Unit 1 (Refactoring) Bowling Alley Simulation- Design Document

Team 13

Title : Bowling Management Simulation.

Date of Submission : February 20th, 2022.

Github Repositery : https://github.com/debashish05/BowlingAlley

Team Memeber	Work of hour	Contribution
Debashsish Roy (2021201034)	30	1. Redesigned the Lane class. 2. Refactored the all classes related to Lane and End Game Report with the help of metric 2.0. 3. Removed code smells using Sonar Lint. 4. Analysing and identifying responsibilites of major classes. 5. Creating UML design for Lane clases. 6. Understanding the matrics and found potential blocks to improve the metrics.
Nikita Rawat (2021201035)	30	1. Redesign the controlDeskView.java, drive.java and addPartyView.java. 2. Creating UML for Control Desk Classes. 3. Bug fix. 4. Identiying weakness in the code.
Janmejay Pratap Singh Baghel (2020201089)	30	1. Understood to the working the whole code and explained it to the team member. 2. Create UML diagram for various classes. 3. Created sequence diagram for the game. 4. Identified code smells in pinsetter class.
Vinaya Sai Revanth Bachu (2020201090)	30	1. Redesigned the ScoreCalculator Class. 2. Refactored the if and else cases to decrease the nested if depth. 3. Identifying some of the code smells. 4. Refactored some of the methods to reduce cyclomatic complexities.

OVERVIEW

The bowling alley management system is a game developed in Java that simulates a bowling alley.Some of the significant features of the game area as follows:

1. **Control Desk** - The control desk is used to control the entire game .It has functionalities like add a party,view pinsetter,view scores,add new patron ,receive maintenance call,finish game,etc. After a player has finished a game the control desk is notified of the same and the controller has an option to allow another game for the player or end it.

2. **Add new player -** This interface is used to do a one time registration for the new bowler.It requires details like the full name,nickname and email id of the player.

3. **Lane** - Each lane can accommodate one to five bowlers.The bowlers who check in as a group are assigned the same lane.

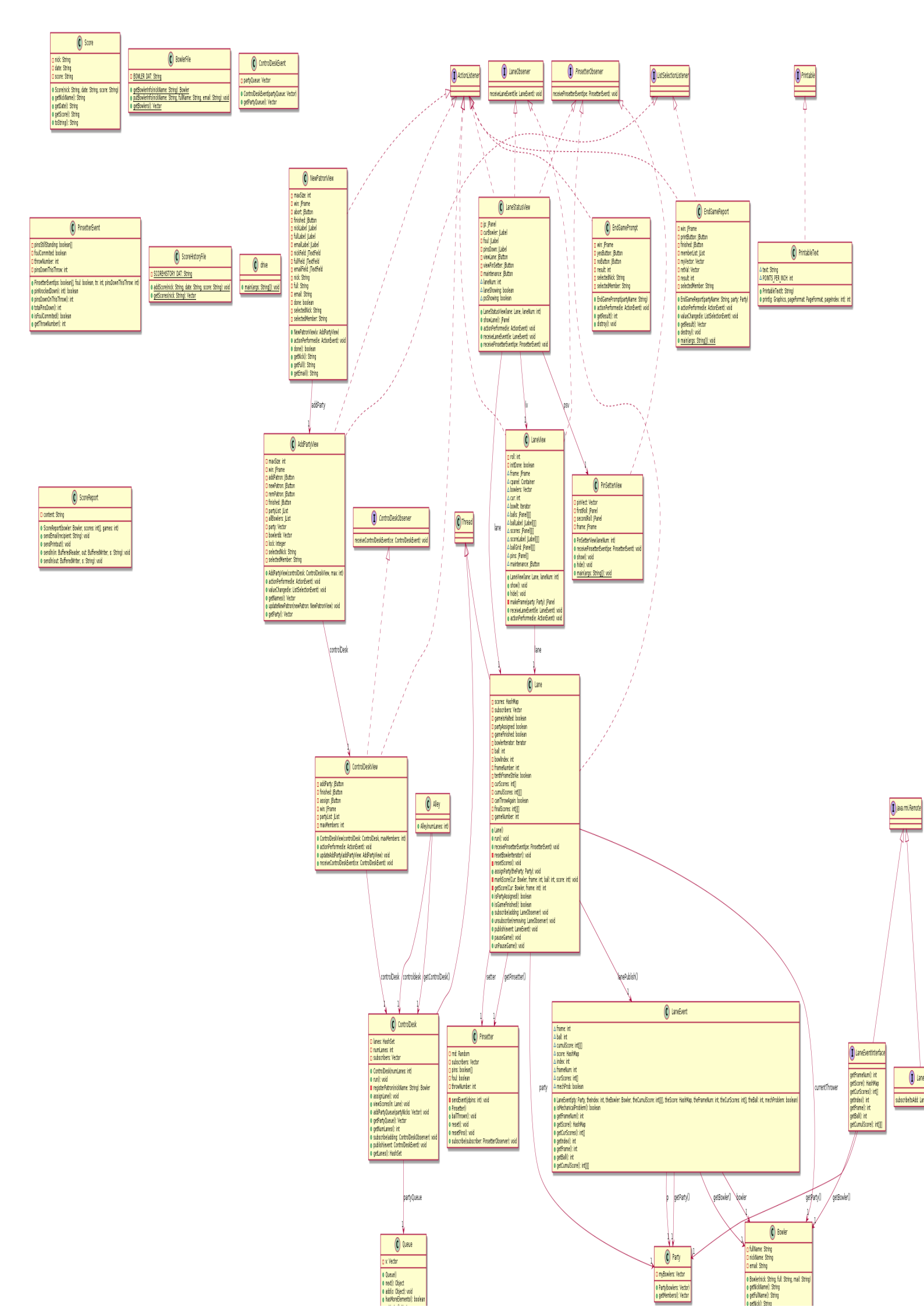
4. **Pinsetter -** The pinsetter can detect which pins were knocked down after each throw and send the score to the scoring station.

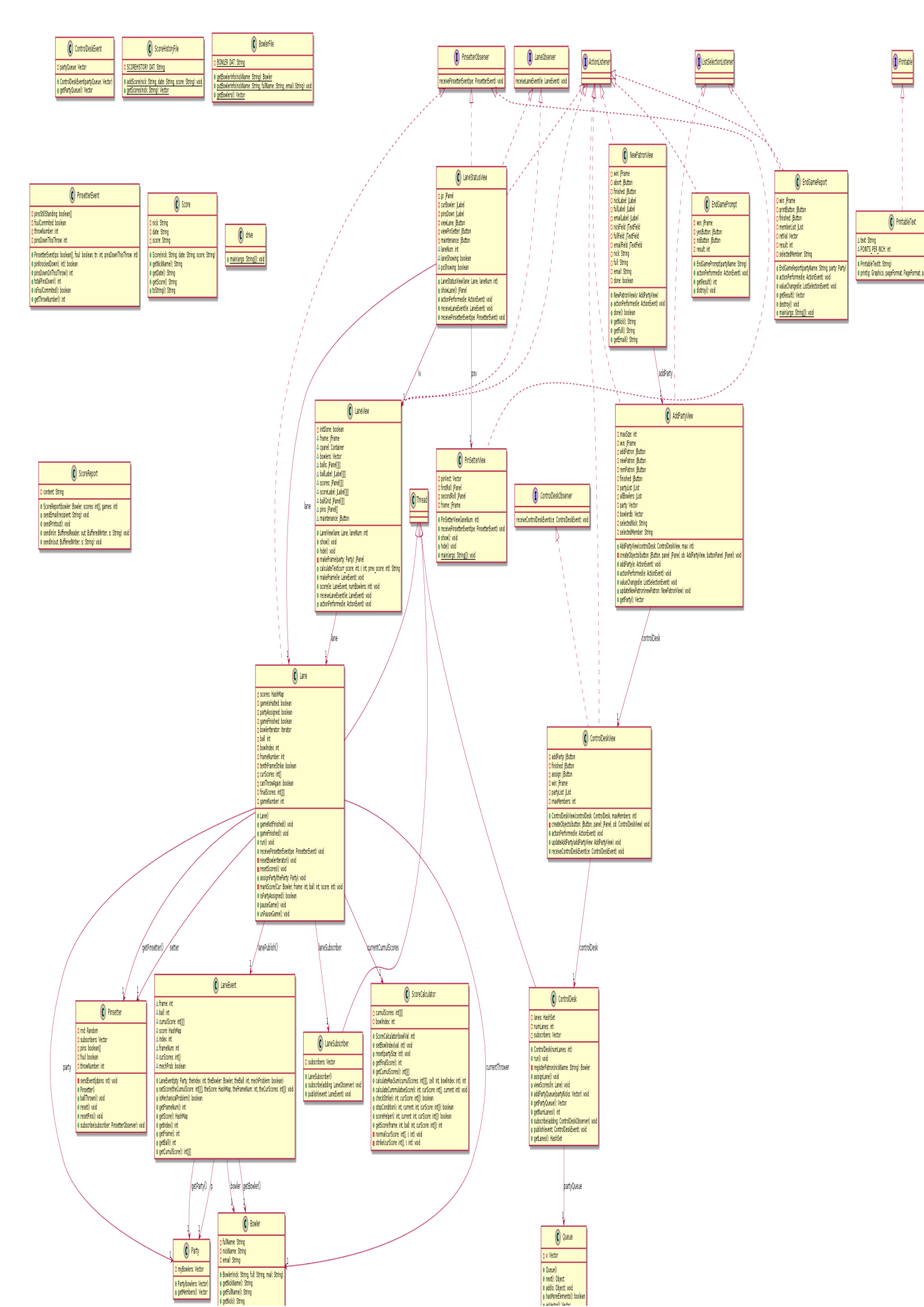
5. **Scoring Station** - The scoring station interface shows a graphical representation of the scores of the bowlers in a lane (in the order in which they checked in).

6. **Maintenance Call** - In case of a mechanical problem in any lane (eg: pinsetter did not re-rack, etc) ,the bowler can contact the control desk through an interface on the scoring station .The control station sends an acknowledgment of the request back to the station.

7. **Get bowler history -** At the end of a game ,the bowling history of the player is automatically emailed to him. He may also choose to get a printout of the same.

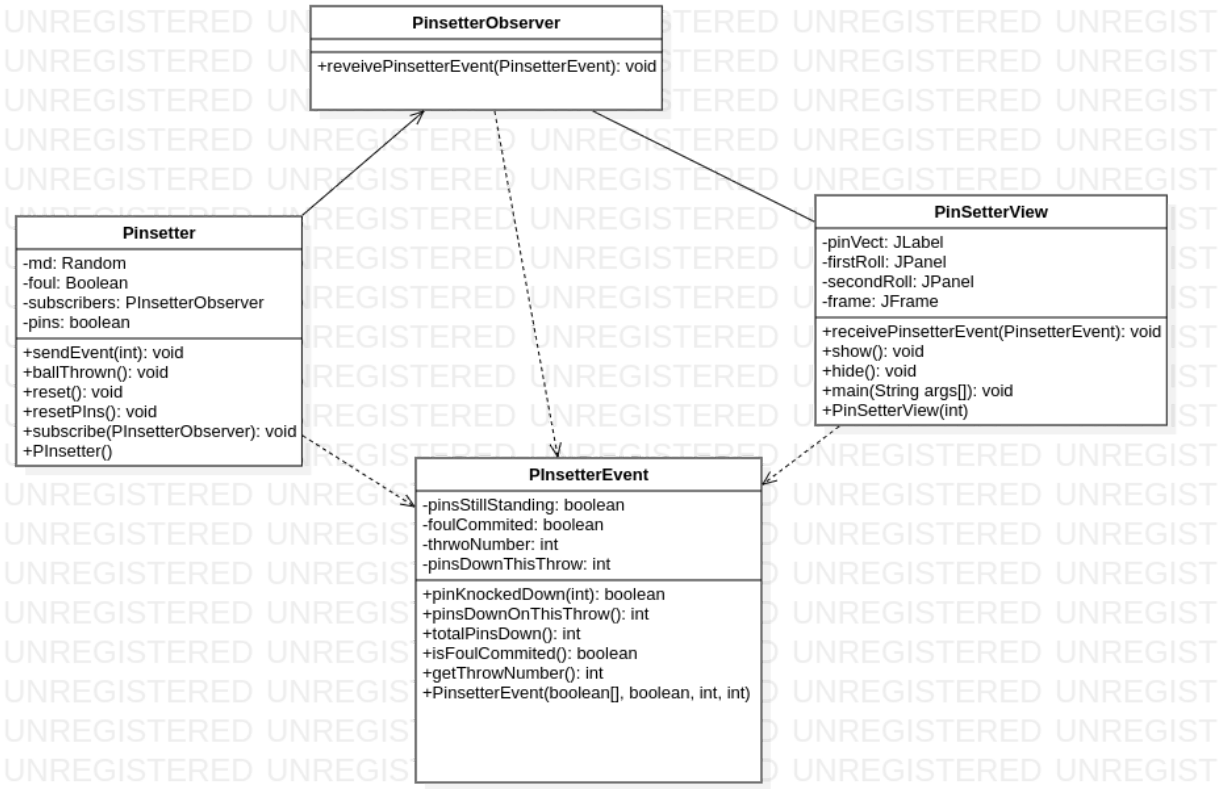
Complete UML Diagrams Before and After Refactoring


