Software Engineering

TEAM 22

DESIGN DOCUMENT

Team Members

Date of Submission: 20/02/2022

NAME	ROLL NUMBER	ROLE
Adidala Divishath Reddy	2021201061	Refactoring of Scoring and related files
Mainak Dhara	2021201062	Refactoring of Lane and related files
Jasika Shah	2021201063	Refactoring of Pinsetter and related files
Akshay M	2020201023	Refactoring of Control Desk and related files

The entire project took around 50 hours of effort and everyone had equal contribution in it. UML diagrams can be found here.

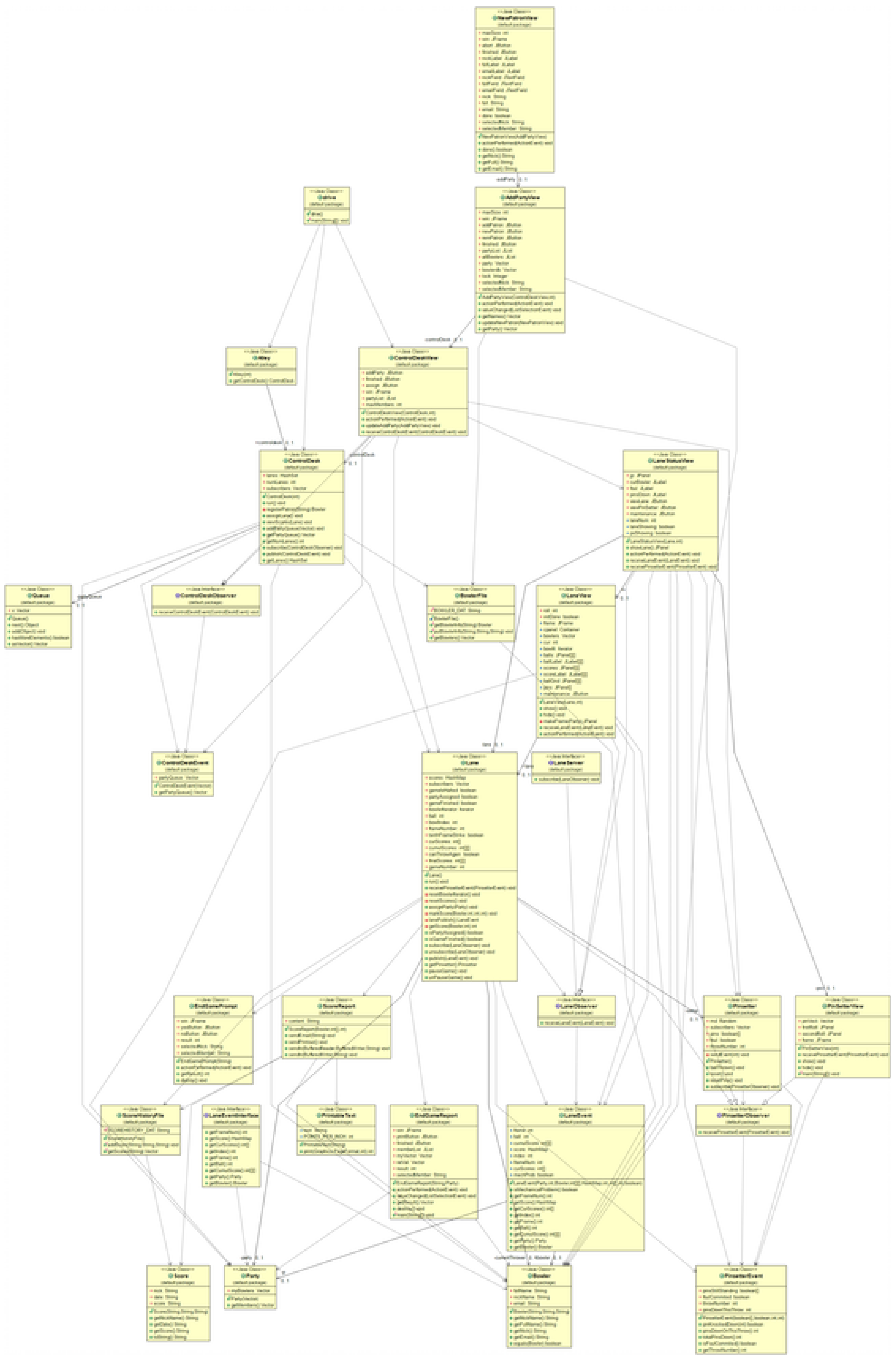
Overview

The project is based on the simulation and management of a Bowling Alley. It is built on an older version of Java. A collection of lanes can be monitored via a control desk in the application, bowler parties can be assigned to those lanes, pinsetter configuration can be viewed, and scores can be printed at the end of the game.

