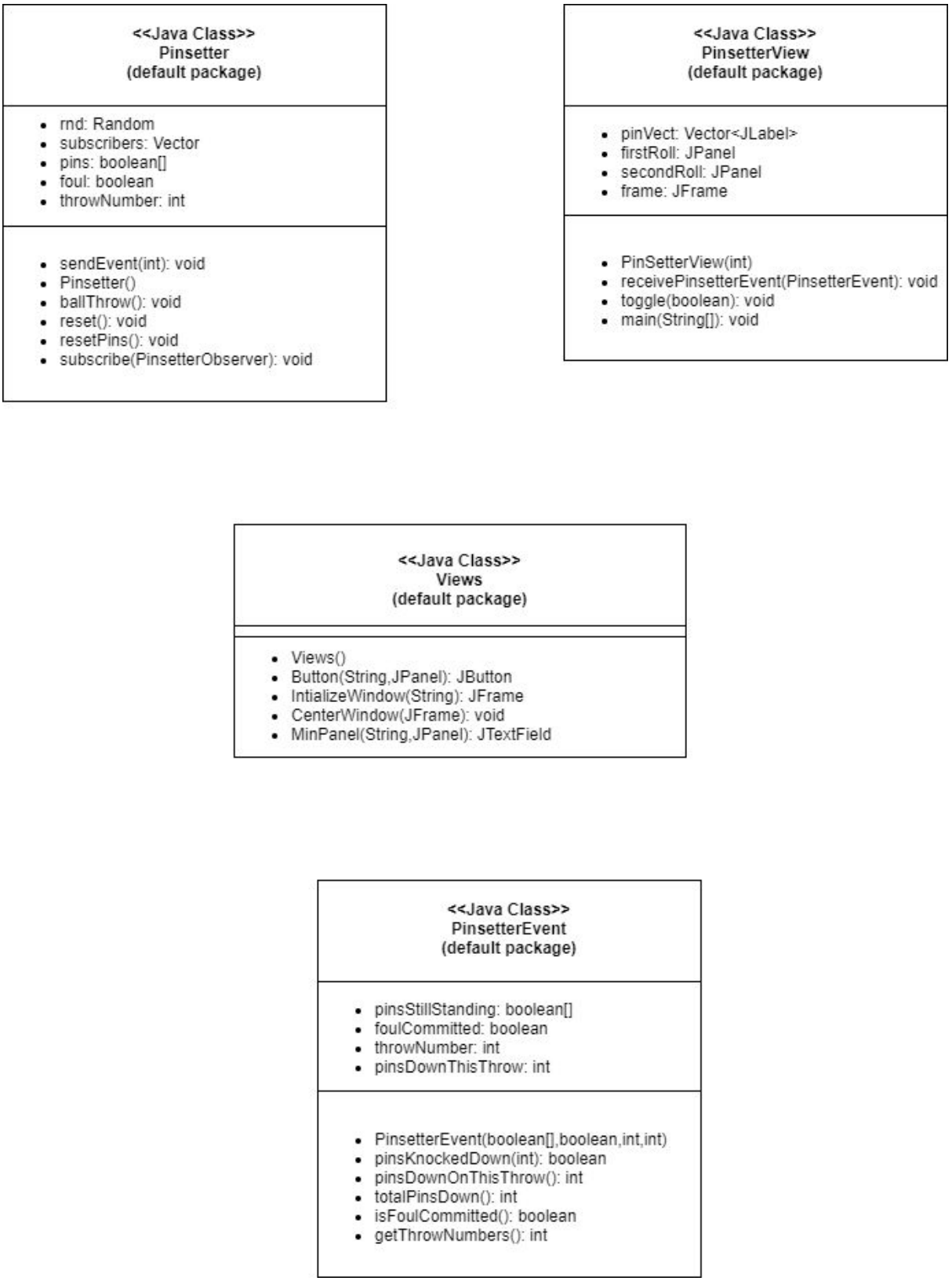
Lane Classes:

End Game Classes:

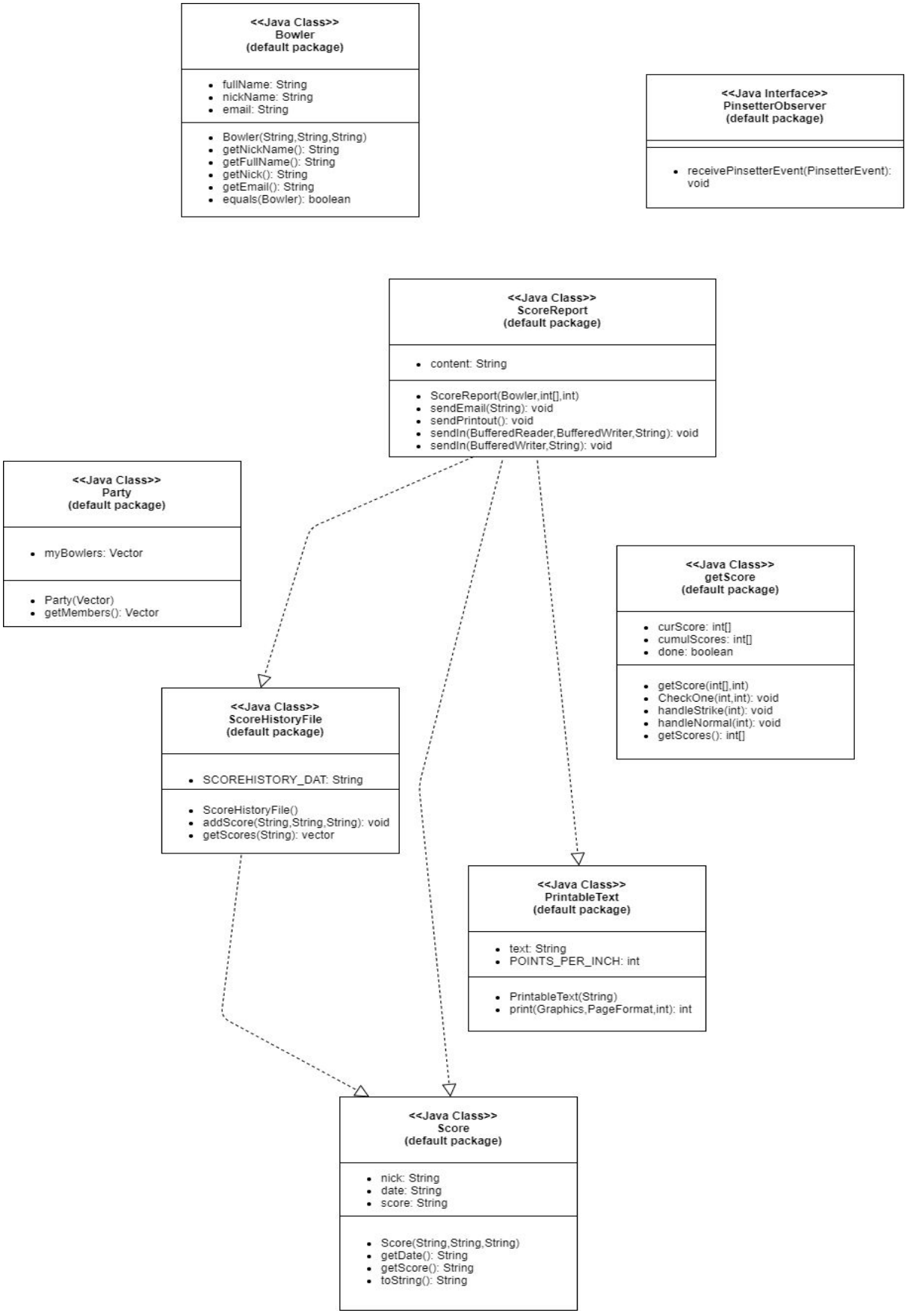




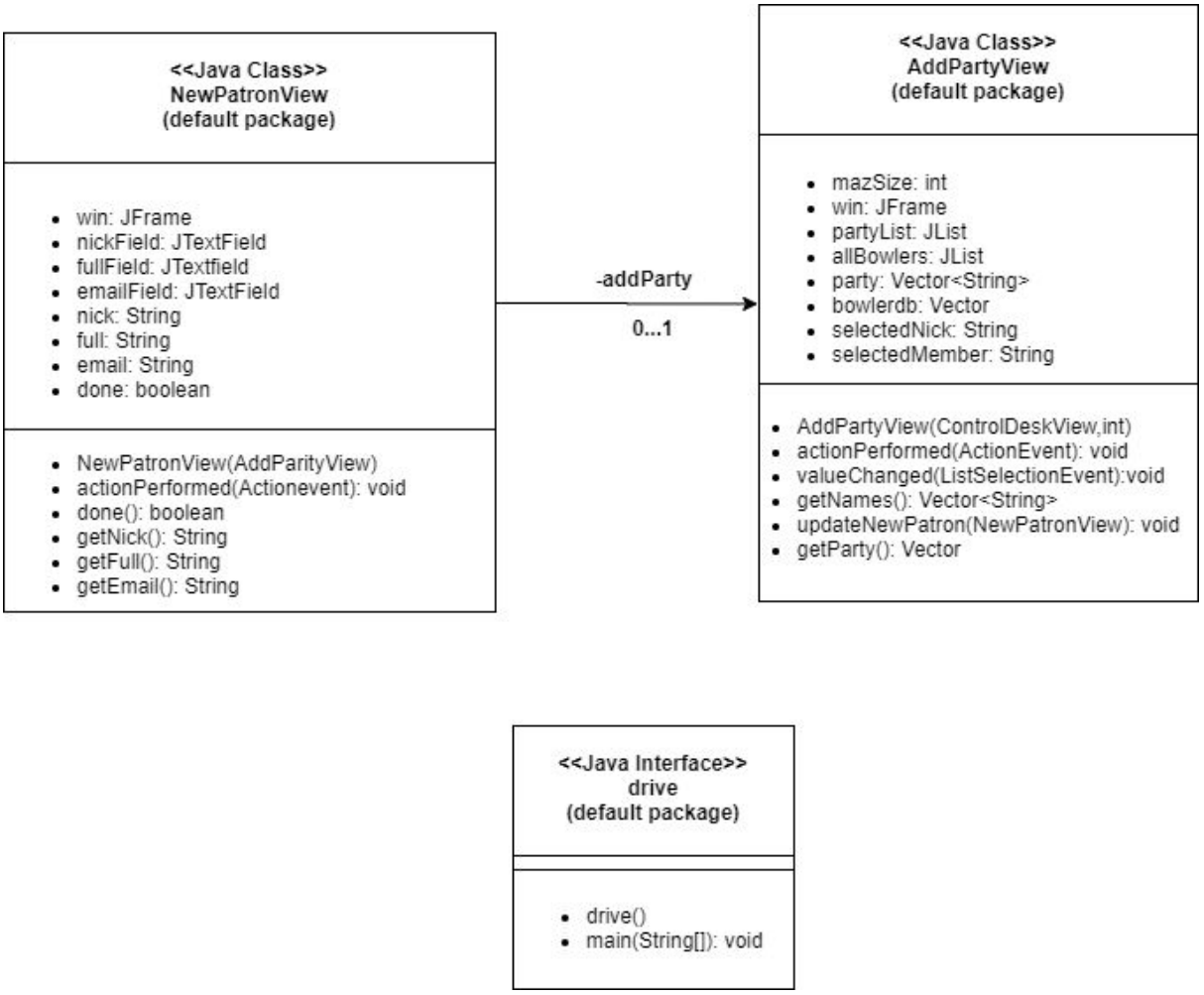
Pin Setter Classes:


<<Java Class>>
Pinsetter
(default package)
rnd: Random
subscribers: Vector
pins: boolean[]
foul: boolean
throwNumber: int
sendEvent(int): void
Pinsetter()
ballThrow(): void
reset(): void
resetPins(): void
subscribe(PinsetterObserver): void
<<Java Class>>
PinsetterView
(default package)
pinVect: Vector<JLabel>
firstRoll: JPanel
secondRoll: JPanel
frame: JFrame
PinSetterView(int)
receivePinsetterEvent(PinsetterEvent): void
toggle(boolean): void
main(String[]): void
<<Java Class>>
Views
(default package)
Views()
Button(String,JPanel): JButton
IntializeWindow(String): JFrame
CenterWindow(JFrame): void
MinPanel(String,JPanel): JTextField
<<Java Class>>
PinsetterEvent
(default package)
pinsStillStanding: boolean[]
foulCommitted: boolean
throwNumber: int
pinsDownThisThrow: int
PinsetterEvent(boolean[],boolean,int,int)
pinsKnockedDown(int): boolean
pinsDownOnThisThrow(): int
totalPinsDown(): int
isFoulCommitted(): boolean
getThrowNumbers(): int

Bowler Classes:

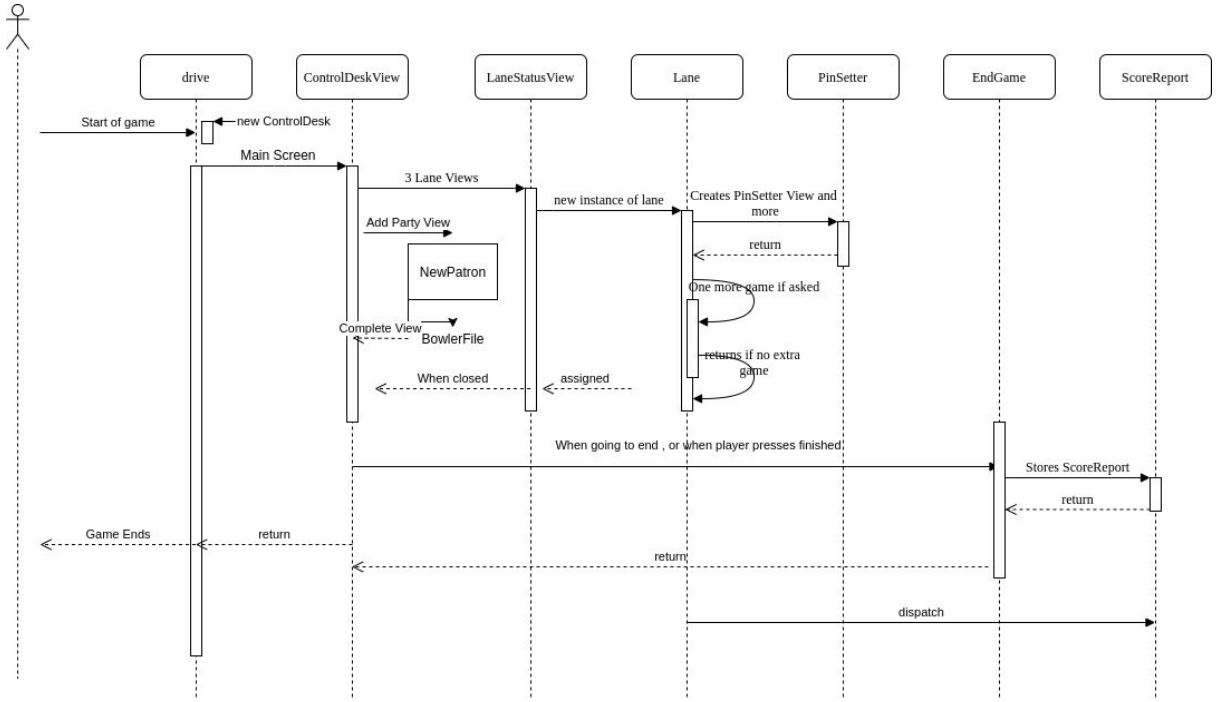


New Parties and Patrons Classes:



UML Sequence Diagram

Old Sequence Diagram



New Sequence Diagram

Differences have been marked with blue arrows. As you can see, we have added some classes so as to split the long methods and increase cohesion. However, to keep coupling in check, we have ensured that not too many parameters are passed to these new files.

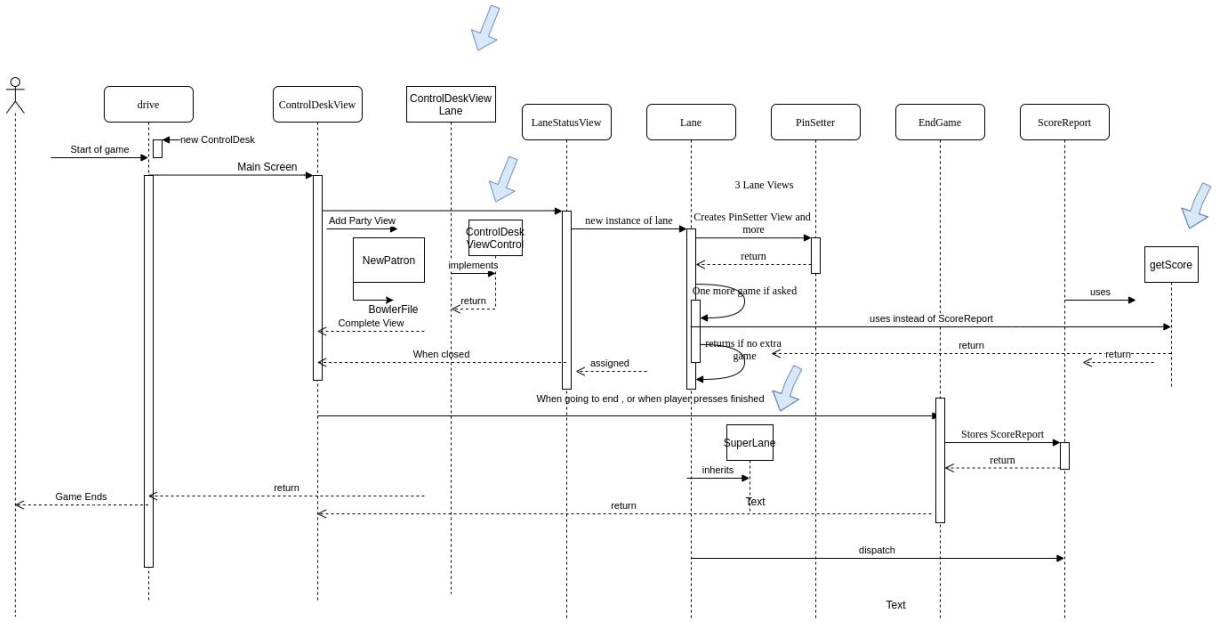


Table with Class Responsibilities

The following is a table containing responsibilities handled by each major class.

Class Name	**In File**	**Responsibilities**
AddPartyView	AddPartyView.java	• Displays parties on the Control Desk
Bowler	Bowler.java	• Create a new Bowler • Store the Bowler's details
BowlerFile	BowlerFile.java	• Manages Bowler database

ControlDesk	ControlDesk.java	• Initiates the main control desk. • Assigns Lanes to Parties • Party Queue maintenance • acts as master class with other classes subscribed to it.
ControlDeskView	ControlDeskView.java	• Acts as GUI for the entire game. • Manages inputs from users.
Drive	drive.java	• Initiates the game.
EndGamePrompt	EndGamePrompt.java	• Displays game end popup • Takes
EndGameReport	EndGameReport.java	• Displays report of the game
Lane	Lane.java	• Initiates the Lanes • Contains Lane elements Pins, Score, game status • Pauses/unpauses game.
LaneStatusView	LaneStatusView.java	• Displays Lane Status, • GUI to make Maintenance calls for a lane. • Shows the lane's pinsetter.
NewPatronView	NewPatronView.java	• Addition of new Patrons to the Database via the GUI.

PinSetter	PinSetter.Java	• Resets pins after each round • Drops pins randomly for each round of ball thrown(gameplay Simulation).
PinSetterView	PinSetterView.java	• Displays the status of Pins after each throw.
ScoreHistoryFile	ScoreHistoryFile.java	• Manages Score database

Original Code Narrative

The original code at first looked very complicated and took us approximately 5 days to understand! However, when we read the files closely, we realized that all the functionality was required and more or less well written. We just had to make it cleaner and split the classes. A more granulated look at the code tells us this:

Design analysis within classes

1. **Long Methods**
   Some of the classes are **arbitrarily long** such as Lane.java. It also has the highest cyclomatic complexity. These could be easily split up and redistributed as we have done in our refactored code. The file sizes are also very varying as some classes have close-to-nothing code such as Alley.java and drive.java but the others are very extensive.

2. **Dead Code**
   A lot of the places had dead code that was not being used or variables and functions that were not called anywhere. These add to the LOC.

3. **Cyclomatic Complexity**
   This was frequent occurring in the project. Many methods had unnecessary if-else conditions and made them super long! Conditional statements should be made crisp and precise.

4. **Repetitive / Duplicate Code**
   This was also frequent in the project initially. Code for creation of buttons, panels, windows is repeated everywhere. Even code that performs the exact same functionality was found in places!

Design analysis between classes

1. **Parameters passing**
   Whilst a good balance has been maintained mostly. Some classes do pass too many parameters. Such as LaneEvent constructor receives 8 parameters at once which is too much!

2. **Feature Envy**
   This occurs often in the code. A lot of functions of classes such as ControlDesk, Lane etc are used in other files extensively.

3. **Indecent Exposure**
   Almost all variables are private and have getter, setter functions which is good! However, some classes such as ControlDesk expose their functions where they are not required also.

Design analysis in general

1. **Commments**
   These were very consistent throughout the code! Infact, they helped us understand what was going on in the code as most functions were coupled with these!

4. **Nomenclature of files and methods**
   The code has been written consistently and the method names communicate their functionality well. Such as all files ending in View are responsible for the UI of the classes in their name before ‘View’. Variable names are also consistent - in some places however, the first letter is capitalized whereas some places it is not. Very minor detail!