Score
□ nick: String
□ date: String
□ score: String
+ Score(nick: String, date: String, score: String)
+ getNickName(): String
+ getDate(): String
+ getScore(): String
+ toString(): String
BowlerFile
□ BOWLER_DAT: String
+ getBowlerInfo(nickName: String): Bowler
+ putBowlerInfo(nickName: String, fullName: String, email: String): void
+ getBowlers(): Vector
ControlDeskEvent
□ partyQueue: Vector
+ ControlDeskEvent(partyQueue: Vector)
+ getParty(Queue(): Vector
ActionListener
LaneObserver
receiveLaneEvent(le: LaneEvent): void
PinsetterObserver
receivePinsetterEvent(pe: PinsetterEvent): void
ListSelectionListener
Printable
NewPatronView
□ maxSize: int
□ win: JFrame
□ abort: JButton
□ finished: JButton
□ nickLabel: JLabel
□ fullLabel: JLabel
□ emailLabel: JLabel
□ nickField: JTextField
□ fullField: JTextField
□ emailField: JTextField
□ nick: String
□ full: String
□ email: String
□ done: boolean
□ selectedNick: String
□ selectedMember: String
+ NewPatronView(v: AddPartyView)
+ actionPerformed(e: ActionEvent): void
+ done(): boolean
+ getNick(): String
+ getFull(): String
+ getEmail(): String
LaneStatusView
□ jp: JPanel
□ curBowler: JLabel
□ foul: JLabel
□ pinsDown: JLabel
□ viewLane: JButton
□ viewPinSetter: JButton
□ maintenance: JButton
△ laneNum: int
△ laneShowing: boolean
△ psShowing: boolean
+ LaneStatusView(lane: Lane, laneNum: int)
+ showLane(): JPanel
+ actionPerformed(e: ActionEvent): void
+ receiveLaneEvent(le: LaneEvent): void
+ receivePinsetterEvent(pe: PinsetterEvent): void
EndGamePrompt
□ win: JFrame
□ yesButton: JButton
□ noButton: JButton
□ result: int
□ selectedNick: String
□ selectedMember: String
+ EndGamePrompt(partyName: String)
+ actionPerformed(e: ActionEvent): void
+ getResult(): int
+ destroy(): void
EndGameReport
□ win: JFrame
□ printButton: JButton
□ finished: JButton
□ memberList: JList
□ myVector: Vector
□ retVal: Vector
□ result: int
□ selectedMember: String
+ EndGameReport(partyName: String, party: Party)
+ actionPerformed(e: ActionEvent): void
+ valueChanged(e: ListSelectionEvent): void
+ getResult(): Vector
+ destroy(): void
+ main(args: String[]): void
PrintableText
△ text: String
△ POINTS_PER_INCH: int
+ PrintableText(t: String)
+ print(g: Graphics, pageFormat: PageFormat, pageIndex: int): int
PinsetterEvent
□ pinsStillStanding: boolean[]
□ foulCommitted: boolean
□ throwNumber: int
□ pinsDownThisThrow: int
+ PinsetterEvent(ps: boolean[], foul: boolean, tn: int, pinsDownThisThrow: int)
+ pinKnockedDown(i: int): boolean
+ pinsDownOnThisThrow(): int
+ totalPinsDown(): int
+ isFoulCommitted(): boolean
+ getThrowNumber(): int
ScoreHistoryFile
□ SCOREHISTORY_DAT: String
+ addScore(nick: String, date: String, score: String): void
+ getScores(nick: String): Vector
drive
+ main(args: String[]): void
addParty
v
psv
AddPartyView
□ maxSize: int
□ win: JFrame
□ addPatron: JButton
□ newPatron: JButton
□ remPatron: JButton
□ finished: JButton
□ partyList: JList
□ allBowlers: JList
□ party: Vector
□ bowlersIn: Vector
□ lock: Integer
□ selectedNick: String
□ selectedMember: String
+ AddPartyView(controlDesk: ControlDeskView, max: int)
+ actionPerformed(e: ActionEvent): void
+ valueChanged(e: ListSelectionEvent): void
+ getNames(): Vector
+ updateNewPatron(newPatron: NewPatronView): void
+ getParty(): Vector
LaneView
□ roll: int
□ initDone: boolean
△ frame: JFrame
△ cpanel: Container
△ bowlers: Vector
△ curr: int
△ bowlIt: Iterator
△ balls: JPanel[]
△ ballLabel: JLabel[][]
△ scores: JPanel[][]
△ scoreLabel: JLabel[][]
△ ballGrid: JPanel[]
△ pins: JPanel[]
△ maintenance: JButton
+ LaneView(lane: Lane, laneNum: int)
+ show(): void
+ hide(): void
# makeFrame(party: Party): JPanel
+ receiveLaneEvent(le: LaneEvent): void
+ actionPerformed(e: ActionEvent): void
PinSetterView
□ pinVect: Vector
□ firstBall: JPanel
□ secondBall: JPanel
□ frame: JFrame
+ PinSetterView(laneNum: int)
+ receivePinsetterEvent(pe: PinsetterEvent): void
+ show(): void
+ hide(): void
+ main(args: String[]): void
ScoreReport
□ content: String
+ ScoreReport(bowler: Bowler, scores: int[], games: int)
+ sendEmail(recipient: String): void
+ sendPrintout(): void
+ sendIn(in: BufferedReader, out: BufferedWriter, s: String): void
+ sendIn(out: BufferedWriter, s: String): void
ControlDeskObserver
receiveControlDeskEvent(ce: ControlDeskEvent): void
Thread
lane
lane
Lane
□ scores: HashMap
□ subscribers: Vector
□ gameIsHalted: boolean
□ partyAssigned: boolean
□ gameFinished: boolean
□ bowlerIterator: Iterator
□ ball: int
□ bowlIndex: int
□ frameNumber: int
□ tenthFrameStrike: boolean
□ curScores: int[]
□ cumuScores: int[][]
□ canThrowAgain: boolean
□ finalScores: int[]
□ gameNumber: int
+ Lane()
+ run(): void
+ receivePinsetterEvent(pe: PinsetterEvent): void
# resetBowlerIterator(): void
# resetScores(): void
+ assignPartyToParty: Party): void
# markScore(Cur: Bowler, frame: int, ball: int, score: int): void
# getScoreOf(Cur: Bowler, frame: int): int
+ isPartyAssigned(): boolean
+ isGameFinished(): boolean
+ subscribeAdding: LaneObserver): void
+ unsubscribeRemoving: LaneObserver): void
+ publish(event: LaneEvent): void
+ pauseGame(): void
+ unPauseGame(): void
ControlDeskView
□ addParty: JButton
□ finished: JButton
□ assign: JButton
□ win: JFrame
□ partyList: JList
□ maxMembers: int
+ ControlDeskView(controlDesk: ControlDesk, maxMembers: int)
+ actionPerformed(e: ActionEvent): void
+ updateAddParty(addPartyView: AddPartyView): void
+ receiveControlDeskEvent(ce: ControlDeskEvent): void
Alley
+ Alley(numLanes: int)
controlDesk
controlDesk
getControlDesk()
setter
getPinsetter()
lanePublish()
party
currentThrower
ControlDesk
□ lanes: HashSet
□ numLanes: int
□ subscribers: Vector
+ ControlDesk(numLanes: int)
+ run(): void
# registerPatron(nickName: String): Bowler
+ assignLane(): void
+ viewScores(ln: Lane): void
+ addPartyQueue(partyNicks: Vector): void
+ getPartyQueue(): Vector
+ getNumLanes(): int
+ subscribeAdding: ControlDeskObserver): void
+ publish(event: ControlDeskEvent): void
+ getLanes(): HashSet
Pinsetter
□ rnd: Random
□ subscribers: Vector
□ pins: boolean[]
□ foul: boolean
□ throwNumber: int
+ serveEvent(pins: int): void
+ Pinsetter()
+ ballThrown(): void
+ reset(): void
+ resetPins(): void
+ subscribe(subscriber: PinsetterObserver): void
LaneEvent
△ frame: int
△ ball: int
△ cumuScore: int[][]
△ score: HashMap
△ index: int
△ frameNum: int
△ curScores: int[]
△ mechProb: boolean
+ LaneEvent(pty: Party, theIndex: int, theBowler: Bowler, theCumuScore: int[][], theScore: HashMap, theFrameNum: int, theCurScores: int[], theBall: int, mechProblem: boolean)
+ isMechanicalProblem(): boolean
+ getFrameNum(): int
+ getScore(): HashMap
+ getCurScores(): int[]
+ getIndex(): int
+ getFrame(): int
+ getBall(): int
+ getCumuScore(): int[][]
java.rmi.Remote
LaneEventInterface
getFrameNum(): int
getScore(): HashMap
getCurScores(): int[]
getIndex(): int
getFrame(): int
getBall(): int
getCumuScore(): int[][]
Lane
subscribeToAdd: Lane
Queue
□ v: Vector
+ Queue()
+ next(): Object
+ add(o: Object): void
+ hasMoreElements(): boolean
partyQueue
p
getParty()
getBowler()
bowler
getParty()
getBowler()
Party
□ myBowlers: Vector
+ Party(bowlers: Vector)
+ getMembers(): Vector
Bowler
□ fullName: String
□ nickName: String
□ email: String
+ Bowler(nick: String, full: String, mail: String)
+ getNickName(): String
+ getFullName(): String
+ getNick(): String