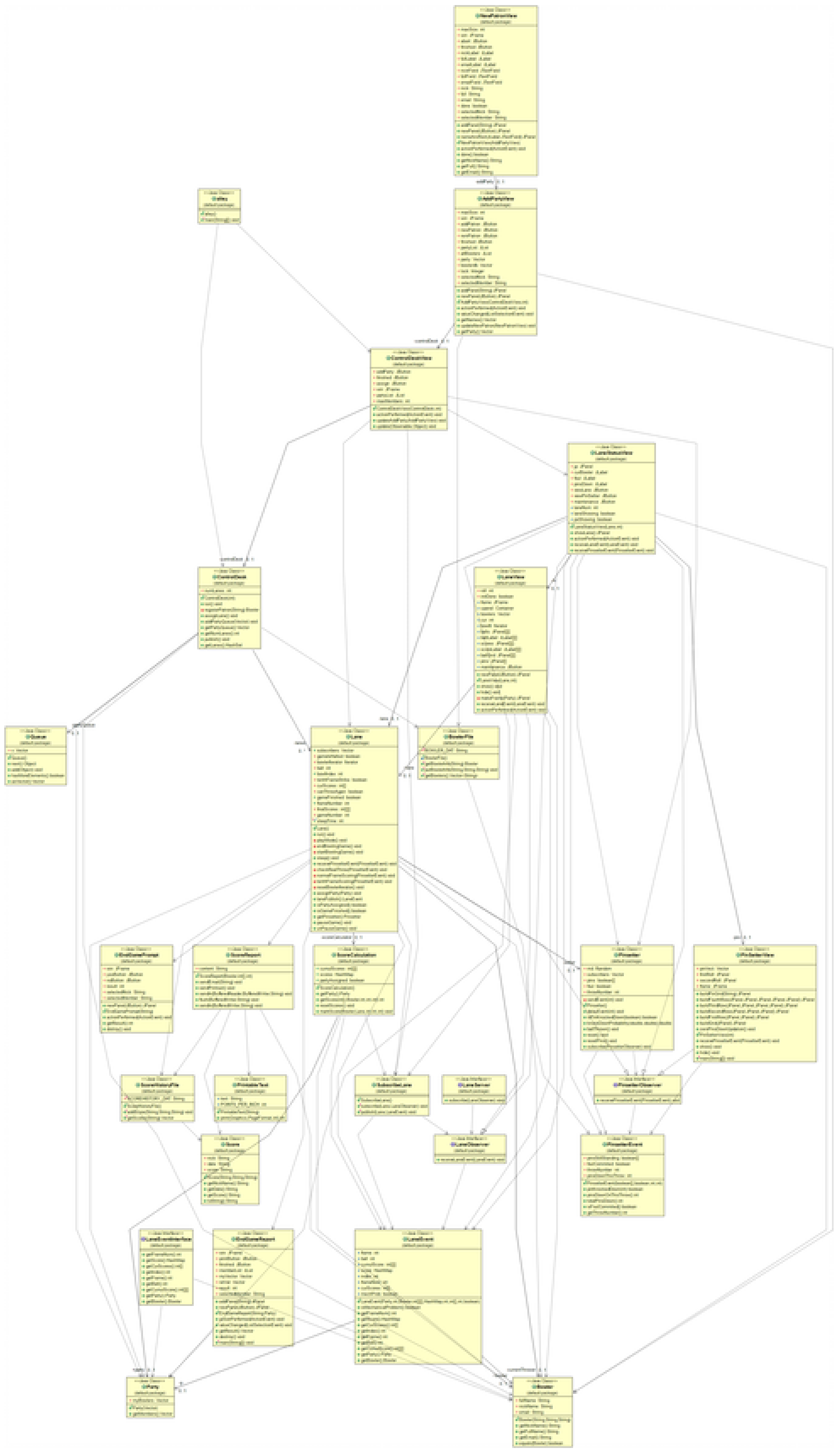
Our goal was to improve the overall design of the source code because the original code base has many design flaws and code smells that demonstrate poor software design principles and antipatterns. We identified potential routines and subroutines for refactoring using metrics and manual code inspection to improve the overall design of the project, making understanding the code and future feature integrations much easier.