Code Smells Documentation

Present in file	Description
LaneView.java	• Has long methods with high conditional complexity.
LaneStatusView.java	• Has high conditional complexity function due to the method actionPerformed(ActionEvent e) given the high action possibilities in the class • Has repetitive code for making buttons and windows for ViewLane, ViewPinStter, Maintenance
Lane.java	• The class is HUGE and needs breaking down for easier readability and debugging • The class's method is too long and has to be broken down into based on the functions run(),ReceivePinSetterEvent(PinsetterEvent pe) and GetScore() function.
ControlDeskView.java	• Variables were declared as hashsets/int instead of var, unnecessary function assignLanes. • The class is unnecessarily big and needs to be broken down.
AddPartyView.java	• Has high conditional complexity in the function actionPerformed(ActionEvent e) due to the number of possibilities of the events in the class. • The feature updateNewPatron is highly based on the usage of the NewPatron class and should be moved there instead.

ControlDesk.java	• The functions addpartQueue and getpartyQueue are highly based on the usage of the class Queue and should be moved there instead. • The subscribe function should be moved to a new class as it sends unnecessary information to the subscribers and decreases cohesion.
NewPatronView.java	• The same panel initialization code is used multiple times for the nickPanel, fullPanel and emailPanel functions which decreases efficiency and increases LOC unnecessarily.
1. Button.java 2. NewPatronView.java 3. EndgamePrompt.java 4. AddPartyView.java 5. ControlDeskView.java 6. LaneStatusView,java 7. ViewLane.java 8. ViewPinSetter	• Several functions utilizing the same code for implementing buttons and windows in all these classes, should be replaced with a single class whose object gets called.

Narrative of Refactored Code

We first read through the entire codebase and made UML diagrams for the same. This helped us understand the inter dependencies and relations between the various classes. We could immediately pin-point the major problems in the code - they were the files we had the hardest time reading! Be it their never-ending length or calling umpteen functions, passing lots of parameters. We decided to start off refactoring by tackling these major issues.

1. **Creating SuperLane - Super Class of Lane:**
   Lane.java was the most problematic file as it showed high complexity, coupling but very low cohesion. It was also abnormally large in size.
   We created a Super Class for it and made Lane.Java inherit from this file. All subscriber and publish related functions were moved to the Super Class. As similar variables and functions were in both classes, cohesion increased. We also separated the biggest functions - run() in Lane and getScore() in SuperLane. This decreased cohesion.

2. **Separating ControlDeskView into different classes:**
   ControlDeskView had low cohesion and coupling as it had 3 major separate things going on - creation of Control Panel, Lane Panel, and Party Panel. We split these functionalities into 3 different files ControlDeskViewControl, ControlDeskViewParty, and ControlDeskViewLane. This increased cohesion as we moved the variables and UI creation functions to the relevant classes. There were also many functions passing ControlDeskView parameters. As we moved them to the respective files, coupling improved.

3. **Creating a Views Class**
   A lot of the GUI code was repeated in all the 'View' Classes. We made a class called Views which deals with that. It has generic code for buttons, panels, and windows. This also made our code extensible as we saw when we implemented the new features. We could use the code for these UI functions immediately rather than having to write them from scratch.
   We also experimented with adding the action listener in 'Views' itself but that increased coupling. We reckon this may be because we had to pass the entire Master Class so we did not include this.

4. **Creating the getScore Class & Improving logic**
   The longest method with the most '**McCabe's Cyclomatic Complexity**' in the entire project was getScore.
   Our first step was to shift this entire function to another file. However to decrease coupling, we altered the code in Lane.Java to pass just 2 parameters instead of 5 as the immediate glance would suggest. This also increased cohesion mildly as not-so-relevant variables and functions were removed from Lane.java.
   We also improved the logic for calculating the score in this new Class. It was vastly complex earlier with many if-else conditions. We read the rules of bowling and implemented our own functions such as handleStrike() and handleNormalThrow(). This reduced complexity as we also broke down the seemingly long if-else conditions into simpler shorter versions.

5. **Creating ControlDeskSubscriber**
   ControlDesk.java had very low cohesion initially. This may be because it was performing a great variety of actions - broad and unfocused. We shifted all the variables and functions relevant to' subscribe' and 'publish' functions to a new Class. This increased cohesion as we separated drastically different features into separate files retaining the similar ones together.

6. **Shifting Panels, Buttons, Windows variables**
   A lot of the View files unnecessarily had such UI features as private variables. Since we made a Views Class, creation of buttons was handled there and we could remove the variables from the Major View classes. This improved cohesion and decreased coupling as we would not return the buttons and panels from the Views class.

7. **Shift registerPatron to BowlerFile from ControlDesk**
   As ControlDesk had low cohesion initially, we moved less related functions to other files. An example is registerPatron() function to BowlerFile as it is more relevant there. This increased cohesion.

8. **Loops, loops, loops!**
   Many places had repetitive code for the same functionality. One such example is **button creation**. We replaced this with loops so that 8-10 lines were reduced to just 3 everywhere.

9. **If-else conditions**
   A lot of the if-else conditions had repetitive checks or were not written in the most optimized fashion. Examples are LaneView and LaneStatusView. We optimized the conditional statements and also moved the more likely statements at a higher position so that conditions are met earlier. Every little counts! This led to the decrease in Cyclomatic Complexity in all our files.

10. **Toggle**
    Many files had separate functions for show(), hide() and pause(), unPause() etc. We replaced these with toggle functions that would just change the state of the variable to !variable in case of boolean or we would pass a bool value to do the same. This was done for EndGamePrompt, EndGameReport, LaneView, NewPatronView and so on.

11. **Using Java Generics**
    In a lot of places we have used Java Generics. This made our code reusable as we were not declaring the variable type beforehand and this allowed us to pass any parentClass. This helped us especially with the actionListener functions as we did not have to write the parent class name every time. We also replaced HashSet and Vector with the general var with return functions.

12. **Removing unused & unnecessary variables, functions, inconsequential statements**
    The original code was flooded with these. We used eciplse's 'view usage' feature to check where all each function or variable was being called. Unused

variables were removed outrightly. Variables that were just passed once or some global variables that were being used in only 1 method of the class - we made such variables local to the methods in which they were being used. Functions that were just called once and had just a line or two were made inline. We also removed statements that had no impact on the code.
Few examples are

1. getNickName function from Score.java, Bowler.java
2. Destroy function in LaneEvent.java
3. cdv variable from Alley.java
4. bowlInt, cur from LaneView
5. foulVariable in LaneStatusView
6. egp = NULL statement
7. Lock variable in AddPartyView
8. Double assignments were removed in many files
9. selectedNick and selectedMembers from EndGamePoint
10. extra sendln method in ScoreReport.java

It's a ***STRIKE***! So we'll stop here, but there are many more.

13. **Replacing deprecated Functions**
    Many deprecated functions and variables existed in the code. Deprecated functions such as .show() and .hide() were replaced with setVisible(boolean). String concatenation was also replaced.

Analysis of Refactored Design

The refactored code has a good blend among competing criteria.
Now we dive into the details for each parameter and how we improved.

Low Coupling

We started out with 29 classes which are already a lot! The dependencies between these were extensive and we got rid of these by somehow making the parameters passed local and removing the redundant ones.
We still extended our class list to include new features and also to break down large files such as Lane into getScore and SuperLane. We ensured that these classes were standalone and did not require too many other dependencies as that increases coupling. We also ensured that functions were in their relevant places so that they did not have to communicate or move around too much.
Implementations 1, 2, 3, 4, and 6 of the previous section handle this.