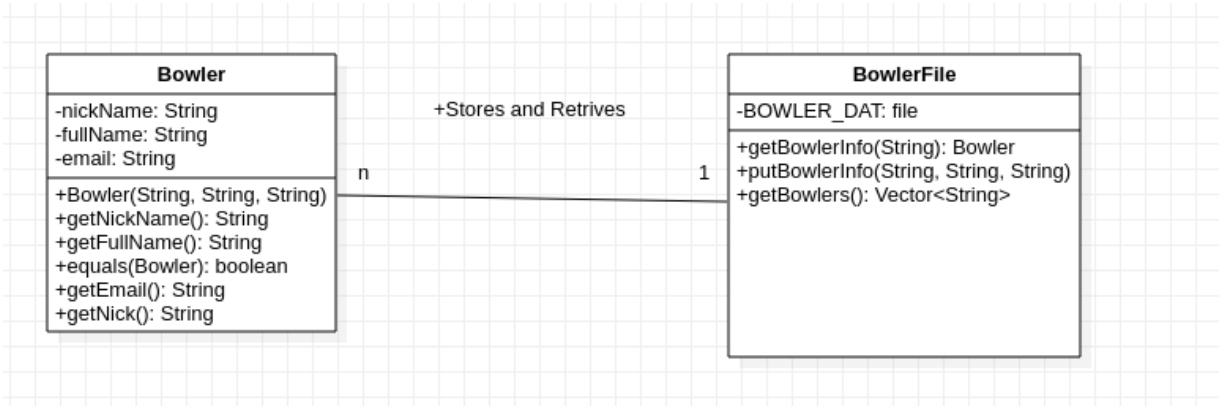

ControlDeskEvent
partyQueue: Vector
ControlDeskEvent(partyQueue: Vector)
getPartyQueue(): Vector
ScoreHistoryFile
SCOREHISTORY_DAT: String
addScore(nick: String, date: String, score: String): void
getScores(nick: String): Vector
BowlerFile
BOWLER_DAT: String
getBowlerInfo(nickName: String): Bowler
putBowlerInfo(nickName: String, fullName: String, email: String): void
getBowlers(): Vector
PinsetterObserver
receivePinsetterEvent(pe: PinsetterEvent): void
LaneObserver
receiveLaneEvent(le: LaneEvent): void
ActionListener
ListSelectionListener
Printable
PinsetterEvent
pinsStillStanding: boolean[]
frameCommited: boolean
throwNumber: int
pinsDownThisThrow: int
PinsetterEvent(ps: boolean[], foul: boolean, tr: int, pinsDownThisThrow: int)
pinKnockedDown(i: int): boolean
pinsDownOnThisThrow(): int
totalPinsDown(): int
isFrameCommited(): boolean
getThrowNumber(): int
Score
nick: String
date: String
score: String
Score(nick: String, date: String, score: String)
getNickName(): String
getDate(): String
getScore(): String
toString(): String
drive
main(args: String[]): void
LaneStatusView
pe: JPanel
curBowler: JLabel
pinsDown: JLabel
viewLane: JButton
viewPinSetter: JButton
maintenance: JButton
laneNum: int
laneShowing: boolean
psShowing: boolean
LaneStatusView(lane: Lane, laneNum: int)
showLane(): JPanel
actionPerformed(e: ActionEvent): void
receiveLaneEvent(le: LaneEvent): void
receivePinsetterEvent(pe: PinsetterEvent): void
NewPatronView
win: JFrame
abort: JButton
finished: JButton
nickLabel: JLabel
fullLabel: JLabel
emailLabel: JLabel
nickField: JTextField
fullField: JTextField
emailField: JTextField
nick: String
full: String
email: String
done: boolean
NewPatronView(v: AddPartyView)
actionPerformed(e: ActionEvent): void
done(): boolean
getNick(): String
getFull(): String
getEmail(): String
EndGamePrompt
win: JFrame
yesButton: JButton
noButton: JButton
result: int
EndGamePrompt(partyName: String)
actionPerformed(e: ActionEvent): void
getResult(): int
destroy(): void
EndGameReport
win: JFrame
printButton: JButton
finished: JButton
memberList: JList
reshid: Vector
result: int
selectedMember: String
EndGameReport(partyName: String, party: Party)
actionPerformed(e: ActionEvent): void
valueChanged(e: ListSelectionEvent): void
getResult(): Vector
destroy(): void
main(args: String[]): void
PrintableText
text: String
POINTS_PER_INCH: int
PrintableText(t: String)
print(g: Graphics, pageFormat: PageFormat, p
lv
psv
addParty
LaneView
initDone: boolean
frame: JFrame
cpanel: Container
bowlers: Vector
balls: JPanel[][]
ballLabel: JLabel[][]
scores: JPanel[][]
scoreLabel: JLabel[][]
ballGrid: JPanel[][]
pins: JPanel[]
maintenance: JButton
LaneView(lane: Lane, laneNum: int)
show(): void
hide(): void
makeFrame(party: Party): JPanel
calculateText(curr: score: int, i: int, prev_score: int): String
makeFrameIe: LaneEvent): void
scoreIe: LaneEvent, numBowlers: int): void
receiveLaneEvent(le: LaneEvent): void
actionPerformed(e: ActionEvent): void
AddPartyView
maxSize: int
win: JFrame
addPatron: JButton
newPatron: JButton
remPatron: JButton
finished: JButton
partyList: JList
allBowlers: JList
party: Vector
bowlerIb: Vector
selectedNick: String
selectedMember: String
AddPartyView(controlDesk: ControlDeskView, max: int)
createObjects(button: JButton, panel: JPanel, ob: AddPartyView, buttonPanel: JPanel): void
addParty(e: ActionEvent): void
actionPerformed(e: ActionEvent): void
valueChanged(e: ListSelectionEvent): void
updateNewPatron(newPatron: NewPatronView): void
getParty(): Vector
controlDesk
ScoreReport
content: String
ScoreReport(bowler: Bowler, scores: int[], games: int)
sendEmail(recipient: String): void
sendPrintout(): void
sendIn(in: BufferedReader, out: BufferedWriter, s: String): void
sendInOut(in: BufferedReader, out: BufferedWriter, s: String): void
PinSetterView
pinVect: Vector
firstBall: JPanel
secondBall: JPanel
frame: JFrame
PinSetterView(laneNum: int)
receivePinsetterEvent(pe: PinsetterEvent): void
show(): void
hide(): void
main(args: String[]): void
Thread
ControlDeskObserver
receiveControlDeskEvent(ce: ControlDeskEvent): void
lane
Lane
scores: HashMap
gameIsHalted: boolean
partyAssigned: boolean
gameFinished: boolean
bowlerIterator: Iterator
ball: int
bowlIndex: int
frameNumber: int
tenthFrameStrike: boolean
curScores: int[]
canThrowAgain: boolean
finalScores: int[]
gameNumber: int
Lane()
gameWillFinish(): void
gameFinished(): void
run(): void
receivePinsetterEvent(pe: PinsetterEvent): void
resetBowlerIterator(): void
resetScores(): void
assignPartyIfNeeded(p: Party): void
markScore(Cur Bowler, frame: int, ball: int, score: int): void
isPartyAssigned(): boolean
pauseGame(): void
unPauseGame(): void
ControlDeskView
addParty: JButton
finished: JButton
assign: JButton
win: JFrame
partyList: JList
maxMembers: int
ControlDeskView(controlDesk: ControlDesk, maxMembers: int)
createObjects(button: JButton, panel: JPanel, ob: ControlDeskView): void
actionPerformed(e: ActionEvent): void
updateAddParty(addPartyView: AddPartyView): void
receiveControlDeskEvent(ce: ControlDeskEvent): void
getPinsetter()
setter
lanePublish()
laneSubscriber
currentCumulScores
party
currentThrower
Pinsetter
rnd: Random
subscribers: Vector
pins: boolean[]
foul: boolean
throwNumber: int
serEvent(pins: int): void
Pinsetter()
ballThrown(): void
reset(): void
resetPins(): void
subscribe(subscriber: PinsetterObserver): void
LaneEvent
frame: int
ball: int
cumulScore: int[][]
score: HashMap
index: int
frameNum: int
curScores: int[]
markPrinb: boolean
LaneEvent(party: Party, theIndex: int, theBowler: Bowler, theBall: int, mechProblem: boolean)
setScores(theCumulScore: int[][], theScore: HashMap, theFrameNum: int, theCurScores: int[]): void
isMechanicalProblem(): boolean
getFrameNum(): int
getScore(): HashMap
getIndex(): int
getFrame(): int
getBall(): int
getCumulScore(): int[][]
LaneSubscriber
subscribers: Vector
LaneSubscriber()
subscribeAdding: LaneObserver): void
publishEvent: LaneEvent): void
ScoreCalculator
cumulScores: int[][]
bowlIndex: int
ScoreCalculator(bowlVal: int)
setBowlIndex(val: int): void
resetPartySize: int): void
getFinalScores(): int
getCumulScores(): int[][]
calculateNewSum(cumulScores: int[][], cell: int, bowlIndex: int): int
calculateCummulativeScore(i: int, curScore: int[], current: int): void
checkStrike(i: int, curScore: int[]): boolean
stopCondition(i: int, current: int, curScore: int[]): boolean
scoreHelper(i: int, current: int, curScore: int[]): boolean
getScoreFrame: int, ball: int, curScore: int[]): int
normalcurScore: int[], i: int): void
strikecurScore: int[] i: int): void
ControlDesk
lanes: HashSet
numLanes: int
subscribers: Vector
ControlDesk(numLanes: int)
run(): void
registerPatron(nickName: String): Bowler
assignLane(i: void
viewScores(ln: Lane): void
addPartyQueue(partyNick: Vector): void
getPartyQueue(): Vector
getNumLanes(): int
subscribeAdding: ControlDeskObserver): void
publishEvent: ControlDeskEvent): void
getLanes(): HashSet
getPinsetter()
getParty()
p
bowler
getBowler()
partyQueue
Party
myBowlers: Vector
Party(bowlers: Vector)
getMembers(): Vector
Bowler
fullName: String
nickName: String
email: String
Bowler(nick: String, full: String, mail: String)
getNickName(): String
getFullName(): String
getNick(): String
Queue
v: Vector
Queue()
next(): Object
add(x: Object): void
hasMoreElements(): boolean