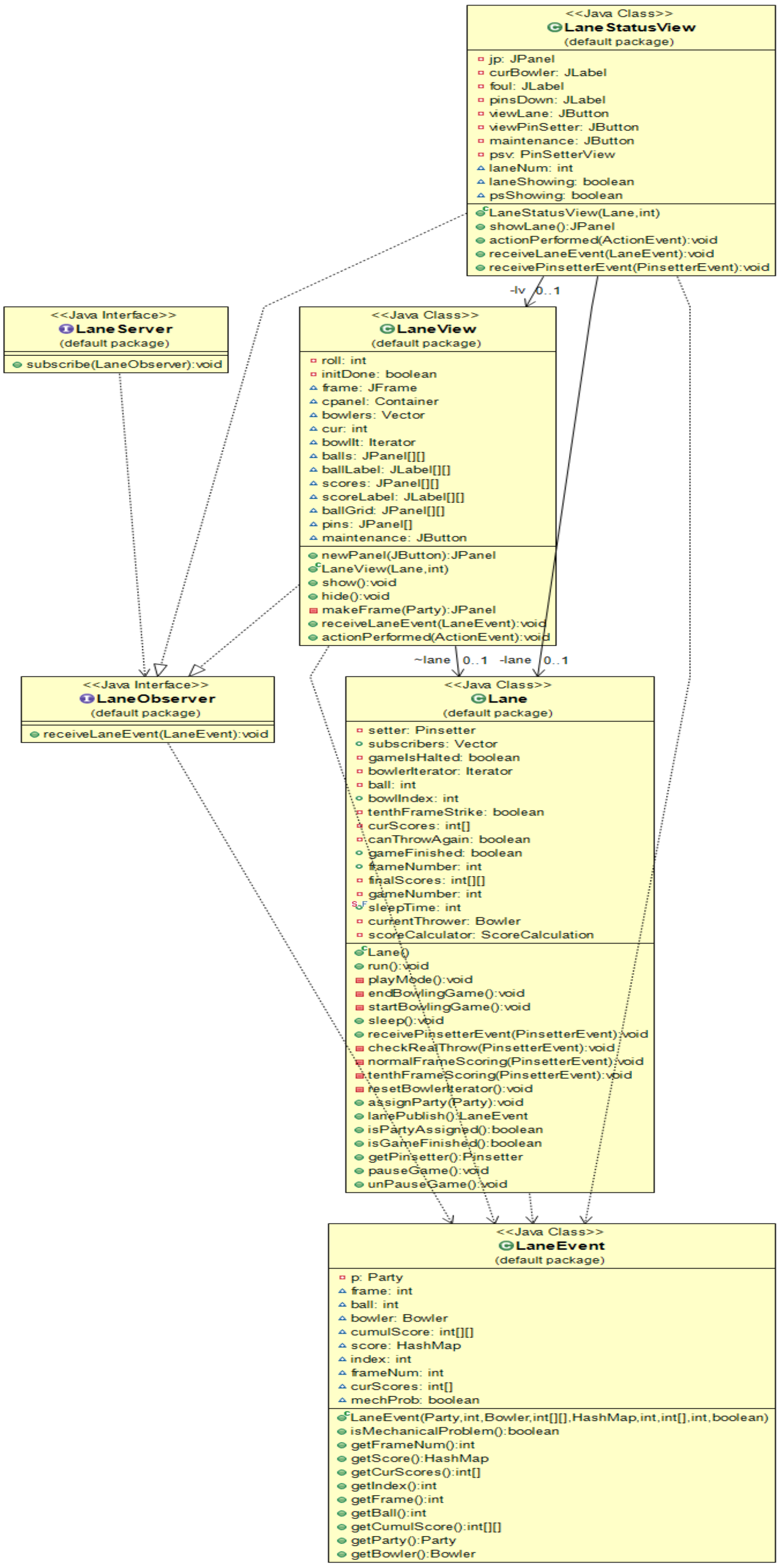
Class Diagrams

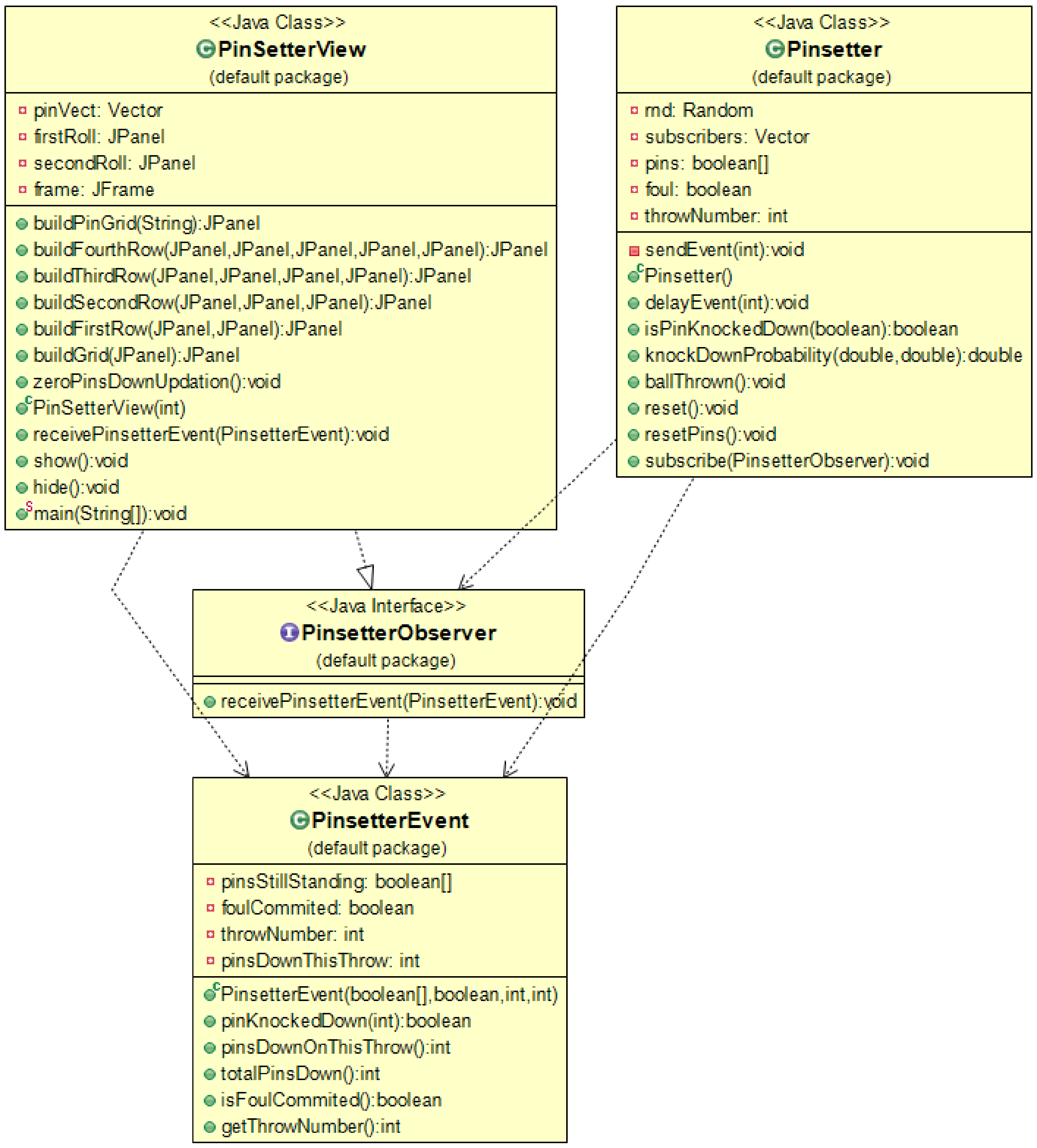
Before Refactoring



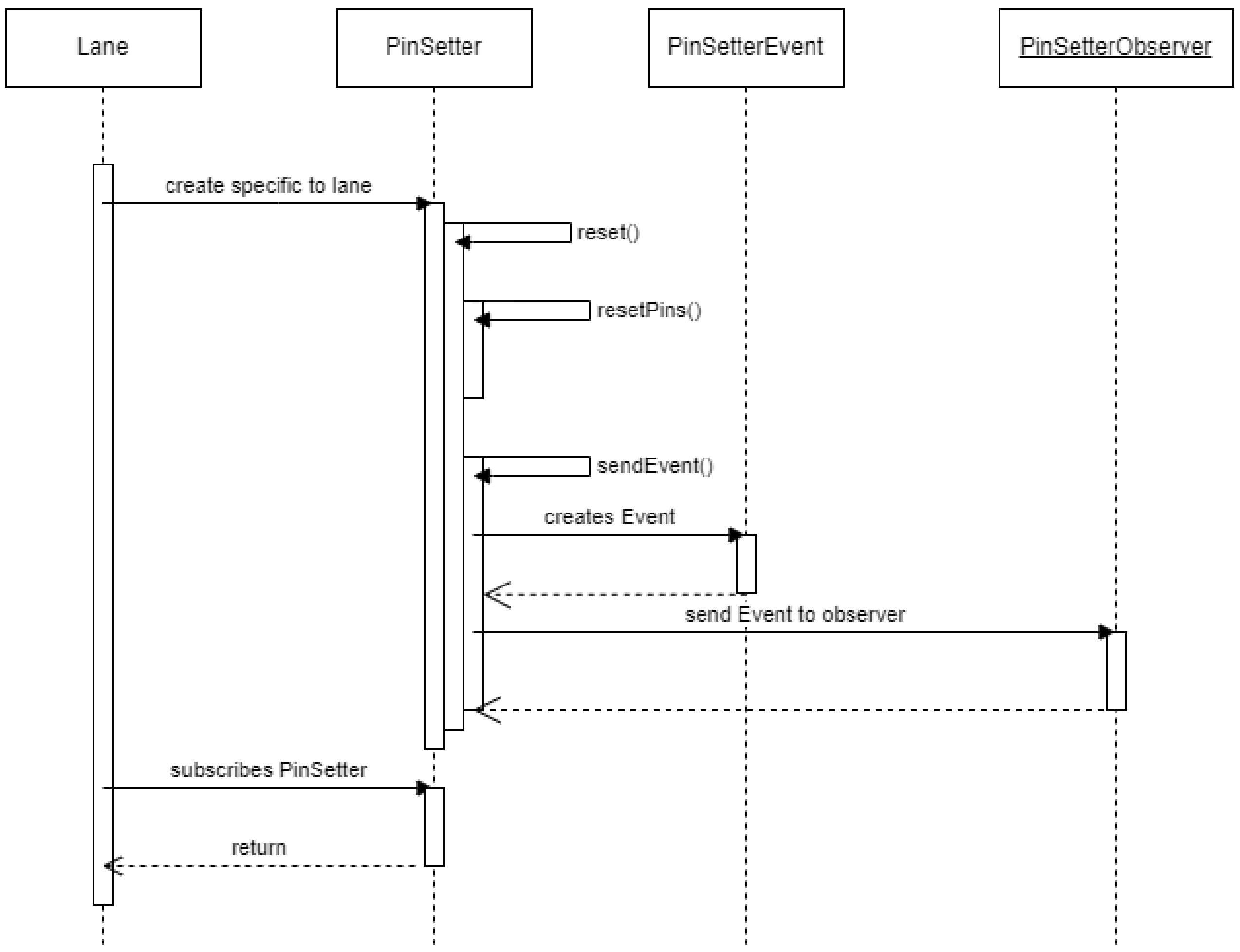
After Refactoring

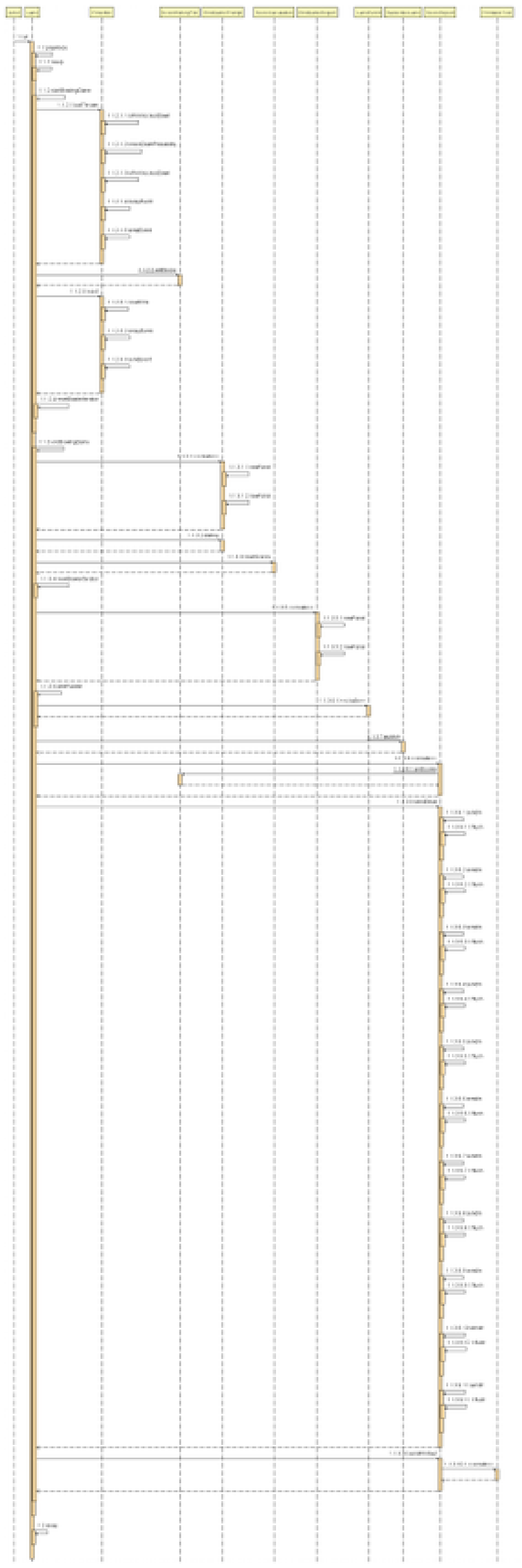



<<Java Class>>
LaneStatusView
(default package)
jp: JPanel
curBowler: JLabel
foul: JLabel
pinsDown: JLabel
viewLane: JButton
viewPinSetter: JButton
maintenance: JButton
psv: PinSetterView
laneNum: int
laneShowing: boolean
psShowing: boolean
LaneStatusView(Lane,int)
showLane():JPanel
actionPerformed(ActionEvent):void
receiveLaneEvent(LaneEvent):void
receivePinsetterEvent(PinsetterEvent):void
-lv
0..1
<<Java Interface>>
LaneServer
(default package)
subscribe(LaneObserver):void
<<Java Class>>
LaneView
(default package)
roll: int
initDone: boolean
frame: JFrame
cpanel: Container
bowlers: Vector
cur: int
bowlIt: Iterator
balls: JPanel[][]
ballLabel: JLabel[][]
scores: JPanel[][]
scoreLabel: JLabel[][]
ballGrid: JPanel[][]
pins: JPanel[]
maintenance: JButton
newPanel(JButton):JPanel
LaneView(Lane,int)
show():void
hide():void
makeFrame(Party):JPanel
receiveLaneEvent(LaneEvent):void
actionPerformed(ActionEvent):void
~lane
0..1
-lane
0..1
<<Java Interface>>
LaneObserver
(default package)
receiveLaneEvent(LaneEvent):void