High Cohesion

Cohesion refers to consistency and organization of different units. Longer classes had a problem with cohesion as they had numerous methods which often became unrelated and too broad. We split such classes eg. ControlPanelView Split **(implementation 2)** and moved around their functions (like **implementation 7** above). **Implementations 1, 3, 4, 5** also helped us achieve this.

Separation of Concerns

We achieved this by separating the biggest concerns or functionalities in our case. When too many concerns were in one file, we broke them down into classes such as we did with **Lane.Java**. We ensured least overlap between these concerns so that if one gets affected, the other does not. OOPs concepts and decoupling helped us achieve this. **Implementation 1 and 2** are the best example of this.

Information hiding

Data Abstraction or Information hiding is a crucial aspect of OOPS. We implemented this by using private variables in all our new classes. We made getter and setter functions to read and alter these.

Law of Demeter

The fundamental notion of the LoD is that a given object should assume as little as possible about the structure or properties of anything else (including its subcomponents). That is what we have done by ensuring low coupling and high cohesion. We have tried to make the classes as independent as possible so they have very few neighbours.

Extensibility

As we ensured low coupling, we made sure that it was easier to introduce new module. As we wrote basic code for the UI, we could easily extend it to add specific features in the respective classes. High cohesion also ensured that as we extended the application, cohesion is not too affected as all relevant features only are included in each file. For eg: the module for database search was easy to implement. **Implementation 11** is quintessential to this.

Reusability

We made our code reusable by adding the Views class. This allowed us to reuse our buttons and panels everywhere instead of writing them all over again. We also

implemented the pause/resume feature using the maintenance call features. **Implementations 3 and 11** helped us.

Metrics Analysis

Metrics Before Refactoring

Evaluation on metrics2

The following are the metrics calculated on metrics2 plugin on the original code. We collected the **max Range values from jArchitect manually** and set them in our eclipse application. This is why our colours look a lot more red than others' since we got the accurate ranges. This allowed us to monitor all of our changes and their impacts.

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

Evaluation on CodeMR

List of all classes (#29)

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	Lane					227	medium-high	low-medium	medium-high	low-medium
2	ControlDeskView					87	low-medium	low-medium	low-medium	low-medium
3	ControlDesk					68	low-medium	low-medium	medium-high	low-medium
4	LaneStatusView					93	low	low-medium	low-medium	low-medium
5	LaneView					140	low-medium	low	low-medium	low-medium
6	AddPartyView					127	low-medium	low	low-medium	low-medium
7	PinSetterView					111	low	low	low	low-medium
8	NewPatronView					85	low	low	low	low-medium
9	EndGameReport					79	low	low	low-medium	low-medium
10	ScoreReport					76	low	low	low	low-medium
11	EndGamePrompt					55	low	low	low	low-medium
12	Pinsetter					47	low	low	low	low
13	LaneEvent					41	low	low	medium-high	low

What these values taught us

The analysis of each parameter and code smells has been done in the above sections. These metrics just lay a number on what was wrong in the code and cemented our beliefs. We could see what was going wrong in our code and test our various implementations. Whilst refactoring our code, we realized that metrics often compete with each other and it is not a linear path. Whenever we modified anything, there would be some kind of tradeoff. Striking the right balance was essential!

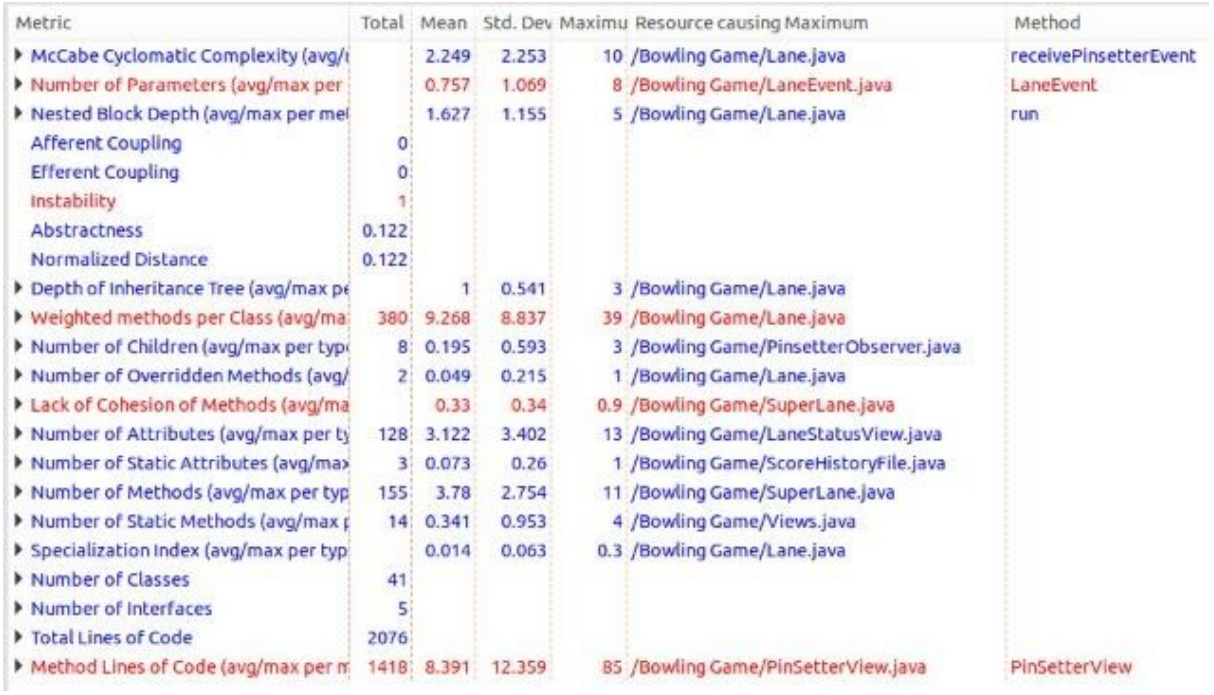
How we used these measurements to guide our refactoring’

After every change, we would first check the functionality of our game to ensure everything was correct. Next we would study all the features metrics in the 2 IDEs. We would understand the impact it had on parameters such as coupling, cohesion, and the most troublesome of all, cyclomatic complexity! The yellow dots in CodeMR were especially troublesome and we focused on removing those as those were the worst parts of our code. These visual and numeric illustrations helped us achieve the

code quality we aspired for as we were not left speculating about our code, but could see and measure its actual worth!