UML Diagrams After Refactoring

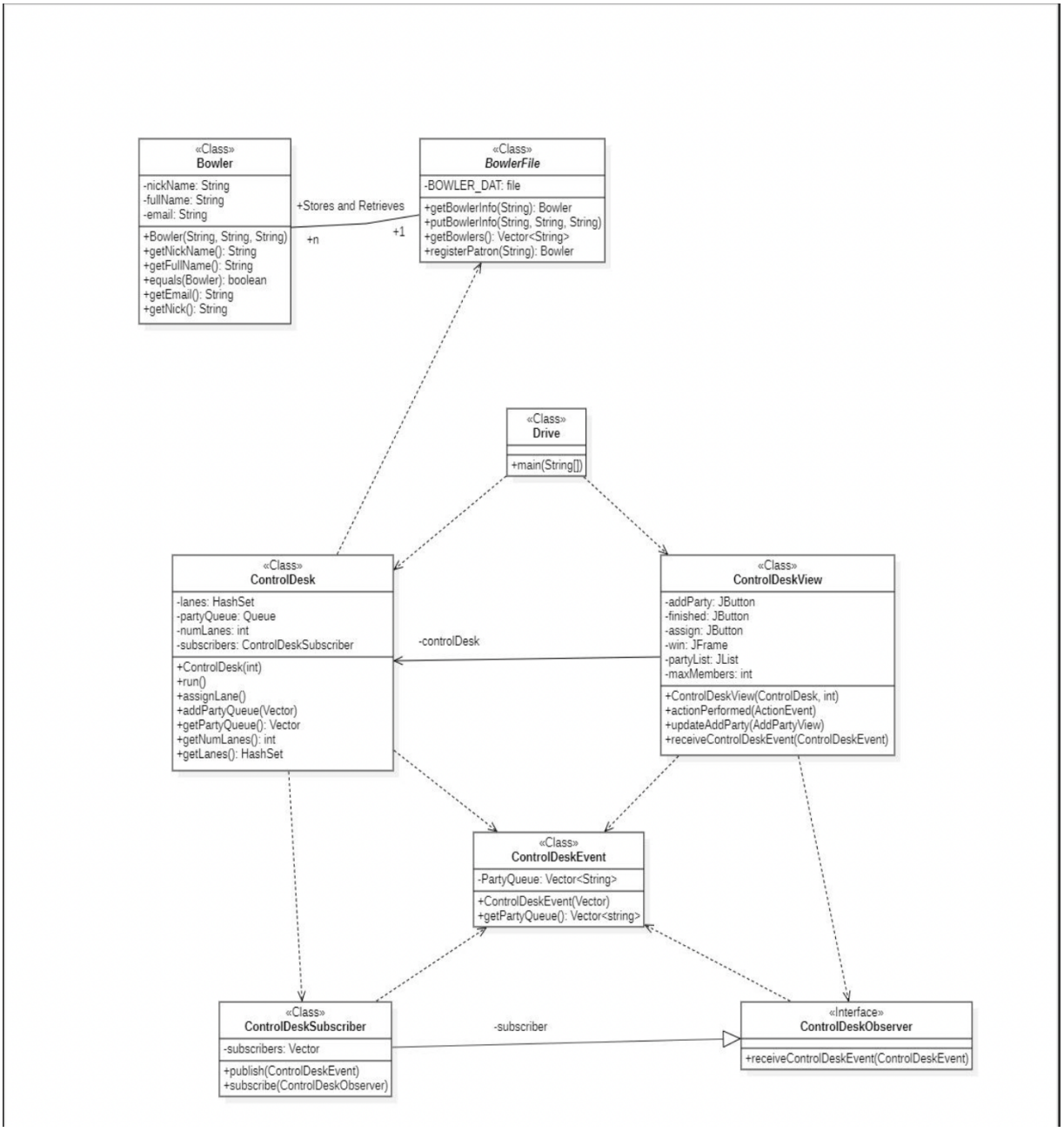
PinSetter Classes



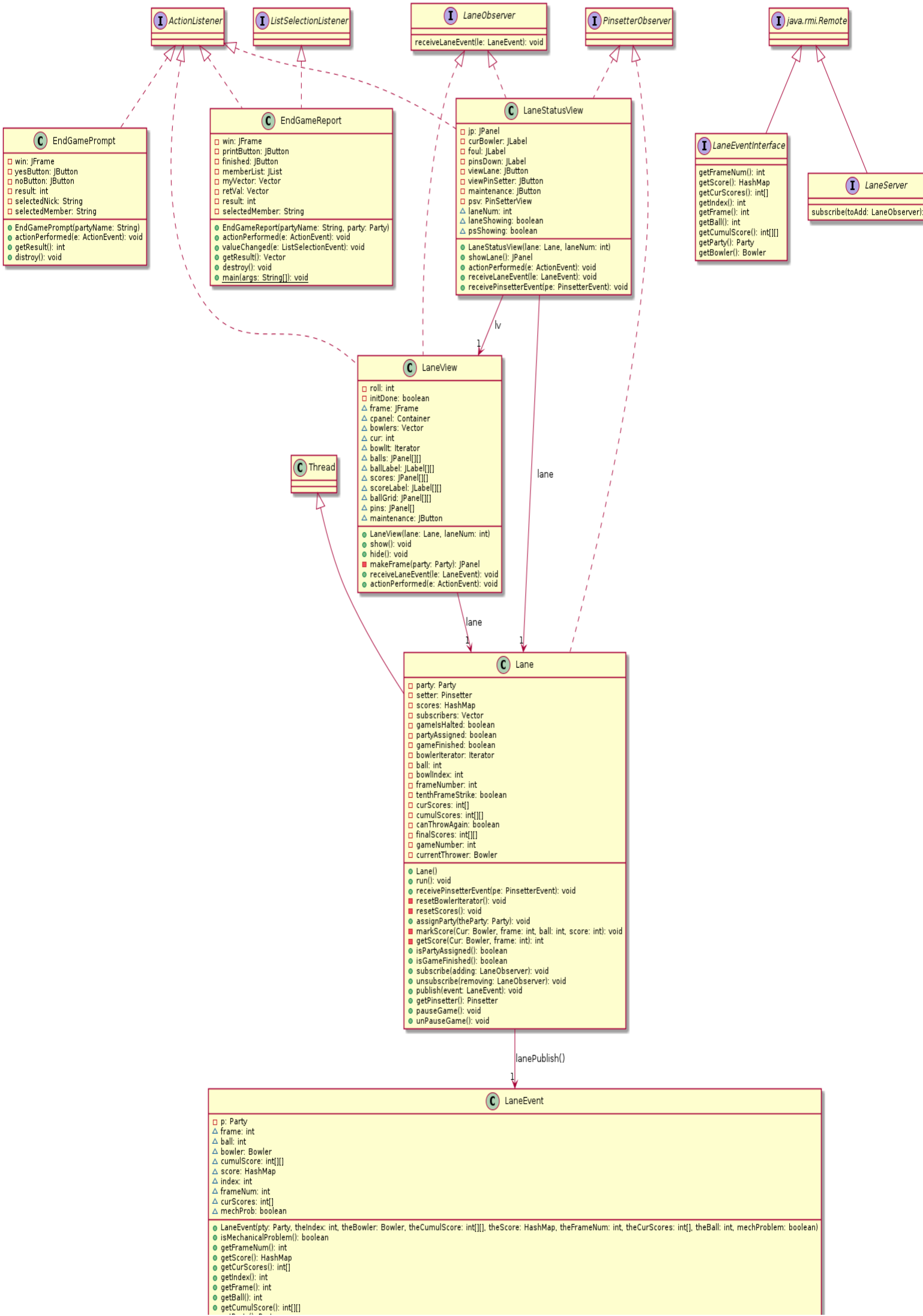
Bowler Classes

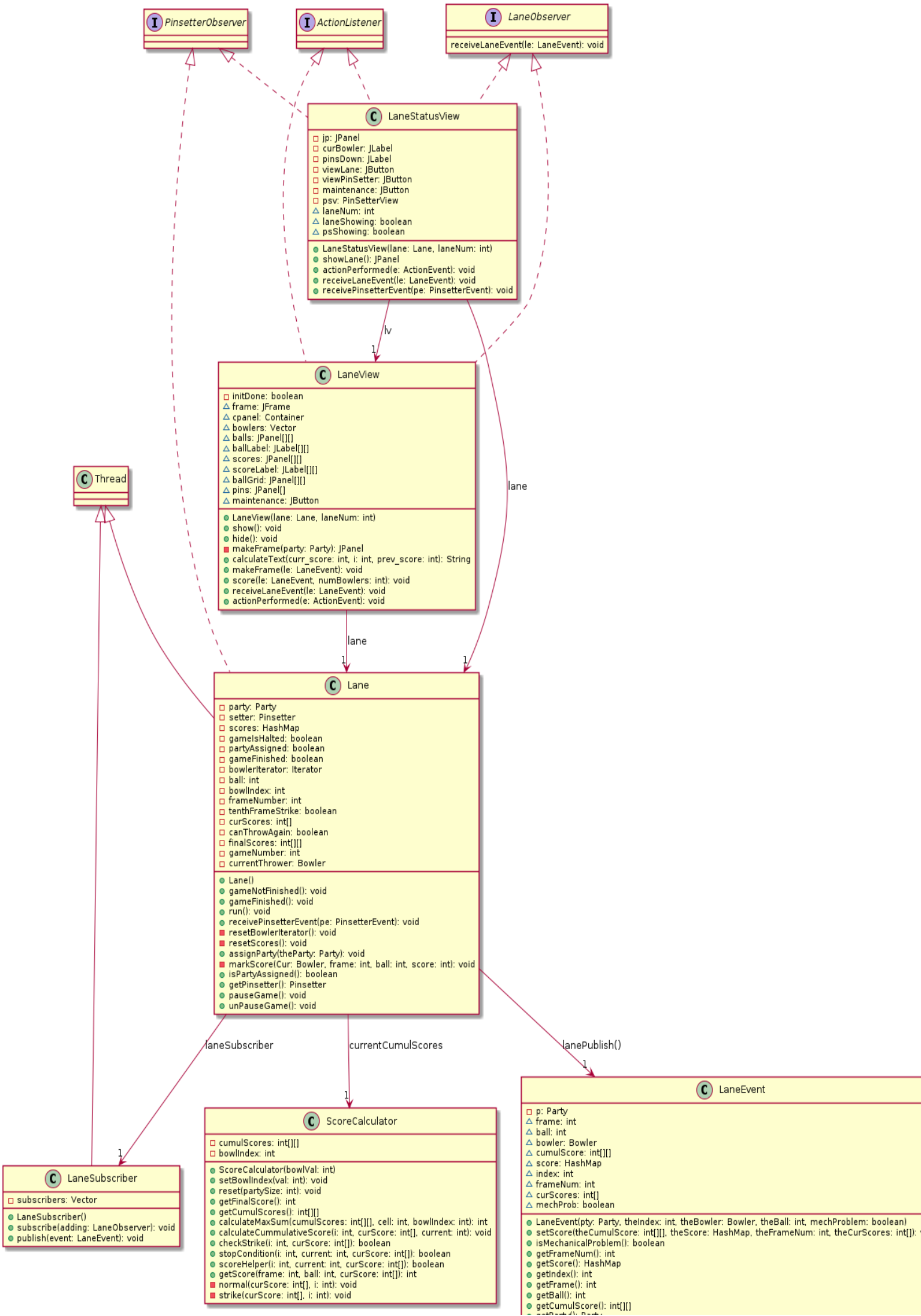


CONTROL DESK AND DRIVE CLASSES

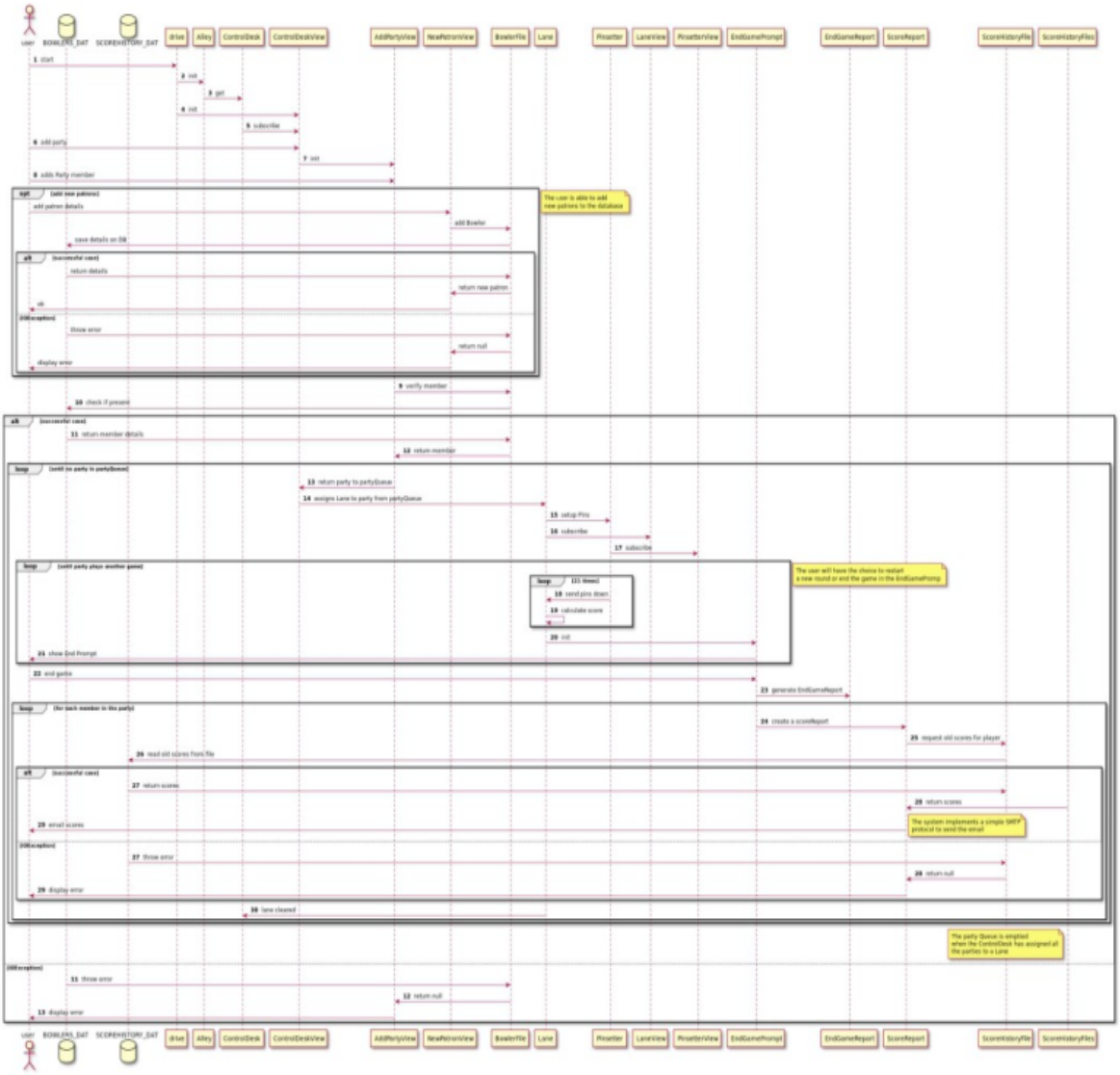


Before and After Refactoring of Lane classes and End Game Report


ActionListener
ListSelectionListener
LaneObserver
receiveLaneEvent(le: LaneEvent): void
PinsetterObserver
java.rmi.Remote
EndGamePrompt
win: JFrame
yesButton: JButton
noButton: JButton
result: int
selectedNick: String
selectedMember: String
EndGamePrompt(partyName: String)
actionPerformed(e: ActionEvent): void
getResult(): int
distroy(): void
EndGameReport
win: JFrame
printButton: JButton
finished: JButton
memberList: JList
myVector: Vector
retVal: Vector
result: int
selectedMember: String
EndGameReport(partyName: String, party: Party)
actionPerformed(e: ActionEvent): void
valueChanged(e: ListSelectionEvent): void
getResult(): Vector
destroy(): void
main(args: String[]): void
LaneStatusView
jp: JPanel
curBowler: JLabel
foul: JLabel
pinsDown: JLabel
viewLane: JButton
viewPinSetter: JButton
maintenance: JButton
psv: PinSetterView
laneNum: int
laneShowing: boolean
psShowing: boolean
LaneStatusView(lane: Lane, laneNum: int)
showLane(): JPanel
actionPerformed(e: ActionEvent): void
receiveLaneEvent(le: LaneEvent): void
receivePinsetterEvent(pe: PinsetterEvent): void
LaneEventInterface
getFrameNum(): int
getScore(): HashMap
getCurScores(): int[]
getIndex(): int
getFrame(): int
getBall(): int
getCumulScore(): int[][]
getParty(): Party
getBowler(): Bowler
LaneServer
subscribe(toAdd: LaneObserver):
lv
1
LaneView
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
LaneView(lane: Lane, laneNum: int)
show(): void
hide(): void
makeFrame(party: Party): JPanel
receiveLaneEvent(le: LaneEvent): void
actionPerformed(e: ActionEvent): void
Thread
lane
lane
1
1
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
receivePinsetterEvent(pe: PinsetterEvent): void
resetBowlerIterator(): void
resetScores(): void
assignParty(theParty: Party): void
markScore(Cur: Bowler, frame: int, ball: int, score: int): void
getScore(Cur: Bowler, frame: int): int
isPartyAssigned(): boolean
isGameFinished(): boolean
subscribe(adding: LaneObserver): void
unsubscribe(removing: LaneObserver): void
publish(event: LaneEvent): void
getPinsetter(): Pinsetter
pauseGame(): void
unPauseGame(): void
lanePublish()
1
LaneEvent
p: Party
frame: int
ball: int
bowler: Bowler
cumulScore: int[][]
score: HashMap
index: int
frameNum: int
curScores: int[]
mechProb: boolean
LaneEvent(pty: Party, theIndex: int, theBowler: Bowler, theCumulScore: int[][], theScore: HashMap, theFrameNum: int, theCurScores: int[], theBall: int, mechProblem: boolean)
isMechanicalProblem(): boolean
getFrameNum(): int
getScore(): HashMap
getCurScores(): int[]
getIndex(): int
getFrame(): int
getBall(): int
getCumulScore(): int[][]