<<Java Class>>
Lane
(default package)
setter: Pinsetter
subscribers: Vector
gameIsHalted: boolean
bowlerIterator: Iterator
ball: int
bowlIndex: int
tenthFrameStrike: boolean
curScores: int[]
canThrowAgain: boolean
gameFinished: boolean
frameNumber: int
finalScores: int[][]
gameNumber: int
sleepTime: int
currentThrower: Bowler
scoreCalculator: ScoreCalculation
Lane()
run():void
playMode():void
endBowlingGame():void
startBowlingGame():void
sleep():void
receivePinsetterEvent(PinsetterEvent):void
checkRealThrow(PinsetterEvent):void
normalFrameScoring(PinsetterEvent):void
tenthFrameScoring(PinsetterEvent):void
resetBowlerIterator():void
assignParty(Party):void
lanePublish():LaneEvent
isPartyAssigned():boolean
isGameFinished():boolean
getPinsetter():Pinsetter
pauseGame():void
unPauseGame():void
<<Java Class>>
LaneEvent
(default package)
p: Party
frame: int
ball: int
bowler: Bowler
cumulScore: int[][]
score: HashMap
index: int
frameNum: int
curScores: int[]
mechProb: boolean
LaneEvent(Party,int,Bowler,int[][],HashMap,int,int[],int,boolean)
isMechanicalProblem():boolean
getFrameNum():int
getScore():HashMap
getCurScores():int[]
getIndex():int
getFrame():int
getBall():int
getCumulScore():int[][]
getParty():Party
getBowler():Bowler