Metrics After Refactoring

Evaluation on metrics2



Metric	Total	Mean	Std. Dev	Maximu	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/		2.249	2.253	10	/Bowling Game/Lane.java	receivePinsetterEvent
Number of Parameters (avg/max per		0.757	1.069	8	/Bowling Game/LaneEvent.java	LaneEvent
Nested Block Depth (avg/max per met		1.627	1.155	5	/Bowling Game/Lane.java	run
Afferent Coupling	0					
Efferent Coupling	0					
Instability	1					
Abstractness	0.122					
Normalized Distance	0.122					
Depth of Inheritance Tree (avg/max pe		1	0.541	3	/Bowling Game/Lane.java	
Weighted methods per Class (avg/ma	380	9.268	8.837	39	/Bowling Game/Lane.java	
Number of Children (avg/max per typ	8	0.195	0.593	3	/Bowling Game/PinsetterObserver.java	
Number of Overridden Methods (avg/	2	0.049	0.215	1	/Bowling Game/Lane.java	
Lack of Cohesion of Methods (avg/ma		0.33	0.34	0.9	/Bowling Game/SuperLane.java	
Number of Attributes (avg/max per ty	128	3.122	3.402	13	/Bowling Game/LaneStatusView.java	
Number of Static Attributes (avg/max	3	0.073	0.26	1	/Bowling Game/ScoreHistoryFile.java	
Number of Methods (avg/max per typ	155	3.78	2.754	11	/Bowling Game/SuperLane.java	
Number of Static Methods (avg/max p	14	0.341	0.953	4	/Bowling Game/Views.java	
Specialization Index (avg/max per typ		0.014	0.063	0.3	/Bowling Game/Lane.java	
Number of Classes	41					
Number of Interfaces	5					
Total Lines of Code	2076					
Method Lines of Code (avg/max per m	1418	8.391	12.359	85	/Bowling Game/PinSetterView.java	PinSetterView

Evaluation on CodeMR

List of all classes (#41)

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	Lane					126	low-medium	low-medium	medium-high	low-medium
2	ControlDeskView					62	low-medium	low-medium	low-medium	low-medium
3	ControlDesk					48	low-medium	low-medium	low	low
4	LaneStatusView					78	low	low-medium	low-medium	low-medium
5	LaneView					132	low-medium	low	low-medium	low-medium
6	AddPartyView					96	low-medium	low	low-medium	low-medium
7	getScore					61	low-medium	low	low	low-medium
8	SuperLane					51	low-medium	low	low	low-medium
9	LaneEvent					23	low-medium	low	low-medium	low
10	PinSetterView					110	low	low	low	low-medium
11	QueryView					93	low	low	low	low-medium
12	OldGameView					83	low	low	low	low-medium

How refactoring affected our metrics

No, we could not get ALL metrics within the range. But we are proud to say that we improved the code substantially and **eliminated a lot of reds**!

Because of the metrics2 plugin, we realized that good quality code is not a linear path and there are tradeoffs to be made. We have to be wise and choose the maximum benefit we can get from a refactor.
The contributions that have led us to this stage are all written in the above **Refactoring section** along with the **feature improvement**.
As for what all we improved, the attached details of the previously red features are shown below:

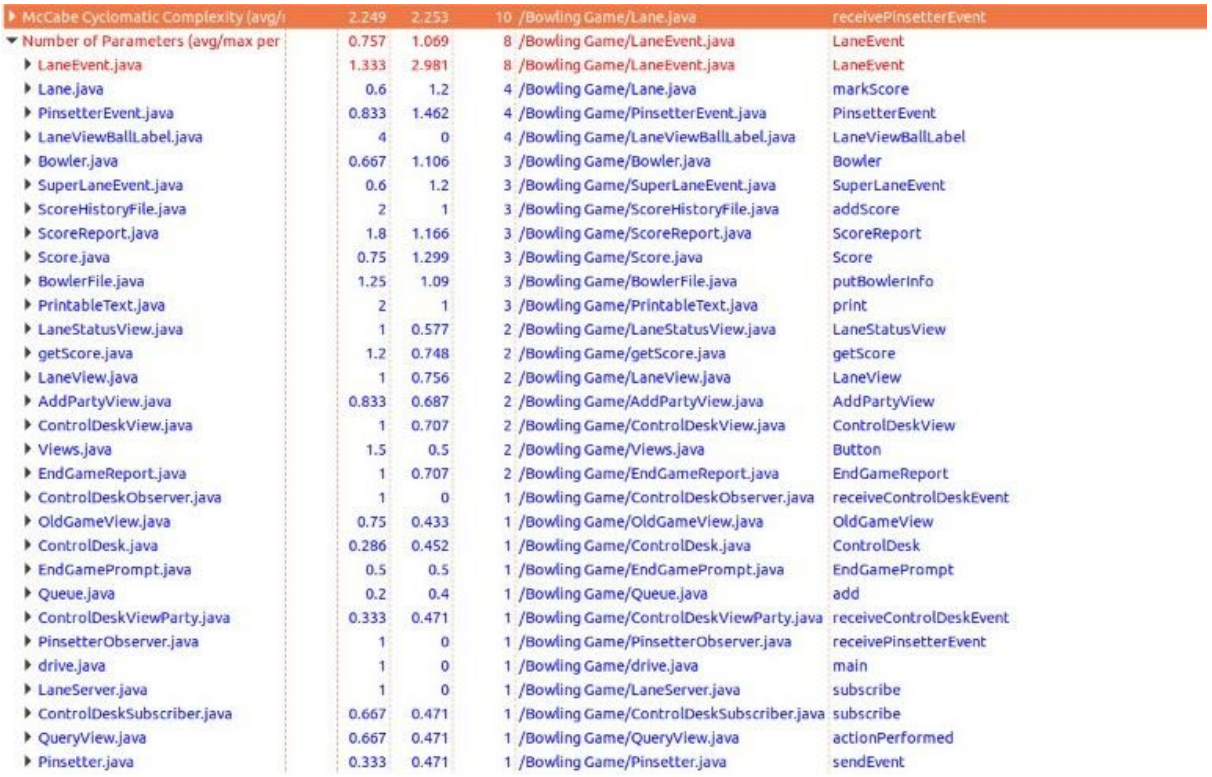
- We got ALL cyclomatic complexities down to within the range (equal to or less than 10)
- We improved the method lines of code significantly. Whilst some may be still red, we have decreased the numbers A LOT.
- Weighted methods per class also reduced significantly as we broke down every long method.
- Also as you can see from our CodeMR graphs, we made the cohesion yellow to green in ControlDesk. We also removed the yellow for complexity in Lane. We improved coupling in LaneView and got rid of the yellow for cohesion in LaneEvent.
- We could not get number of parameters of LaneEvent.java despite trying a lot. This is because its functions had all interdependent features. A potential way of doing this may be to create other classes objects holding these parameters, but we believe that is too complicated for such a simple class and would increase coupling!

A lot of ups and downs were seen during refactoring, but all in all, we can say that the code has improved and gotten simpler and better. The details are attached here:

Metric	Total	Mean	Std. Dev	Maximu	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/		2.249	2.253	10	/Bowling Game/Lane.java	receivePinsetterEvent
Lane.java		3.9	3.239	10	/Bowling Game/Lane.java	receivePinsetterEvent
getScore.java		7	3.521	10	/Bowling Game/getScore.java	getScore
LaneView.java		4.571	3.959	10	/Bowling Game/LaneView.java	receiveLaneEvent
AddPartyView.java		3.5	3.041	10	/Bowling Game/AddPartyView.java	actionPerformed
LaneViewBallLabel.java		10	0	10	/Bowling Game/LaneViewBallLabel.java	LaneViewBallLabel
Pinsetter.java		2.5	2.062	7	/Bowling Game/Pinsetter.java	ballThrown
PinSetterView.java		2.75	2.487	7	/Bowling Game/PinSetterView.java	receivePinsetterEvent
LaneStatusView.java		2.333	1.795	6	/Bowling Game/LaneStatusView.java	ButtonCheck
QueryView.java		3.333	1.886	6	/Bowling Game/QueryView.java	actionPerformed
OldGameView.java		3.75	1.299	5	/Bowling Game/OldGameView.java	actionPerformed
ControlDeskView.java		3	2	5	/Bowling Game/ControlDeskView.java	actionPerformed
findmax.java		3	2	5	/Bowling Game/findmax.java	getDescScores
ControlDesk.java		2.143	0.99	4	/Bowling Game/ControlDesk.java	assignLane
Bowler.java		1.5	1.118	4	/Bowling Game/Bowler.java	equals
ScoreReport.java		2.6	0.8	4	/Bowling Game/ScoreReport.java	ScoreReport
PrintableText.java		2.5	1.5	4	/Bowling Game/PrintableText.java	print
EndGamePrompt.java		2	1	3	/Bowling Game/EndGamePrompt.java	actionPerformed
PinsetterEvent.java		1.5	0.764	3	/Bowling Game/PinsetterEvent.java	totalPinsDown
ScoreHistoryFile.java		2	1	3	/Bowling Game/ScoreHistoryFile.java	getScores
NewPatronView.java		1.333	0.745	3	/Bowling Game/NewPatronView.java	actionPerformed
BowlerFile.java		2.25	0.829	3	/Bowling Game/BowlerFile.java	registerPatron
SuperLane.java		1.364	0.771	3	/Bowling Game/SuperLane.java	resetScores
EndGameReport.java		2.25	0.829	3	/Bowling Game/EndGameReport.java	actionPerformed
ControlDeskSubscriber.java		1.333	0.471	2	/Bowling Game/ControlDeskSubscriber.java	publish
ControlDeskViewLane.java		1.5	0.5	2	/Bowling Game/ControlDeskViewLane.java	ControlDeskViewLane
ControlDeskObserver.java		1	0	1	/Bowling Game/ControlDeskObserver.java	receiveControlDeskEvent
Queue.java		1	0	1	/Bowling Game/Queue.java	Queue
ControlDeskViewParty.java		1	0	1	/Bowling Game/ControlDeskViewParty.java	ControlDeskViewParty
LaneEventInterface.java		1	0	1	/Bowling Game/LaneEventInterface.java	getFrameNum
PinsetterObserver.java		1	0	1	/Bowling Game/PinsetterObserver.java	receivePinsetterEvent
SuperLaneEvent.java		1	0	1	/Bowling Game/SuperLaneEvent.java	SuperLaneEvent

Metric	Total	Mean	Std. Dev	Maximu	Resource causing Maximum	Method
Number of Classes	41					
Number of Interfaces	5					
Total Lines of Code	2076					
Method Lines of Code (avg/max per m	1418	8.391	12.359	85	/Bowling Game/PinSetterView.java	PinSetterView
PinSetterView.java	107	26.75	34.383	85	/Bowling Game/PinSetterView.java	PinSetterView
QueryView.java	90	30	25.703	65	/Bowling Game/QueryView.java	QueryView
LaneView.java	132	18.857	19.831	59	/Bowling Game/LaneView.java	makeFrame
AddPartyView.java	96	16	15.166	44	/Bowling Game/AddPartyView.java	AddPartyView
EndGameReport.java	53	13.25	13.33	36	/Bowling Game/EndGameReport.java	EndGameReport
LaneStatusView.java	70	11.667	12.051	34	/Bowling Game/LaneStatusView.java	LaneStatusView
OldGameView.java	85	21.25	10.329	33	/Bowling Game/OldGameView.java	actionPerformed
Lane.java	130	13	9.869	29	/Bowling Game/Lane.java	gameEnded
NewPatronView.java	43	7.167	9.737	27	/Bowling Game/NewPatronView.java	NewPatronView
LaneViewBallLabel.java	27	27	0	27	/Bowling Game/LaneViewBallLabel.java	LaneViewBallLabel
EndGamePrompt.java	40	10	9.028	25	/Bowling Game/EndGamePrompt.java	EndGamePrompt
getScore.java	70	14	9.798	25	/Bowling Game/getScore.java	handleStrike
ControlDeskView.java	55	13.75	8.584	25	/Bowling Game/ControlDeskView.java	ControlDeskView
ScoreReport.java	68	13.6	7.06	24	/Bowling Game/ScoreReport.java	sendEmail
Pinsetter.java	42	7	6.952	22	/Bowling Game/Pinsetter.java	ballThrown
findmax.java	20	10	9	19	/Bowling Game/findmax.java	getDescScores
BowlerFile.java	39	9.75	4.437	17	/Bowling Game/BowlerFile.java	getBowlerInfo
PrintableText.java	19	9.5	7.5	17	/Bowling Game/PrintableText.java	print
ControlDeskViewLane.java	16	8	7	15	/Bowling Game/ControlDeskViewLane.java	ControlDeskViewLane
ControlDeskViewParty.java	14	4.667	5.185	12	/Bowling Game/ControlDeskViewParty.java	ControlDeskViewParty
Bowler.java	18	3	3.651	11	/Bowling Game/Bowler.java	equals
ScoreHistoryFile.java	16	8	3	11	/Bowling Game/ScoreHistoryFile.java	getScores
SuperLane.java	31	2.818	2.724	10	/Bowling Game/SuperLane.java	resetScores
ControlDesk.java	42	6	3.295	9	/Bowling Game/ControlDesk.java	assignLane
Views.java	24	6	1.414	8	/Bowling Game/Views.java	MiniPanel
PinsetterEvent.java	18	3	2.828	7	/Bowling Game/PinsetterEvent.java	PinsetterEvent
ControlDeskViewControl.java	8	4	3	7	/Bowling Game/ControlDeskViewControl.jav	ControlDeskViewControl
LaneEvent.java	11	1.833	1.863	6	/Bowling Game/LaneEvent.java	LaneEvent

Metric	Total	Mean	Std. Dev	Maximu	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/		2.249	2.253	10	/Bowling Game/Lane.java	receivePinsetterEvent
Number of Parameters (avg/max per		0.757	1.069	8	/Bowling Game/LaneEvent.java	LaneEvent
Nested Block Depth (avg/max per mel		1.627	1.155	5	/Bowling Game/Lane.java	run
Afferent Coupling	0					
Efferent Coupling	0					
Instability	1					
Abstractness	0.122					
Normalized Distance	0.122					
Depth of Inheritance Tree (avg/max pe		1	0.541	3	/Bowling Game/Lane.java	
Weighted methods per Class (avg/ma	380	9.268	8.837	39	/Bowling Game/Lane.java	
Number of Children (avg/max per typ	8	0.195	0.593	3	/Bowling Game/PinsetterObserver.java	
Number of Overridden Methods (avg/	2	0.049	0.215	1	/Bowling Game/Lane.java	
Lack of Cohesion of Methods (avg/ma		0.33	0.34	0.9	/Bowling Game/SuperLane.java	
SuperLane.java		0.9	0	0.9	/Bowling Game/SuperLane.java	
OldGameView.java		0.815	0	0.815	/Bowling Game/OldGameView.java	
SuperLaneEvent.java		0.812	0	0.812	/Bowling Game/SuperLaneEvent.java	
LaneEvent.java		0.8	0	0.8	/Bowling Game/LaneEvent.java	
NewPatronView.java		0.778	0	0.778	/Bowling Game/NewPatronView.java	
PinsetterEvent.java		0.75	0	0.75	/Bowling Game/PinsetterEvent.java	
LaneStatusView.java		0.738	0	0.738	/Bowling Game/LaneStatusView.java	
LaneView.java		0.738	0	0.738	/Bowling Game/LaneView.java	
QueryView.java		0.714	0	0.714	/Bowling Game/QueryView.java	
AddPartyView.java		0.667	0	0.667	/Bowling Game/AddPartyView.java	
Lane.java		0.667	0	0.667	/Bowling Game/Lane.java	
PinSetterView.java		0.625	0	0.625	/Bowling Game/PinSetterView.java	
ControlDesk.java		0.6	0	0.6	/Bowling Game/ControlDesk.java	
EndGameReport.java		0.6	0	0.6	/Bowling Game/EndGameReport.java	
Pinsetter.java		0.56	0	0.56	/Bowling Game/Pinsetter.java	
ControlDeskView.java		0.556	0	0.556	/Bowling Game/ControlDeskView.java	
Bowler.java		0.533	0	0.533	/Bowling Game/Bowler.java	
EndGamePrompt.java		0.5	0	0.5	/Bowling Game/EndGamePrompt.java	
ControlDeskViewParty.java		0.5	0	0.5	/Bowling Game/ControlDeskViewParty.java	