PinsetterObserver
ActionListener
LaneObserver
receiveLaneEvent(le: LaneEvent): void
LaneStatusView
jp: JPanel
curBowler: JLabel
pinsDown: JLabel
viewLane: JButton
viewPinSetter: JButton
maintenance: JButton
psv: PinSetterView
laneNum: int
laneShowing: boolean
psShowing: boolean
LaneStatusView(lane: Lane, laneNum: int)
showLane(): JPanel
actionPerformed(e: ActionEvent): void
receiveLaneEvent(le: LaneEvent): void
receivePinsetterEvent(pe: PinsetterEvent): void
lv
1
LaneView
initDone: boolean
frame: JFrame
cpanel: Container
bowlers: Vector
balls: JPanel[][]
ballLabel: JLabel[][]
scores: JPanel[][]
scoreLabel: JLabel[][]
ballGrid: JPanel[][]
pins: JPanel[]
maintenance: JButton
LaneView(lane: Lane, laneNum: int)
show(): void
hide(): void
makeFrame(party: Party): JPanel
calculateText(curr_score: int, i: int, prev_score: int): String
makeFrame(le: LaneEvent): void
score(le: LaneEvent, numBowlers: int): void
receiveLaneEvent(le: LaneEvent): void
actionPerformed(e: ActionEvent): void
lane
1
1
Thread
Lane
party: Party
setter: Pinsetter
scores: HashMap
gameIsHalted: boolean
partyAssigned: boolean
gameFinished: boolean
bowlerIterator: Iterator
ball: int
bowlIndex: int
frameNumber: int
tenthFrameStrike: boolean
curScores: int[]
canThrowAgain: boolean
finalScores: int[][]
gameNumber: int
currentThrower: Bowler
Lane()
gameNotFinished(): void
gameFinished(): void
run(): void
receivePinsetterEvent(pe: PinsetterEvent): void
resetBowlerIterator(): void
resetScores(): void
assignParty(theParty: Party): void
markScore(Cur: Bowler, frame: int, ball: int, score: int): void
isPartyAssigned(): boolean
getPinsetter(): Pinsetter
pauseGame(): void
unPauseGame(): void
laneSubscriber
currentCumulScores
lanePublish()
1
1
1
LaneSubscriber
subscribers: Vector
LaneSubscriber()
subscribe(adding: LaneObserver): void
publish(event: LaneEvent): void
ScoreCalculator
cumulScores: int[][]
bowlIndex: int
ScoreCalculator(bowlVal: int)
setBowlIndex(val: int): void
reset(partySize: int): void
getFinalScore(): int
getCumulScores(): int[][]
calculateMaxSum(cumulScores: int[][], cell: int, bowlIndex: int): int
calculateCummulativeScore(i: int, curScore: int[], current: int): void
checkStrike(i: int, curScore: int[]): boolean
stopCondition(i: int, current: int, curScore: int[]): boolean
scoreHelper(i: int, current: int, curScore: int[]): boolean
getScore(frame: int, ball: int, curScore: int[]): int
normal(curScore: int[], i: int): void
strike(curScore: int[], i: int): void
LaneEvent
p: Party
frame: int
ball: int
bowler: Bowler
cumulScore: int[][]
score: HashMap
index: int
frameNum: int
curScores: int[]
mechProb: boolean
LaneEvent(pty: Party, theIndex: int, theBowler: Bowler, theBall: int, mechProblem: boolean)
setScore(theCumulScore: int[][], theScore: HashMap, theFrameNum: int, theCurScores: int[]):
isMechanicalProblem(): boolean
getFrameNum(): int
getScore(): HashMap
getIndex(): int
getFrame(): int
getBall(): int
getCumulScore(): int[][]

UML Sequence Diagram



Classes Overview

S.No.	Class Name	Responsibility
1.	Driver	It is the main class of the project. It initializes the game, sets the parameters like number of lanes, maximum patrons per party. It also initializes all the necessary objects.
2.	Control Desk	It handles control desk window. It assigns the lane, creates parties and maintain the wait queue. It is responsible for broadcasting the event to subscribing objects. Creating a new patron.
3.	Lane	Lanes are handled by the lane class. This is reponsible to start a new thread for each lane and assigning party to that lane. It also updates the score and act likes a observer to the subscriber. Later we have separated it into two more classes.
4.	Pin Setter	It performs pin setter simulation and notifies the events.
5.	Bowler and Bowler File	Information about the bowler will be stored here. It used to add new bowler, getting details of all bowlers, validation of the bowler.
6.	Endgame prompt	Prompts after the end of the game to prompt if the player wants to play another game.
7.	EndgameRep.	It allows user to make the game report at the end.
8.	Score Calculator	It helps the lane class to caculate score using strike and normal functions. This is the result of refactoring the prior code.
9.	Score report	Creates the final score of each person in tha party and sends their respective reports by email.
10.	AddPartyView	It will add a new patron to the party. Removing a patron from the party. Creating a new Patron. Finished party selection and returning the latest state of the party.
11.	ControlDeskEvent	It return the vector consisting names of the party in wait queue.
12.	ControlDeskView	It handles GUI for control desk. It facilitates handler for action event. Receive a new party from addPartyView.
13.	Drive	It is driver class for the game. Creates alles with number of lanes. It also helps in activating the control desk object.
14.	Laneevent	It consists of setter and getter function for all lane functions.
15.	PinSetterView	It displays the pin setter GUI. Receive the current state of the pins and change the GUI looks.

ANALYZING THE ORIGINAL CODE DESIGN

We shall start with the Pros and then will discuss the Cons of the given project.

Pros :-

- **Proper Comments** :- The code was well commented and the purpose and functionality of the most part of the system was provided.
- **Low Coupling** :- The overall system as well as the subsystems were having low coupling metric.

Cons :-

- **Dead Code :-** The original code contains a lot of code which is commented out or has empty blocks. Some variables and methods which are declared, have also not been used anywhere in the system.
- **Large Classes and Big Functions** :- Some classes were burdened with way too many methods and some functions had a lot of code which could have been modularized.
- **Big Nested If Depth** :- Some of the functions which could have been simply written had way too many if's and else.
- **Duplicate Code** :- Similar kind of job was being done in some of the methods where the code was copy pasted.
- **Use of Deprecated Methods** :- Many system functions used in the project were deprecated which could have been replaced by new functions.
- **Too Many Parameters** :- Some functions and constructors had way too many parameters.
- "Print Report" functionality is missing.

CODE SMELLS AND CHANGES

S.No	Class Name	Code Smell Instances and changes performed
1	**LaneView**	Removed variables which are unused. The variables are :- ● private int roll ● int cur

		• Iterator bowlIt • Insets buttonMargin = new Insets(4,4,4,4) High Cyclomatic Complexity :- • public void receiveLaneEvent(LaneEvent le) High Nested If Depth :- • Public void receiveLaneEvent(LaneEvent le) Improvements we have made :- • Added new functions calculateText,makeFrame to reduce cyclomatic complexity and have reduced Nested If Depth.
2	**LaneStatusView**	Removed variables which are unused. The variables are :- • Insets buttonMargin = new Insets(4,4,4,4) • private JLabel foul • Removed dead code
3	**Lane**	Removed functions which are not used. The functions are :- • public boolean isGameFinished() Also removed empty blocks of code present in this class. Removed variables :- • EndGamePrompt egp • vector subscriber Moved some functions out of Lane to create LaneSubscirbe class and ScoreCalculate class. Added GameFinished and GameNotFinished functions to reduce the cyclomatic complexity of run function, and also have simplified some of the if conditions.Also changed receivePinSetterEvent function to reduce complexity. Changed the if condition to correctly check if 2 strings are not same or not.
4	**LaneEventInterface**	Removed these functions as there are not used anywhere. • public int getFrameNum( ) throws java.rmi.RemoteException • public int[] getCurScores( ) throws java.rmi.RemoteException • public int[][] getCumulScore() throws java.rmi.RemoteException

5	**EndGamePrompt**	Removed Unused Variables :- ● selectedNick ● selectedMember ● ButtonMargin Also removed many unused import statements.
6	**EndGameReport**	Removed Unused Variables :- ● myVector ● EndGame e ● ButtonMargin ● String partyName Also refactored code in this class.
7	**LaneEvent**	Removed getCurScores() function as it is not used. Also reduced the number of parameters passed to the constructor and added a new set method.
8	**LaneObserver**	Removed extra semicolons
9	**ScoreCalculator (Derived from Lane Class)**	normal function :- ● Reduced the nested if depth. getScore :- ● Removed unused argument Bowler. ● Reduced nested if depth ● Created a new function calculateCumulativeScore to reduce the cyclomatic complexity.
10	**AddPartyView**	Removed unused variables :- ● lock ● Insets buttonMargin ● NewPatronView Removed unused getNames() function and unused import statements. In actionPerformed function reduced nested if depth. Removed repeating code by moving it into a new function called createObjects
11	**LaneSubscriber**	Modified the if conditions. Removed unsubscribe() function which is not used.
12	**LaneStatusView**	Reduced the actionperformed function cyclomatic complexity.
13	**NewPatronView**	Removed variables :- ● int maxSize

		• Insets buttonMargin • selectedNick • selectedMember Removed unused import statements.
14	**Alley.java**	Removed the class by creating the ControlDesk object directly in the drive.java class.
15	**ControlDeskView.java**	Removed repeating code by moving it into a new function called createObjects
16	**PinSetterView**	Contained Dead code and Deprecated Functions
17	**PinSetterObserver**	Modifier public is redundant for interface
18	**Printable Text**	Unused Import Statement
19	**ScoreHistoryFile**	More General exception is already used in throws list
20	**NewPatronView**	Unused variables and import statements

Analyzing the refactored code

- **Low Coupling** :- The dependencies between the classes were moderate and we have tried to make it low by passing the parameters locally and removing the redundant ones wherever possible. We have extended our class list to break down large files such as Lane into different subclasses. We have made sure that these classes were mostly independent and did not require too many other dependencies to increase the coupling.

- **High Cohesion** :- Cohesion tells about the consistency and organization of different units. The more tasks a single class tries to perform, we have a problem with cohesion there . The long classes had numerous methods which often became unrelated and too broad. We split such classes eg. Lane class has been divided into Lane, LaneSubscribe and ScoreCalculator class.

- **Separation of Concerns** :- Separation of Concerns is a design principle for separating a system into distinct sections such that each section addresses a separate concern. An example of how we achieved this in the refactored design is by creating a separate score calculating class. Previously Lane Class had a method getScore() which calculates the score but we have created a separate class ScoreCalculator for calculating the score and the updated score is sent to Lane Class to mark.

- **Dead Code Elimination** :- An optimization that removes code which does not affect the program results. We also eliminated extra methods and variables which are mentioned above.

- **Extensibility** :- As we ensured low coupling, we made sure that it was easier to introduce new modules. As we wrote basic code for the UI, we could easily extend it to add specific features in the respective classes.

- **Reusability** :- To ensure the code reusability several methods were written. Wherever in the original code we found that a similar kind of task was done by copy-pasting we modularized the code.

Metric Analysis

Following questions has been answered in this section.
- What were the metrics for the code base? What did these initial measurements tell you about the system.
- How did you use these measurements to guide your refactoring?
- How did your refactoring affect the metrics? Did your refactoring improve the metrics? In all areas? In some areas? What contributed to these results?

1. McCabe Cyclomatic Complexity

Measurement tell us about the system
- McCabe's cyclomatic complexity is a software quality metric that quantifies the complexity of a software program.
- It indicate how difficult it will be for the user to traverse through the whole code and understand.
- We have used Metric 2.0 and by default the max limit of the McCabe Cyclomatic Complexity is set to 10.

Inference: getScore, run and recieveLaneEvent have very high complexity. If we are able to simply these functions, this parameter will definitely fall down.

Functions that violated this constraints are as follows:
Lane.java(Lane):
- getScore - 38
- run - 19
- receivePinsetterEvent - 12

LaneView.java(LaneView)

- receiveLaneEvent - 19

LaneStatusView.java
- actionPerformed - 11

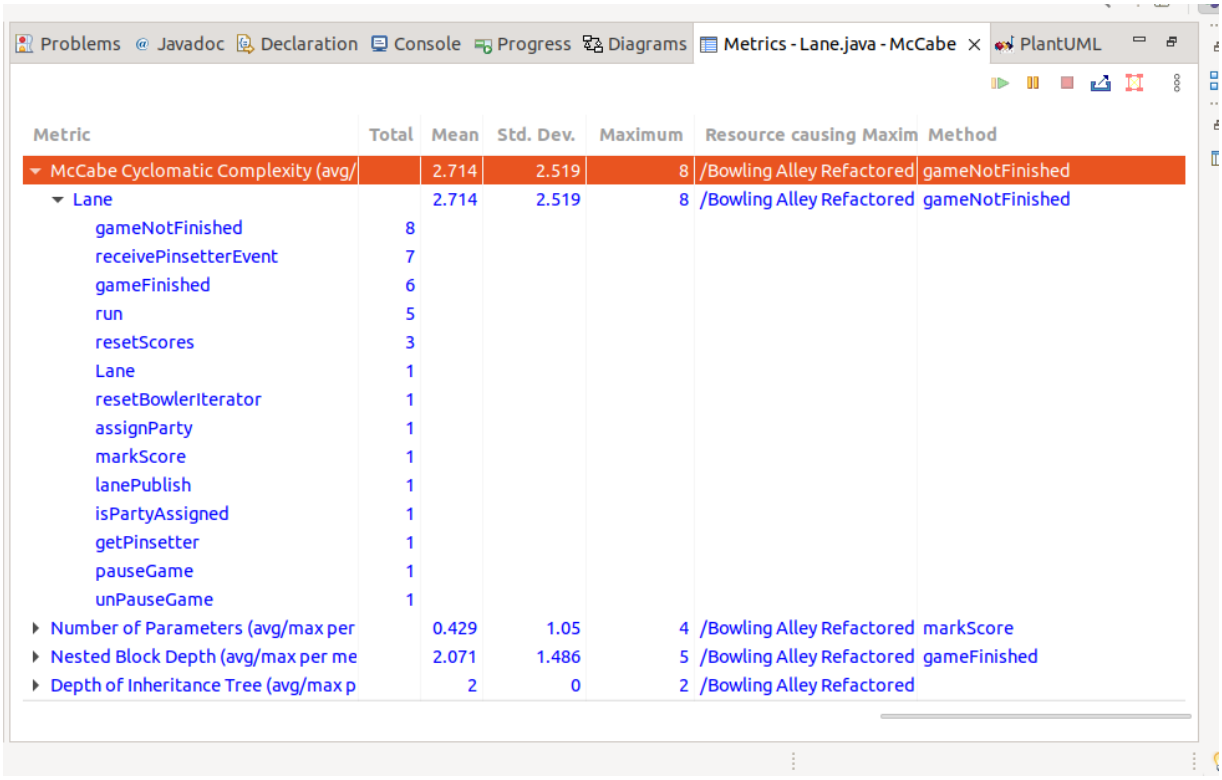
AddPartyView.java
- actionPerformed - 11

Guide for refactoring

We targetted these functions and try to simply it, so that it is much readable. So we try to split up the methods and made different functions for a specific task. Multiple nested loops blocks has been converted to functions. We try to stick to make the function smallers and decompose large functions into multiple smaller ones.