<<Java Class>>
PinSetterView
(default package)
pinVect: Vector
firstRoll: JPanel
secondRoll: JPanel
frame: JFrame
buildPinGrid(String):JPanel
buildFourthRow(JPanel,JPanel,JPanel,JPanel,JPanel):JPanel
buildThirdRow(JPanel,JPanel,JPanel,JPanel):JPanel
buildSecondRow(JPanel,JPanel,JPanel):JPanel
buildFirstRow(JPanel,JPanel):JPanel
buildGrid(JPanel):JPanel
zeroPinsDownUpdation():void
PinSetterView(int)
receivePinsetterEvent(PinsetterEvent):void
show():void
hide():void
main(String[]):void
<<Java Class>>
Pinsetter
(default package)
rnd: Random
subscribers: Vector
pins: boolean[]
foul: boolean
throwNumber: int
sendEvent(int):void
Pinsetter()
delayEvent(int):void
isPinKnockedDown(boolean):boolean
knockDownProbability(double,double):double
ballThrown():void
reset():void
resetPins():void
subscribe(PinsetterObserver):void
<<Java Interface>>
PinsetterObserver
(default package)
receivePinsetterEvent(PinsetterEvent):void
<<Java Class>>
PinsetterEvent
(default package)
pinsStillStanding: boolean[]
foulCommited: boolean
throwNumber: int
pinsDownThisThrow: int
PinsetterEvent(boolean[],boolean,int,int)
pinKnockedDown(int):boolean
pinsDownOnThisThrow():int
totalPinsDown():int
isFoulCommited():boolean
getThrowNumber():int

Sequence Diagrams





Class Responsibility Table

Class Name	**Responsibility**
Bowler	Contains information about the bowler (nickname, fullname, email)
BowlerFile	It can be used to interact with the bowler database.
ControlDesk	It assigns Lane to party, adds a party to waiting queue if no lane is free.
EndGamePrompt	Displays the prompt that pops up at the end of the game.
EndGameReport	Displays the game report at the end.
Lane	It creates a new lane and starts it’s thread, assigns party to a lane and marks score for the bowlers on the board.
Party	It maintains the bowlers list who belong to the party
Pinsetter	It handles resetting, simulation and detection of pins appropriately during each throw.
Queue	It creates a new party waiting queue.
ScoreReport	It creates a final score report for all the people in a party and sends the report as an email to each person.

Original Design

Weakness

The original project consists of code and design that conveys poor software design and practices, as well as bugs that affect the application's performance. We used code smells and anti-patterns to describe the system flaws.

Code Smells

Type	Class name	Issue
Large Class	Lane.java	Methods that could potentially be classified as a separate class
Long Methods	AddPartyView.java, Lane.java, PinSetterView.java, LaneStatusView.java	Massive constructors and/or functions that could be broken down into separate functions
Dead/Duplicate code	AddPartyView.java, Lane.java, LaneServer.java, ScoreReport.java	There are a lot of repeating statements that create buttons and other UI elements.
God object	ControlDesk ControlDeskView LaneStatusView ScoreReport	Observers usually have access to a diverse set of classes. For example, LaneStatusView has access to LaneView, Lane, and PinSetterView.
Lazy/Freeloader Class	ControlDeskObserver.java, ControlDeskInterface.java, LaneServer.java, LaneObserver.java, LaneEventInterface.java, PinSetterEvent.java, Queue.java	Custom observer interfaces and classes were created for each type of observer, which could have been avoided by using a single observer interface or one provided by Java.

Feature Envy	Lane.java	Excessively employs methods from various classes

Anti-Patterns

Type	Class name	Issue
Lava Flow	Lane.java	There's a lot of TODO code. Unnecessary stipulation branches. Redundant code.
	LaneServer.java	Didn’t use this class
Repeating Yourself	AddPartyView.java	Statements that are repeated to create buttons and panes
	ScoreReport.java	String code duplication
	PinsetterView.java	Repetitive statements that generate buttons and panes
Functional Decomposition	ControlDeskEvent.java	There is only one method and variable.
Data Class	PinSetterEvent.java LaneEvent.java	Only properties and getter-setter methods are present in this class. Doesn't perform any significant operations on the data
Not Invented Here syndrome	Queue.java	Instead of using the one provided by Java, the project employs a custom queue class.
	All classes that use observable pattern	Instead of using the built-in observer and observable interfaces, I created custom observer interfaces and event classes for each type of observer.

Strengths

1. One of this project's major strengths is that it employs the MVC pattern. View classes communicate with their corresponding model classes and vice versa.
2. It employs the Observable pattern to implement the aforementioned pattern.
3. The majority of the classes' coupling has been kept to a minimum.

Fidelity to the Design Document

The provided project met all of the requirements in the design document and did not deviate from it by adding any unnecessary functionalities.

Narrative of the refactored design

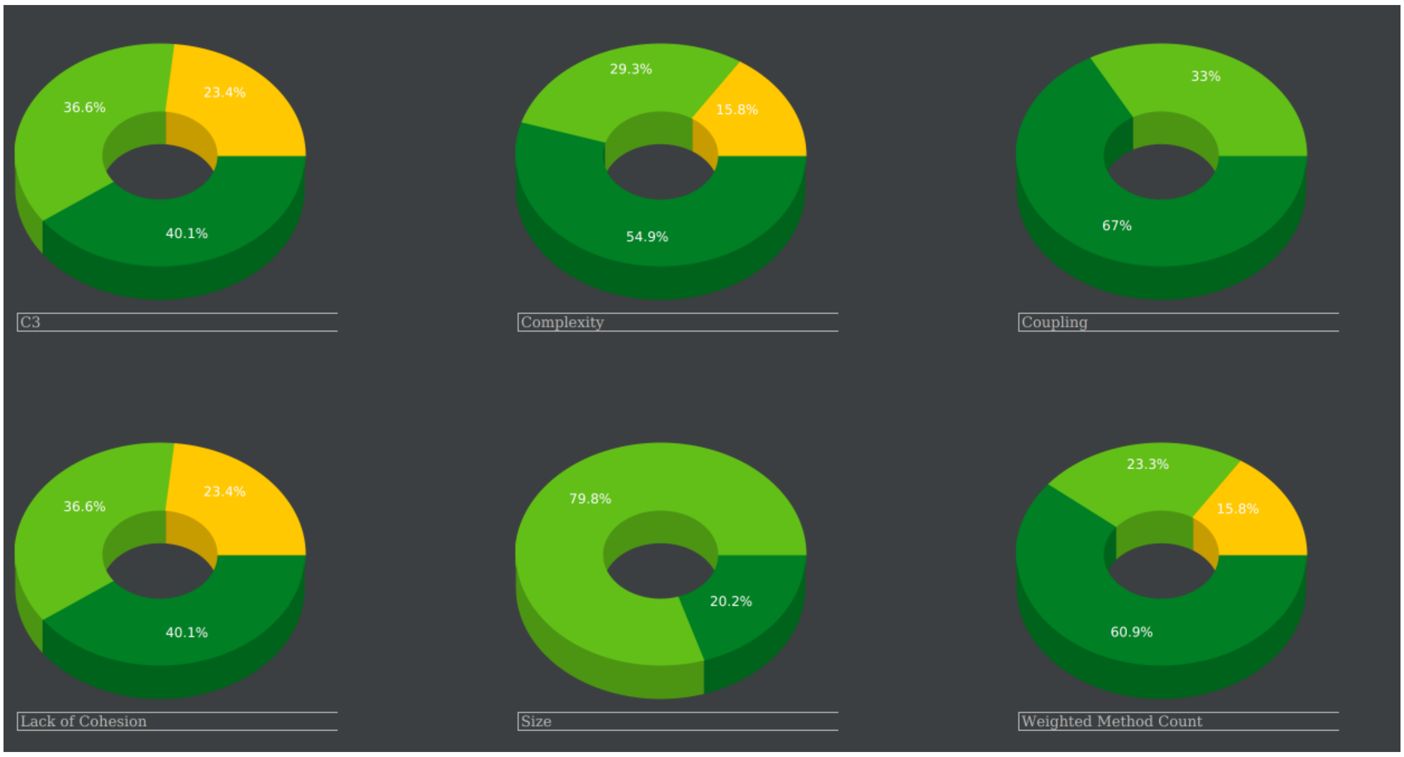
As previously stated, our key refactoring goals were to reduce coupling, promote cohesiveness, and eliminate code smells. While refactoring our code, we used a variety of ways to ensure that all of our Key metrics were met. For example, to reduce complexity and the number of blobs in the original code, we added additional classes: The control desk class in the Bowling Alley game is the center of the game. The control desk assigns a new Lane to a new party as soon as it is established, allowing them to add members to their game. This class also has functions for displaying names on the control panel. We implemented Observer pattern , singleton pattern, adapter pattern and following changes were made to de code,