McCabe Cyclomatic Complexity (avg/	2.249	2.253	10	/Bowling Game/Lane.java	receivePinsetterEvent
Number of Parameters (avg/max per	0.757	1.069	8	/Bowling Game/LaneEvent.java	LaneEvent
LaneEvent.java	1.333	2.981	8	/Bowling Game/LaneEvent.java	LaneEvent
Lane.java	0.6	1.2	4	/Bowling Game/Lane.java	markScore
PinsetterEvent.java	0.833	1.462	4	/Bowling Game/PinsetterEvent.java	PinsetterEvent
LaneViewBallLabel.java	4	0	4	/Bowling Game/LaneViewBallLabel.java	LaneViewBallLabel
Bowler.java	0.667	1.106	3	/Bowling Game/Bowler.java	Bowler
SuperLaneEvent.java	0.6	1.2	3	/Bowling Game/SuperLaneEvent.java	SuperLaneEvent
ScoreHistoryFile.java	2	1	3	/Bowling Game/ScoreHistoryFile.java	addScore
ScoreReport.java	1.8	1.166	3	/Bowling Game/ScoreReport.java	ScoreReport
Score.java	0.75	1.299	3	/Bowling Game/Score.java	Score
BowlerFile.java	1.25	1.09	3	/Bowling Game/BowlerFile.java	putBowlerInfo
PrintableText.java	2	1	3	/Bowling Game/PrintableText.java	print
LaneStatusView.java	1	0.577	2	/Bowling Game/LaneStatusView.java	LaneStatusView
getScore.java	1.2	0.748	2	/Bowling Game/getScore.java	getScore
LaneView.java	1	0.756	2	/Bowling Game/LaneView.java	LaneView
AddPartyView.java	0.833	0.687	2	/Bowling Game/AddPartyView.java	AddPartyView
ControlDeskView.java	1	0.707	2	/Bowling Game/ControlDeskView.java	ControlDeskView
Views.java	1.5	0.5	2	/Bowling Game/Views.java	Button
EndGameReport.java	1	0.707	2	/Bowling Game/EndGameReport.java	EndGameReport
ControlDeskObserver.java	1	0	1	/Bowling Game/ControlDeskObserver.java	receiveControlDeskEvent
OldGameView.java	0.75	0.433	1	/Bowling Game/OldGameView.java	OldGameView
ControlDesk.java	0.286	0.452	1	/Bowling Game/ControlDesk.java	ControlDesk
EndGamePrompt.java	0.5	0.5	1	/Bowling Game/EndGamePrompt.java	EndGamePrompt
Queue.java	0.2	0.4	1	/Bowling Game/Queue.java	add
ControlDeskViewParty.java	0.333	0.471	1	/Bowling Game/ControlDeskViewParty.java	receiveControlDeskEvent
PinsetterObserver.java	1	0	1	/Bowling Game/PinsetterObserver.java	receivePinsetterEvent
drive.java	1	0	1	/Bowling Game/drive.java	main
LaneServer.java	1	0	1	/Bowling Game/LaneServer.java	subscribe
ControlDeskSubscriber.java	0.667	0.471	1	/Bowling Game/ControlDeskSubscriber.java	subscribe
QueryView.java	0.667	0.471	1	/Bowling Game/QueryView.java	actionPerformed
Pinsetter.java	0.333	0.471	1	/Bowling Game/Pinsetter.java	sendEvent

Metric	Total	Mean	Std. Dev	Maximu	Resource causing Maximum	Method
Normalized Distance	0.122					
Depth of Inheritance Tree (avg/max pe		1	0.541	3	/Bowling Game/Lane.java	
Weighted methods per Class (avg/ma	380	9.268	8.837	39	/Bowling Game/Lane.java	
Lane.java	39	39	0	39	/Bowling Game/Lane.java	
getScore.java	35	35	0	35	/Bowling Game/getScore.java	
LaneView.java	32	32	0	32	/Bowling Game/LaneView.java	
AddPartyView.java	21	21	0	21	/Bowling Game/AddPartyView.java	
OldGameView.java	15	15	0	15	/Bowling Game/OldGameView.java	
ControlDesk.java	15	15	0	15	/Bowling Game/ControlDesk.java	
Pinsetter.java	15	15	0	15	/Bowling Game/Pinsetter.java	
SuperLane.java	15	15	0	15	/Bowling Game/SuperLane.java	
LaneStatusView.java	14	14	0	14	/Bowling Game/LaneStatusView.java	
ScoreReport.java	13	13	0	13	/Bowling Game/ScoreReport.java	
ControlDeskView.java	12	12	0	12	/Bowling Game/ControlDeskView.java	
PinSetterView.java	11	11	0	11	/Bowling Game/PinSetterView.java	
QueryView.java	10	10	0	10	/Bowling Game/QueryView.java	
LaneViewBallLabel.java	10	10	0	10	/Bowling Game/LaneViewBallLabel.java	
Bowler.java	9	9	0	9	/Bowling Game/Bowler.java	
PinsetterEvent.java	9	9	0	9	/Bowling Game/PinsetterEvent.java	
LaneEventInterface.java	9	9	0	9	/Bowling Game/LaneEventInterface.java	
BowlerFile.java	9	9	0	9	/Bowling Game/BowlerFile.java	
EndGameReport.java	9	9	0	9	/Bowling Game/EndGameReport.java	
EndGamePrompt.java	8	8	0	8	/Bowling Game/EndGamePrompt.java	
NewPatronView.java	8	8	0	8	/Bowling Game/NewPatronView.java	
LaneEvent.java	6	6	0	6	/Bowling Game/LaneEvent.java	
findmax.java	6	6	0	6	/Bowling Game/findmax.java	
Queue.java	5	5	0	5	/Bowling Game/Queue.java	
SuperLaneEvent.java	5	5	0	5	/Bowling Game/SuperLaneEvent.java	
PrintableText.java	5	5	0	5	/Bowling Game/PrintableText.java	
ScoreHistoryFile.java	4	4	0	4	/Bowling Game/ScoreHistoryFile.java	
ControlDeskSubscriber.java	4	4	0	4	/Bowling Game/ControlDeskSubscriber.java	
Views.java	4	4	0	4	/Bowling Game/Views.java	