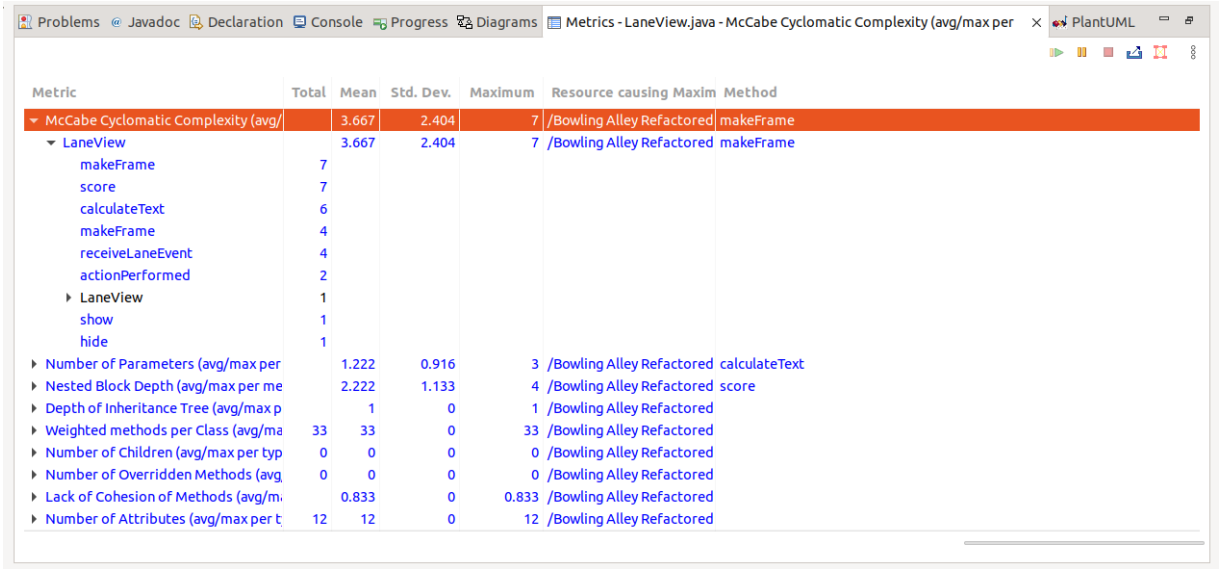
After Refactoring

- **Lane.java**



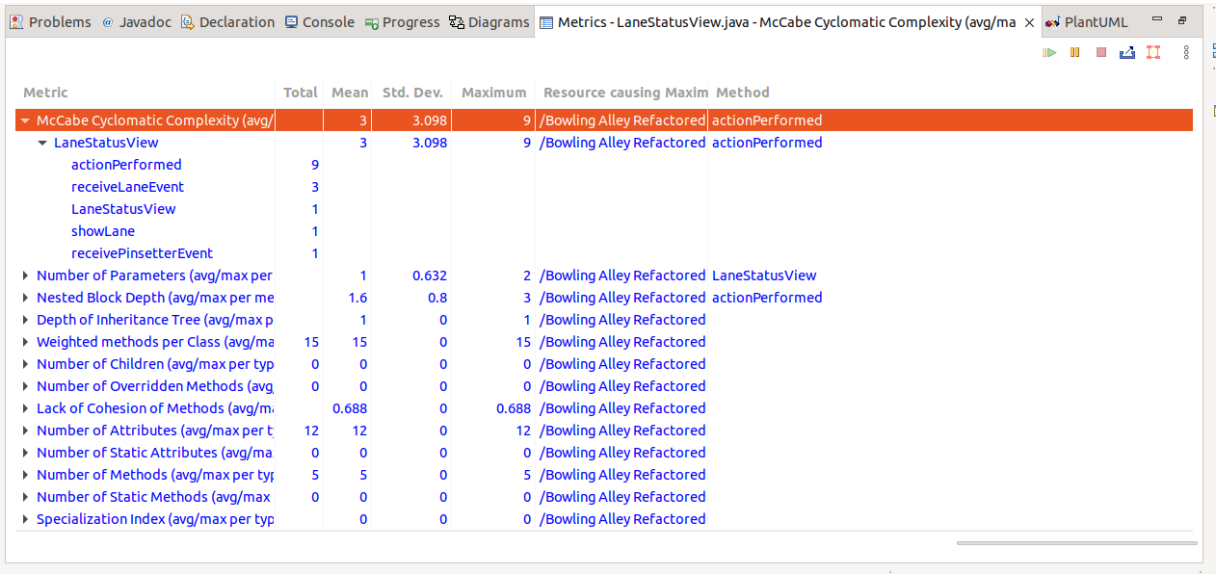
Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maxim	Method
McCabe Cyclomatic Complexity (avg/		2.714	2.519	8	/Bowling Alley Refactored	gameNotFinished
Lane		2.714	2.519	8	/Bowling Alley Refactored	gameNotFinished
gameNotFinished	8					
receivePinsetterEvent	7					
gameFinished	6					
run	5					
resetScores	3					
Lane	1					
resetBowlerIterator	1					
assignParty	1					
markScore	1					
lanePublish	1					
isPartyAssigned	1					
getPinsetter	1					
pauseGame	1					
unPauseGame	1					
Number of Parameters (avg/max per		0.429	1.05	4	/Bowling Alley Refactored	markScore
Nested Block Depth (avg/max per me		2.071	1.486	5	/Bowling Alley Refactored	gameFinished
Depth of Inheritance Tree (avg/max p		2	0	2	/Bowling Alley Refactored	

- **LaneView.java**



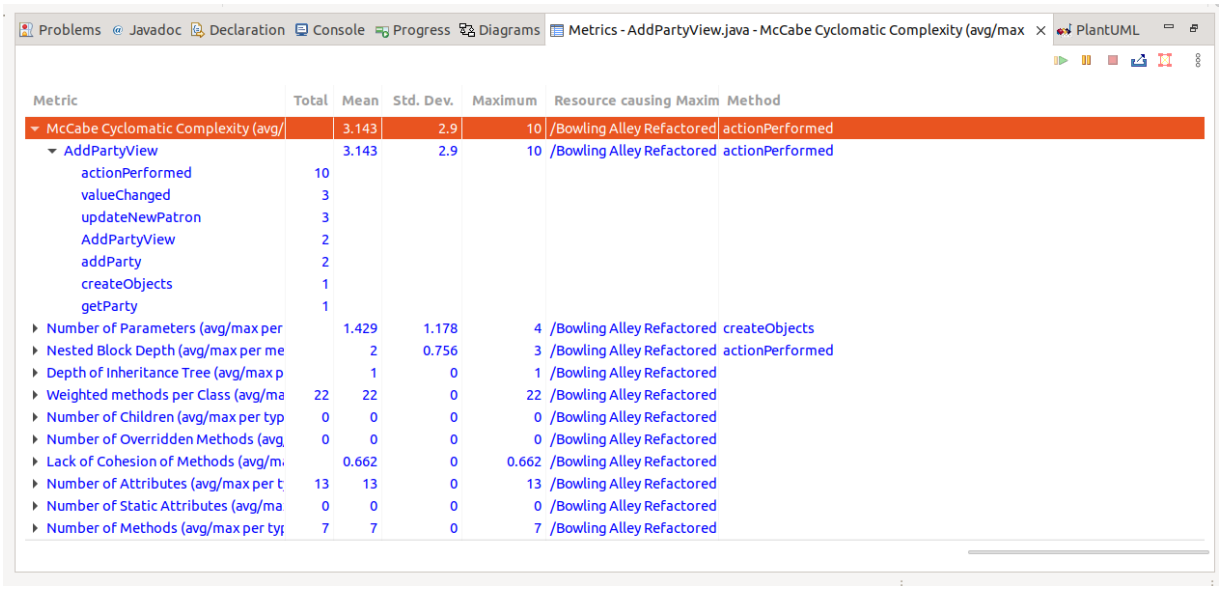
Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maxim	Method
McCabe Cyclomatic Complexity (avg/		3.667	2.404	7	/Bowling Alley Refactored	makeFrame
LaneView		3.667	2.404	7	/Bowling Alley Refactored	makeFrame
makeFrame	7					
score	7					
calculateText	6					
makeFrame	4					
receiveLaneEvent	4					
actionPerformed	2					
LaneView	1					
show	1					
hide	1					
Number of Parameters (avg/max per		1.222	0.916	3	/Bowling Alley Refactored	calculateText
Nested Block Depth (avg/max per me		2.222	1.133	4	/Bowling Alley Refactored	score
Depth of Inheritance Tree (avg/max p		1	0	1	/Bowling Alley Refactored	
Weighted methods per Class (avg/ma	33	33	0	33	/Bowling Alley Refactored	
Number of Children (avg/max per typ	0	0	0	0	/Bowling Alley Refactored	
Number of Overridden Methods (avg	0	0	0	0	/Bowling Alley Refactored	
Lack of Cohesion of Methods (avg/m		0.833	0	0.833	/Bowling Alley Refactored	
Number of Attributes (avg/max per t	12	12	0	12	/Bowling Alley Refactored	

- **LaneStatusView.java**



Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maxim	Method
McCabe Cyclomatic Complexity (avg/		3	3.098	9	/Bowling Alley Refactored	actionPerformed
LaneStatusView		3	3.098	9	/Bowling Alley Refactored	actionPerformed
actionPerformed	9					
receiveLaneEvent	3					
LaneStatusView	1					
showLane	1					
receivePinsetterEvent	1					
Number of Parameters (avg/max per		1	0.632	2	/Bowling Alley Refactored	LaneStatusView
Nested Block Depth (avg/max per me		1.6	0.8	3	/Bowling Alley Refactored	actionPerformed
Depth of Inheritance Tree (avg/max p		1	0	1	/Bowling Alley Refactored	
Weighted methods per Class (avg/ma	15	15	0	15	/Bowling Alley Refactored	
Number of Children (avg/max per typ	0	0	0	0	/Bowling Alley Refactored	
Number of Overridden Methods (avg	0	0	0	0	/Bowling Alley Refactored	
Lack of Cohesion of Methods (avg/m		0.688	0	0.688	/Bowling Alley Refactored	
Number of Attributes (avg/max per t	12	12	0	12	/Bowling Alley Refactored	
Number of Static Attributes (avg/ma	0	0	0	0	/Bowling Alley Refactored	
Number of Methods (avg/max per typ	5	5	0	5	/Bowling Alley Refactored	
Number of Static Methods (avg/max	0	0	0	0	/Bowling Alley Refactored	
Specialization Index (avg/max per typ		0	0	0	/Bowling Alley Refactored	

- **AddPartyView.java**



Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maxim	Method
McCabe Cyclomatic Complexity (avg/		3.143	2.9	10	/Bowling Alley Refactored	actionPerformed
AddPartyView		3.143	2.9	10	/Bowling Alley Refactored	actionPerformed
actionPerformed	10					
valueChanged	3					
updateNewPatron	3					
AddPartyView	2					
addParty	2					
createObjects	1					
getParty	1					
Number of Parameters (avg/max per		1.429	1.178	4	/Bowling Alley Refactored	createObjects
Nested Block Depth (avg/max per me		2	0.756	3	/Bowling Alley Refactored	actionPerformed
Depth of Inheritance Tree (avg/max p		1	0	1	/Bowling Alley Refactored	
Weighted methods per Class (avg/ma	22	22	0	22	/Bowling Alley Refactored	
Number of Children (avg/max per typ	0	0	0	0	/Bowling Alley Refactored	
Number of Overridden Methods (avg	0	0	0	0	/Bowling Alley Refactored	
Lack of Cohesion of Methods (avg/m		0.662	0	0.662	/Bowling Alley Refactored	
Number of Attributes (avg/max per t	13	13	0	13	/Bowling Alley Refactored	
Number of Static Attributes (avg/ma	0	0	0	0	/Bowling Alley Refactored	
Number of Methods (avg/max per typ	7	7	0	7	/Bowling Alley Refactored	

2. Number of parameters

Measurements tell us about the system

- It is very difficult to understand the code if the we pass to many parameter to the functions.
- By default in Metric 2.0 it is set to maximum of 5.

Inference

Function LaneEvent have higher number of parameter. If we can reduce it to 5 this issue will be resolved.