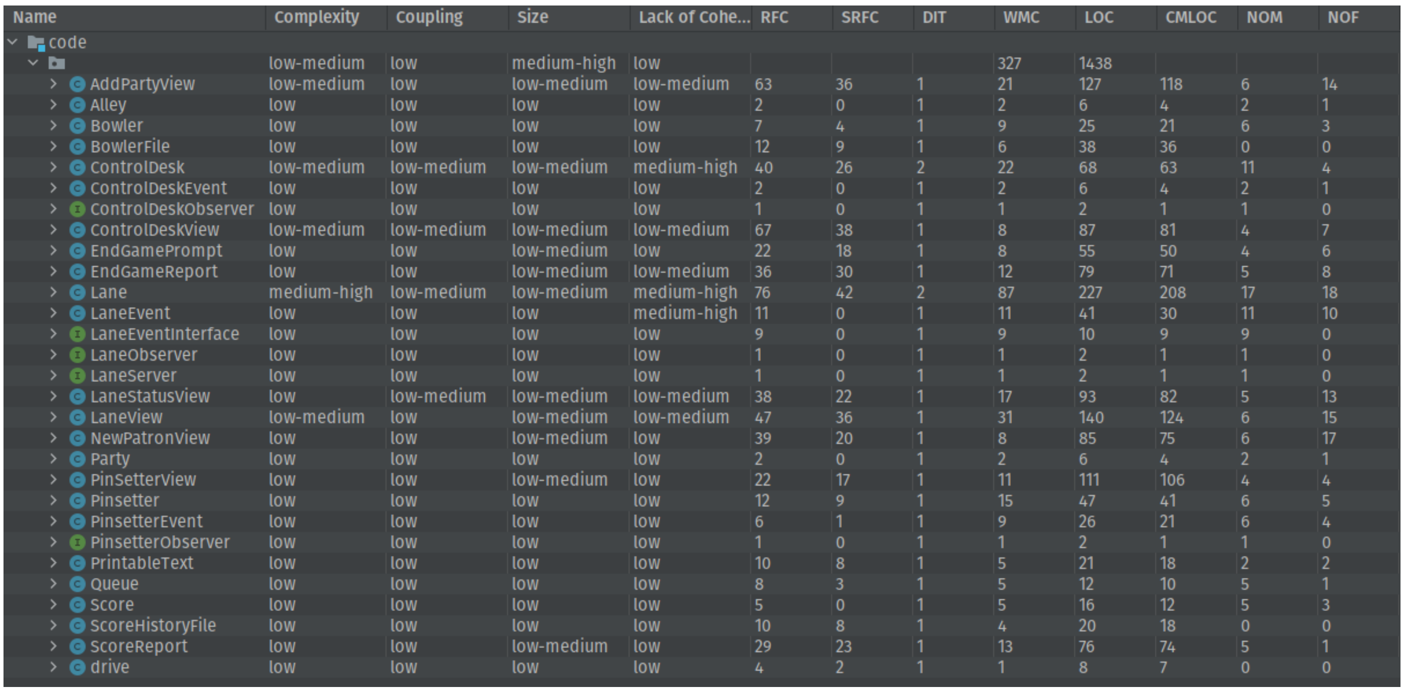
Changes made in the code

- In AddPartyView.java, redundant code is eliminated by creating two new methods named addPanel and newPanel.
- Alley.java is removed and drive class is modified and renamed as alley.java
- In Bowler.java, two methods named getNickName and getNick have the same definitions. So one method is removed and modified calls to that method accordingly.
- In ControlDesk.java observer pattern was implemented.
- In EndGamePrompt.java, LaneView.java and NewPatronView.java, redundant code is eliminated by creating a method named newPanel.
- In EndGameReport.java, redundant code is eliminated by creating two new methods named addPanel and newPanel.
- In ScoreReport.java, a new method named flush is created and removes redundant code in sendln methods.
- In Lane.java code complexity is reduced by creating two new classes ScoreCalculation.java and SubscribeLane.java
- ScoreCalculation.java handles all the calculation of scores
- PinsetterObserver.java observer pattern was implemented

Metrics

Before Refactoring

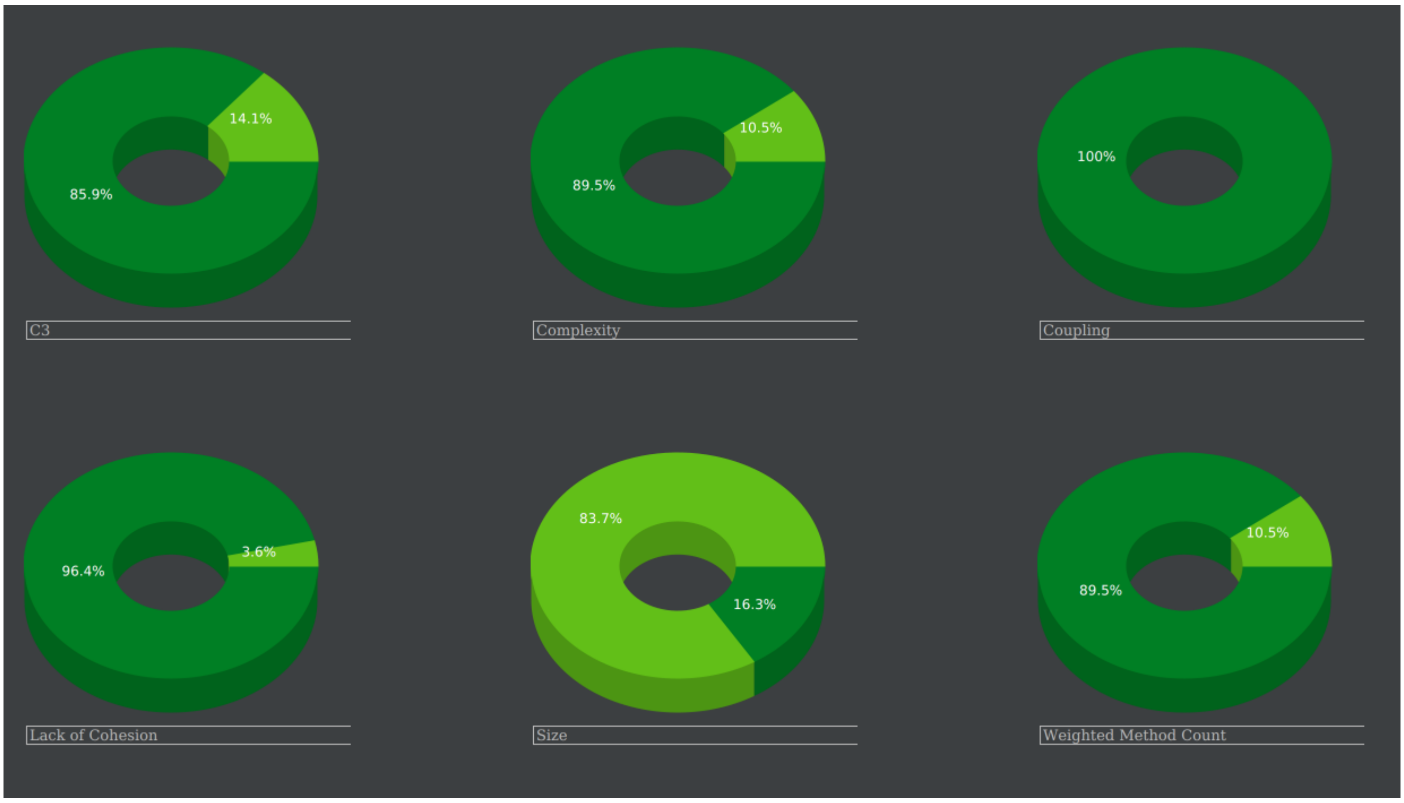


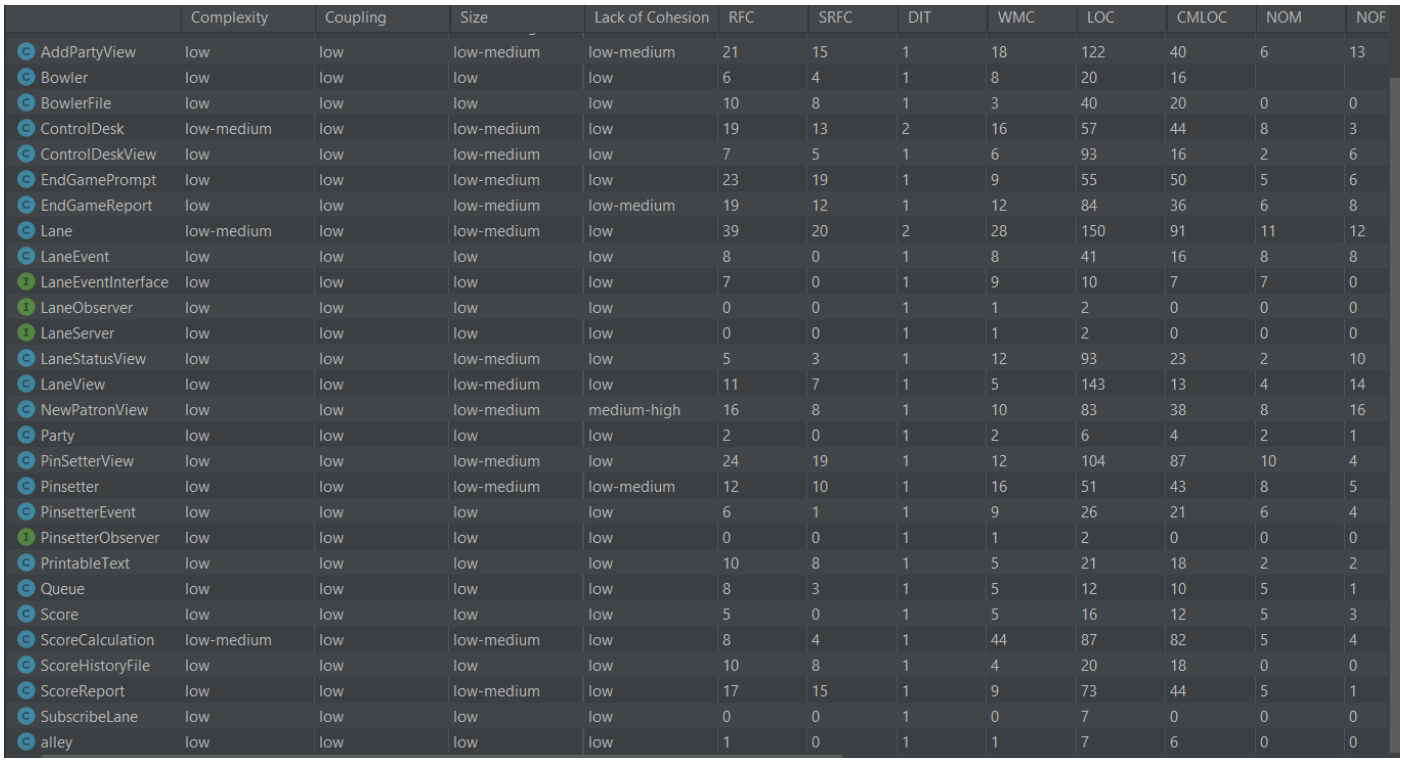


Name	Complexity	Coupling	Size	Lack of Cohe...	RFC	SRFC	DIT	WMC	LOC	CMLOC	NOM	NOF
code												
	low-medium	low	medium-high	low				327	1438			
AddPartyView	low-medium	low	low-medium	low-medium	63	36	1	21	127	118	6	14
Alley	low	low	low	low	2	0	1	2	6	4	2	1
Bowler	low	low	low	low	7	4	1	9	25	21	6	3
BowlerFile	low	low	low	low	12	9	1	6	38	36	0	0
ControlDesk	low-medium	low-medium	low-medium	medium-high	40	26	2	22	68	63	11	4
ControlDeskEvent	low	low	low	low	2	0	1	2	6	4	2	1
ControlDeskObserver	low	low	low	low	1	0	1	1	2	1	1	0
ControlDeskView	low-medium	low-medium	low-medium	low-medium	67	38	1	8	87	81	4	7
EndGamePrompt	low	low	low-medium	low	22	18	1	8	55	50	4	6
EndGameReport	low	low	low-medium	low-medium	36	30	1	12	79	71	5	8
Lane	medium-high	low-medium	low-medium	medium-high	76	42	2	87	227	208	17	18
LaneEvent	low	low	low	medium-high	11	0	1	11	41	30	11	10
LaneEventInterface	low	low	low	low	9	0	1	9	10	9	9	0
LaneObserver	low	low	low	low	1	0	1	1	2	1	1	0
LaneServer	low	low	low	low	1	0	1	1	2	1	1	0
LaneStatusView	low	low-medium	low-medium	low-medium	38	22	1	17	93	82	5	13
LaneView	low-medium	low	low-medium	low-medium	47	36	1	31	140	124	6	15
NewPatronView	low	low	low-medium	low	39	20	1	8	85	75	6	17
Party	low	low	low	low	2	0	1	2	6	4	2	1
PinSetterView	low	low	low-medium	low	22	17	1	11	111	106	4	4
Pinsetter	low	low	low	low	12	9	1	15	47	41	6	5
PinsetterEvent	low	low	low	low	6	1	1	9	26	21	6	4
PinsetterObserver	low	low	low	low	1	0	1	1	2	1	1	0
PrintableText	low	low	low	low	10	8	1	5	21	18	2	2
Queue	low	low	low	low	8	3	1	5	12	10	5	1
Score	low	low	low	low	5	0	1	5	16	12	5	3
ScoreHistoryFile	low	low	low	low	10	8	1	4	20	18	0	0
ScoreReport	low	low	low-medium	low	29	23	1	13	76	74	5	1
drive	low	low	low	low	4	2	1	1	8	7	0	0

From the above metrics, it was evident that we needed to reduce complexity, coupling, and Lack of cohesion in many of the classes. We made the changes as mentioned previously to achieve our goal.
As previously documented, we used several design patterns and removed code smells to achieve the following final results. Many metrics were clearly out of range based on the above analysis. I refactored the classes that contained a large number of lines of code.

After Refactoring





	Complexity	Coupling	Size	Lack of Cohesion	RFC	SRFC	DIT	WMC	LOC	CMLOC	NOM	NOF
AddPartyView	low	low	low-medium	low-medium	21	15	1	18	122	40	6	13
Bowler	low	low	low	low	6	4	1	8	20	16		
BowlerFile	low	low	low	low	10	8	1	3	40	20	0	0
ControlDesk	low-medium	low	low-medium	low	19	13	2	16	57	44	8	3
ControlDeskView	low	low	low-medium	low	7	5	1	6	93	16	2	6