Functions that violated this constraints are as follows:

LaneEvent.java(LaneView)

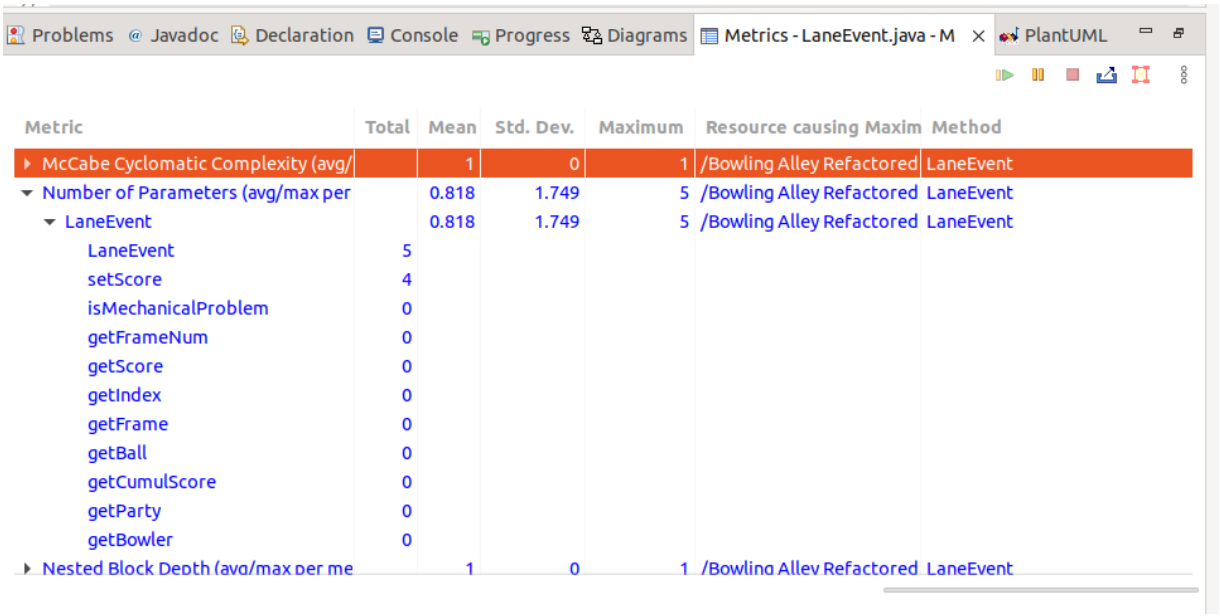
- LaneEvent - 9

Guide for refactoring

- Instead of passing all the parameter at once we have added a extra function which facilitates the exchange of parameter.
- Objects can also be passed but we used the earlier one.
- Splitting up the method also reduces the parameter, this can also help.

After Refactoring

- **LaneEvent.java**



Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maxim	Method
McCabe Cyclomatic Complexity (avg/		1	0	1	/Bowling Alley Refactored	LaneEvent
Number of Parameters (avg/max per		0.818	1.749	5	/Bowling Alley Refactored	LaneEvent
LaneEvent		0.818	1.749	5	/Bowling Alley Refactored	LaneEvent
LaneEvent	5					
setScore	4					
isMechanicalProblem	0					
getFrameNum	0					
getScore	0					
getIndex	0					
getFrame	0					
getBall	0					
getCumulScore	0					
getParty	0					
getBowler	0					
Nested Block Depth (avg/max per me		1	0	1	/Bowling Alley Refactored	LaneEvent

3. Nested Block Depth

Measurements tell us about the system

- It should be kept lower because as the nested block depth increases it will be difficult for us to track the flow as a human point of view.
- Metrics set maximum block depth to 5.

Inference

- Lane class have function run() that violates the constraints and can be rectified by rephrasing the if conditions and thus reducing the number of blocks.

Functions that violated this constraints are as follows:

Lane.java(Lane)

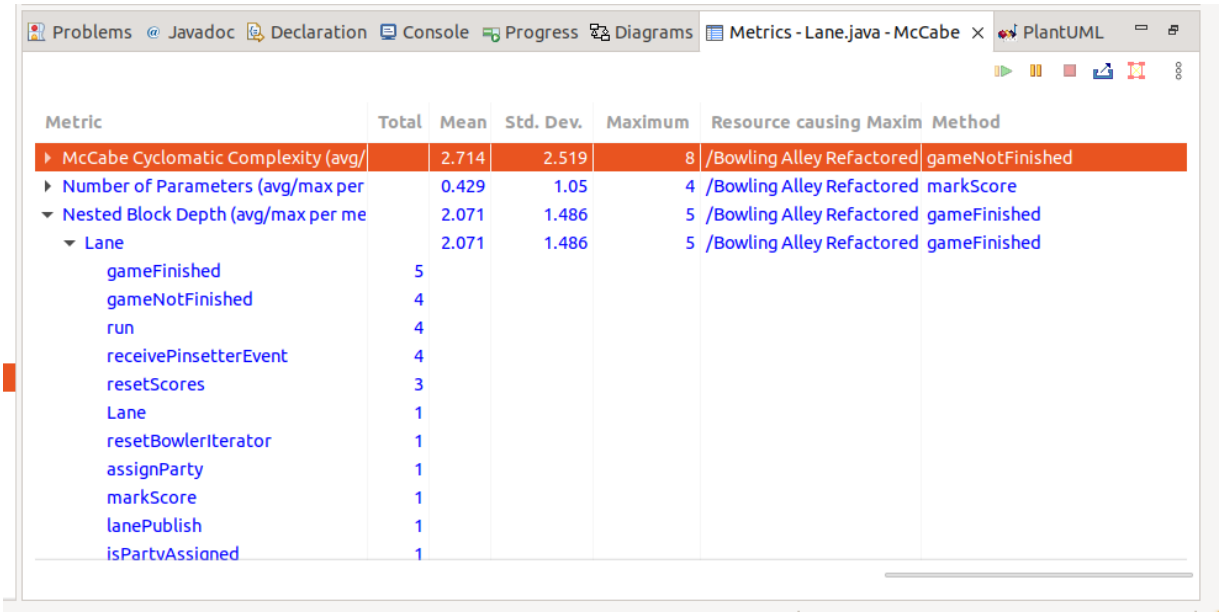
- run - 7

Guide for refactoring

- We have added the functions to avoid the nested block depth.
- We replaced redundant blocks with functions.

After Refactoring

- **Lane.java**



4. Lack of Cohesion of Methods

Measurements tell us about the system

- Cohesion is a measure of the degree to which the elements of the module are functionally related. A good software design will have high cohesion.
- Getters and setter used can increase cohesion.

Inference

- Getters didn't need splitting
- LaneView class does needed splitting. But sorted out while solving for first 3 parameters.

Functions that violated this constraints are as follows:

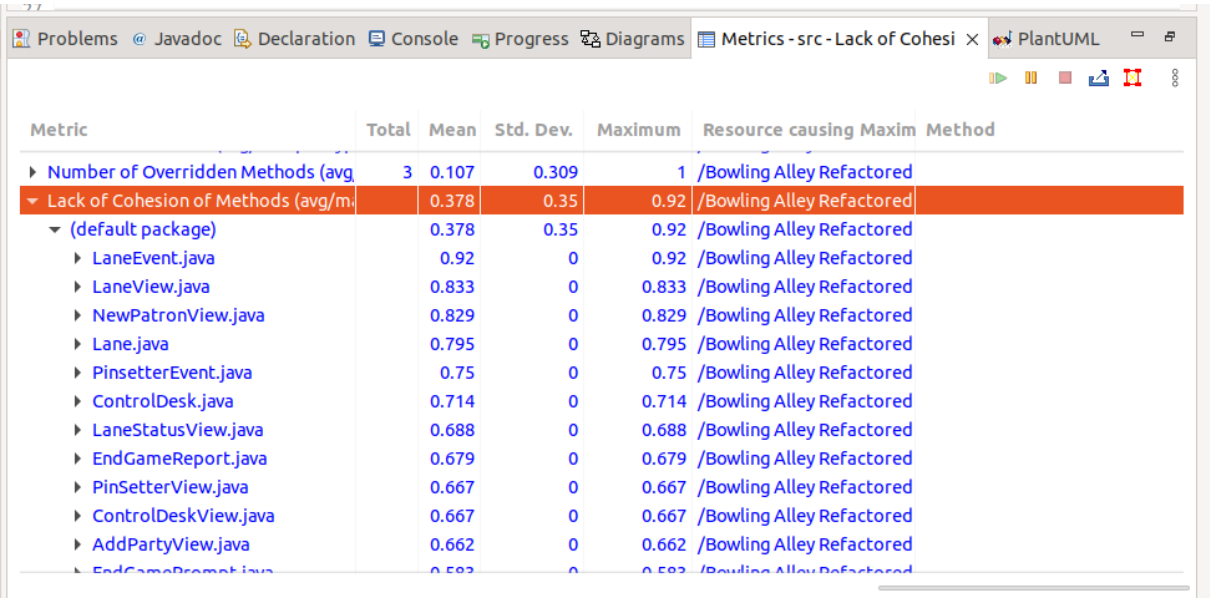
- No function violated any constraints here.

Guide for refactoring

- Split up into simple commonents by adding functions or making new classes altogether such that all the related functions are in a same module.

After Refactoring

- Very minor change in the metric.



Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maxim	Method
Number of Overridden Methods (avg	3	0.107	0.309	1	/Bowling Alley Refactored	
Lack of Cohesion of Methods (avg/m		0.378	0.35	0.92	/Bowling Alley Refactored	
(default package)		0.378	0.35	0.92	/Bowling Alley Refactored	
LaneEvent.java		0.92	0	0.92	/Bowling Alley Refactored	
LaneView.java		0.833	0	0.833	/Bowling Alley Refactored	
NewPatronView.java		0.829	0	0.829	/Bowling Alley Refactored	
Lane.java		0.795	0	0.795	/Bowling Alley Refactored	
PinsetterEvent.java		0.75	0	0.75	/Bowling Alley Refactored	
ControlDesk.java		0.714	0	0.714	/Bowling Alley Refactored	
LaneStatusView.java		0.688	0	0.688	/Bowling Alley Refactored	
EndGameReport.java		0.679	0	0.679	/Bowling Alley Refactored	
PinSetterView.java		0.667	0	0.667	/Bowling Alley Refactored	
ControlDeskView.java		0.667	0	0.667	/Bowling Alley Refactored	
AddPartyView.java		0.662	0	0.662	/Bowling Alley Refactored	

5. Method lines of code

Measurements tell us about the system

- Metric set the maximum limit to 100.
- It should not be too large. Because larger functions are hard to read.
- Usually large functions tend to have higher chances of bug.

Inference

- Try to keep number of line in any function less.

Functions that violated this constraints are as follows:

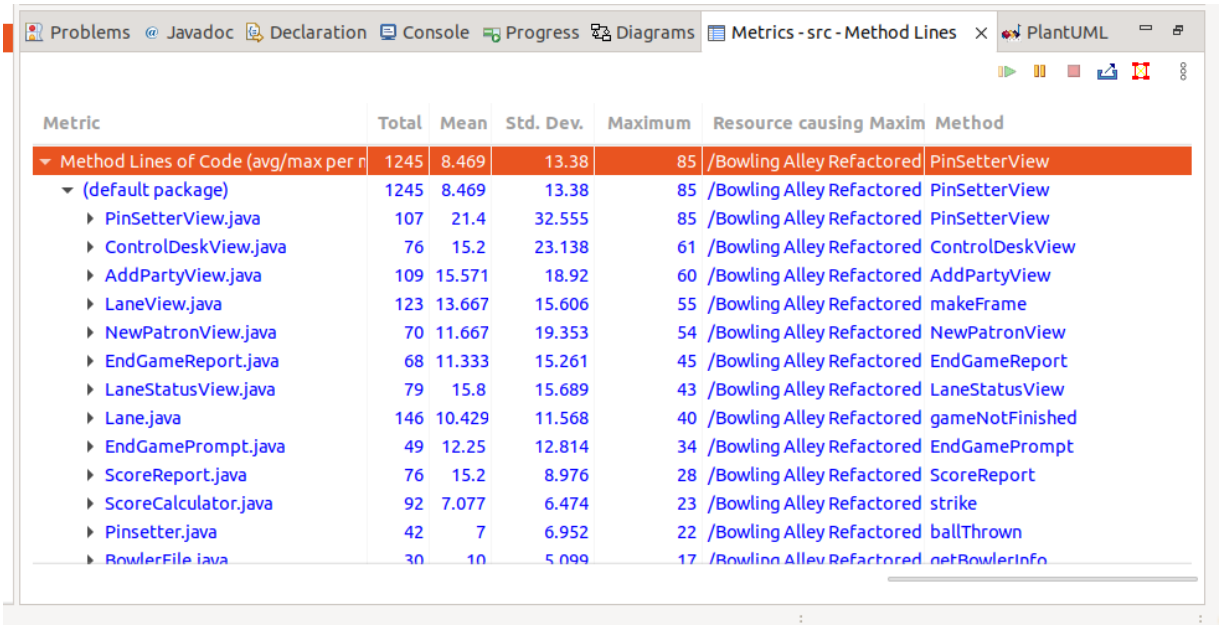
- No function violated any constraints here.

Guide for refactoring

- Decompose large functions into multiple functions.

After Refactoring

- The value decrease a bit. Since we refactore large functions into smaller one. But no significant changes.



Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maxim	Method
Method Lines of Code (avg/max per n	1245	8.469	13.38	85	/Bowling Alley Refactored	PinSetterView
(default package)	1245	8.469	13.38	85	/Bowling Alley Refactored	PinSetterView
PinSetterView.java	107	21.4	32.555	85	/Bowling Alley Refactored	PinSetterView
ControlDeskView.java	76	15.2	23.138	61	/Bowling Alley Refactored	ControlDeskView
AddPartyView.java	109	15.571	18.92	60	/Bowling Alley Refactored	AddPartyView
LaneView.java	123	13.667	15.606	55	/Bowling Alley Refactored	makeFrame
NewPatronView.java	70	11.667	19.353	54	/Bowling Alley Refactored	NewPatronView
EndGameReport.java	68	11.333	15.261	45	/Bowling Alley Refactored	EndGameReport
LaneStatusView.java	79	15.8	15.689	43	/Bowling Alley Refactored	LaneStatusView
Lane.java	146	10.429	11.568	40	/Bowling Alley Refactored	gameNotFinished
EndGamePrompt.java	49	12.25	12.814	34	/Bowling Alley Refactored	EndGamePrompt
ScoreReport.java	76	15.2	8.976	28	/Bowling Alley Refactored	ScoreReport
ScoreCalculator.java	92	7.077	6.474	23	/Bowling Alley Refactored	strike
Pinsetter.java	42	7	6.952	22	/Bowling Alley Refactored	ballThrown
BowlerFile.java	30	10	5.099	17	/Bowling Alley Refactored	getBowlerInfo

6. Depth of Inheritance Tree

Measurements tell us about the system

- Limit on the depth of inheritance was set to 5 by the Metric 2.0.
- Our maximum depth was 2.
- We have removed unnecessary inheritance.
- With such low value of depth of inheritance tree indicate the code base is relatively small and less complex.
- In mid size project if the value is less, it implies poor usage of OOPs concept.

Inference

- The code metric of the original code is strong in this aspects.

Functions that violated this constraints are as follows:

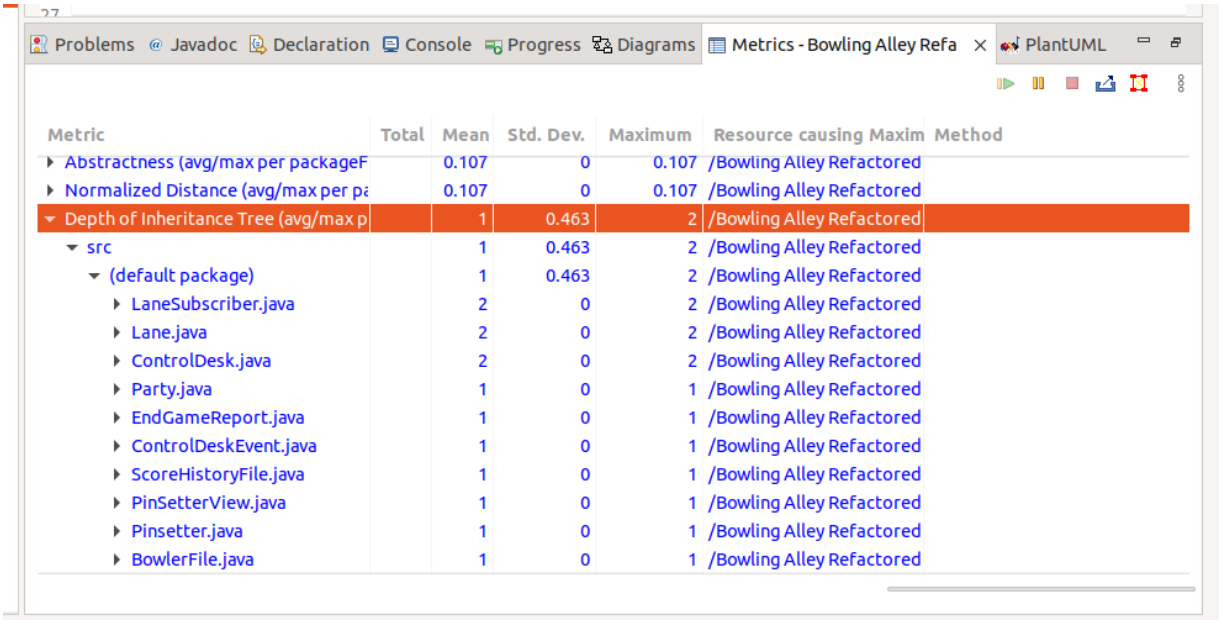
- No function violated any constraints here.

Guide for refactoring

- We made sure that the values remain maintained after recfactoring.
- If the requirements are not ment we can decompose classes.

After Refactoring

- Values retained the same.



Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maxim	Method
Abstractness (avg/max per packageF		0.107	0	0.107	/Bowling Alley Refactored	
Normalized Distance (avg/max per pa		0.107	0	0.107	/Bowling Alley Refactored	
Depth of Inheritance Tree (avg/max p		1	0.463	2	/Bowling Alley Refactored	
src		1	0.463	2	/Bowling Alley Refactored	
(default package)		1	0.463	2	/Bowling Alley Refactored	
LaneSubscriber.java		2	0	2	/Bowling Alley Refactored	
Lane.java		2	0	2	/Bowling Alley Refactored	
ControlDesk.java		2	0	2	/Bowling Alley Refactored	
Party.java		1	0	1	/Bowling Alley Refactored	
EndGameReport.java		1	0	1	/Bowling Alley Refactored	
ControlDeskEvent.java		1	0	1	/Bowling Alley Refactored	
ScoreHistoryFile.java		1	0	1	/Bowling Alley Refactored	
PinSetterView.java		1	0	1	/Bowling Alley Refactored	
Pinsetter.java		1	0	1	/Bowling Alley Refactored	
BowlerFile.java		1	0	1	/Bowling Alley Refactored	

7. Number of methods

Measurements tell us about the system

- Classes with less methods are less complex.
- In the original code there are classes Lane.java which has 17 methods.
- So we need to reduce this as it will difficult for people who will read this in future.

Inference

- Decompose the class into smaller ones. So, lane should be reduced.
- Lane Event have higher getters and setters, so can be ignored.
- Lane Interface can be ignored for the same reasons.

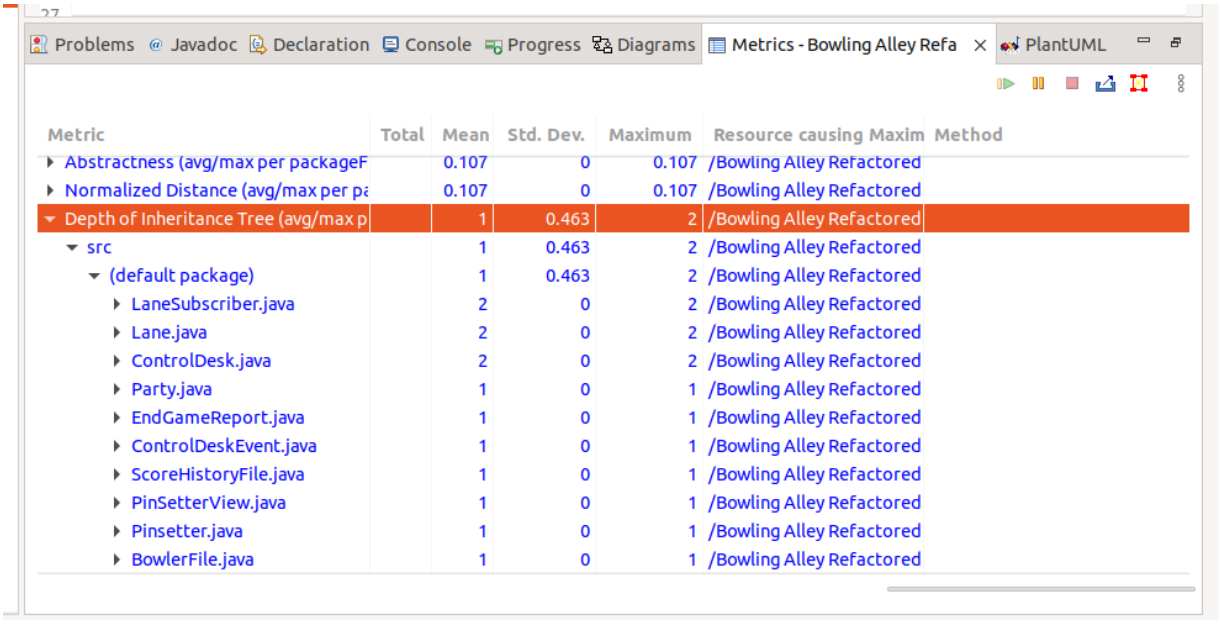
Classes that violated this constraints are as follows:

- Lane - 17
- LaneEvent - 11
- ControlDesk - 11
- LaneEventInterface - 9

Guide for refactoring

- We tried to decompose the Lane class into two more classes. ScoreCalculator and Lanesubsribe.
- To reduce the complexity. We were only able to reduce the dominant Lane Class.
- Removed unused methods.

After Refactoring



Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maxim	Method
Abstractness (avg/max per packageF		0.107	0	0.107	/Bowling Alley Refactored	
Normalized Distance (avg/max per pa		0.107	0	0.107	/Bowling Alley Refactored	
Depth of Inheritance Tree (avg/max p		1	0.463	2	/Bowling Alley Refactored	
src		1	0.463	2	/Bowling Alley Refactored	
(default package)		1	0.463	2	/Bowling Alley Refactored	
LaneSubscriber.java		2	0	2	/Bowling Alley Refactored	
Lane.java		2	0	2	/Bowling Alley Refactored	
ControlDesk.java		2	0	2	/Bowling Alley Refactored	
Party.java		1	0	1	/Bowling Alley Refactored	
EndGameReport.java		1	0	1	/Bowling Alley Refactored	
ControlDeskEvent.java		1	0	1	/Bowling Alley Refactored	
ScoreHistoryFile.java		1	0	1	/Bowling Alley Refactored	
PinSetterView.java		1	0	1	/Bowling Alley Refactored	
Pinsetter.java		1	0	1	/Bowling Alley Refactored	
BowlerFile.java		1	0	1	/Bowling Alley Refactored	

8. Number of classes

Measurements tell us about the system

- Higher number of classes shows the more complex systems.
- Earlier we have 29 classes, now we have increased this.
- It tells the project is relatively small sized.

Inference

- The code metric has given strong indication in this sense.
- So we can try to add a few extra classes by decomposing current one.

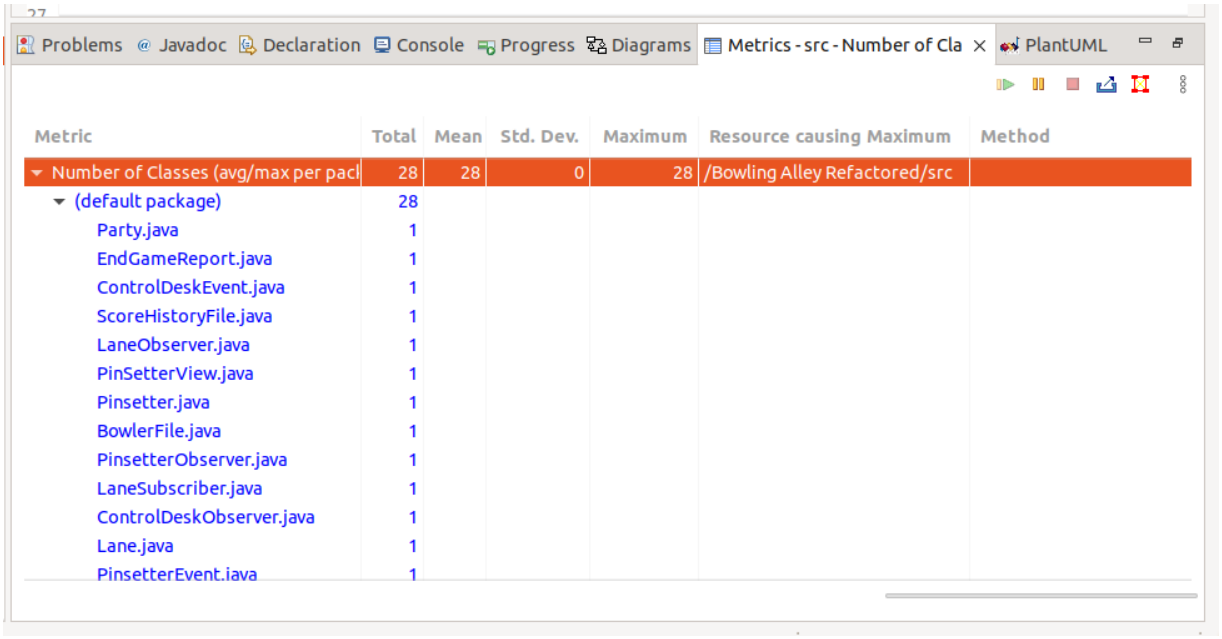
Violated of metric Limit:

- None

Guide for refactoring

- We tried to keep the metric value same.
- We have reduced useless classes like Alley and some interface.
- We have decomposed the Lane class and added two new class.
- We removed 3 classes and added 2 extra class. So keeping the count same to 28.

After Refactoring


Problems
Javadoc
Declaration
Console
Progress
Diagrams
Metrics - src - Number of Cla
PlantUML
Metric
Total
Mean
Std. Dev.
Maximum
Resource causing Maximum
Method
Number of Classes (avg/max per pacl
28
28
0
28
/Bowling Alley Refactored/src
(default package)
28
Party.java
1
EndGameReport.java
1
ControlDeskEvent.java
1
ScoreHistoryFile.java
1
LaneObserver.java
1
PinSetterView.java
1
Pinsetter.java
1
BowlerFile.java
1
PinsetterObserver.java
1
LaneSubscriber.java
1
ControlDeskObserver.java
1
Lane.java
1
PinsetterEvent.java
1