EndGamePrompt	low	low	low-medium	low	23	19	1	9	55	50	5	6
EndGameReport	low	low	low-medium	low-medium	19	12	1	12	84	36	6	8
Lane	low-medium	low	low-medium	low	39	20	2	28	150	91	11	12
LaneEvent	low	low	low	low	8	0	1	8	41	16	8	8
LaneEventInterface	low	low	low	low	7	0	1	9	10	7	7	0
LaneObserver	low	low	low	low	0	0	1	1	2	0	0	0
LaneServer	low	low	low	low	0	0	1	1	2	0	0	0
LaneStatusView	low	low	low-medium	low	5	3	1	12	93	23	2	10
LaneView	low	low	low-medium	low	11	7	1	5	143	13	4	14
NewPatronView	low	low	low-medium	medium-high	16	8	1	10	83	38	8	16
Party	low	low	low	low	2	0	1	2	6	4	2	1
PinSetterView	low	low	low-medium	low	24	19	1	12	104	87	10	4
Pinsetter	low	low	low-medium	low-medium	12	10	1	16	51	43	8	5
PinsetterEvent	low	low	low	low	6	1	1	9	26	21	6	4
PinsetterObserver	low	low	low	low	0	0	1	1	2	0	0	0
PrintableText	low	low	low	low	10	8	1	5	21	18	2	2
Queue	low	low	low	low	8	3	1	5	12	10	5	1
Score	low	low	low	low	5	0	1	5	16	12	5	3
ScoreCalculation	low-medium	low	low-medium	low	8	4	1	44	87	82	5	4
ScoreHistoryFile	low	low	low	low	10	8	1	4	20	18	0	0
ScoreReport	low	low	low-medium	low	17	15	1	9	73	44	5	1
SubscribeLane	low	low	low	low	0	0	1	0	7	0	0	0
alley	low	low	low	low	1	0	1	1	7	6	0	0

As can be seen above, there has been a significant improvement in the metrics after refactoring was done. This was accomplished by striking a compromise between competing objectives like low coupling and high cohesion, resulting in efficient classes with no dead or duplicate code.

The metrics helped us in finding which classes need more refactoring based on their coupling/cohesion. Eg: Lane class needed the most refactoring as nothing was on the lower side in metrics: complexity, coupling, or lack of cohesion.

RFC:Response For a Class
The number of methods that can potentially be invoked in response to a public message received by an object of a specific class.

DIT: Depth of Inheritance
Except for classes that extended the Observer class, inheritance has remained unchanged.

WMC: Weighted Method Count
The weighted sum of all class methods represents a class's McCabe complexity. WMC has increased for a few classes because we removed the event classes and moved the event methods to the observable itself.

LOC: Lines of Code. Quality Attribute Related To: Size
Tried lowering LOC removing unnecessary code Modularizing code everywhere possible.

The code quality of the code base has significantly improved as a result of the refactoring. This was accomplished by striking a balance between competing criteria such as low coupling and high cohesion in order to create efficient classes with no dead or duplicate code.