Software Engineering

Unit-1 Project

Bowling Alley

Team Number-12 :

Akshay Phate (2021201033)

Haridasu Yaswanth (2021201032)

Darshit Khant (2020201085)

Vishal Patel (2020201082)

Github: https://github.com/akshayphate/bowlingalley

Effort and Roles:

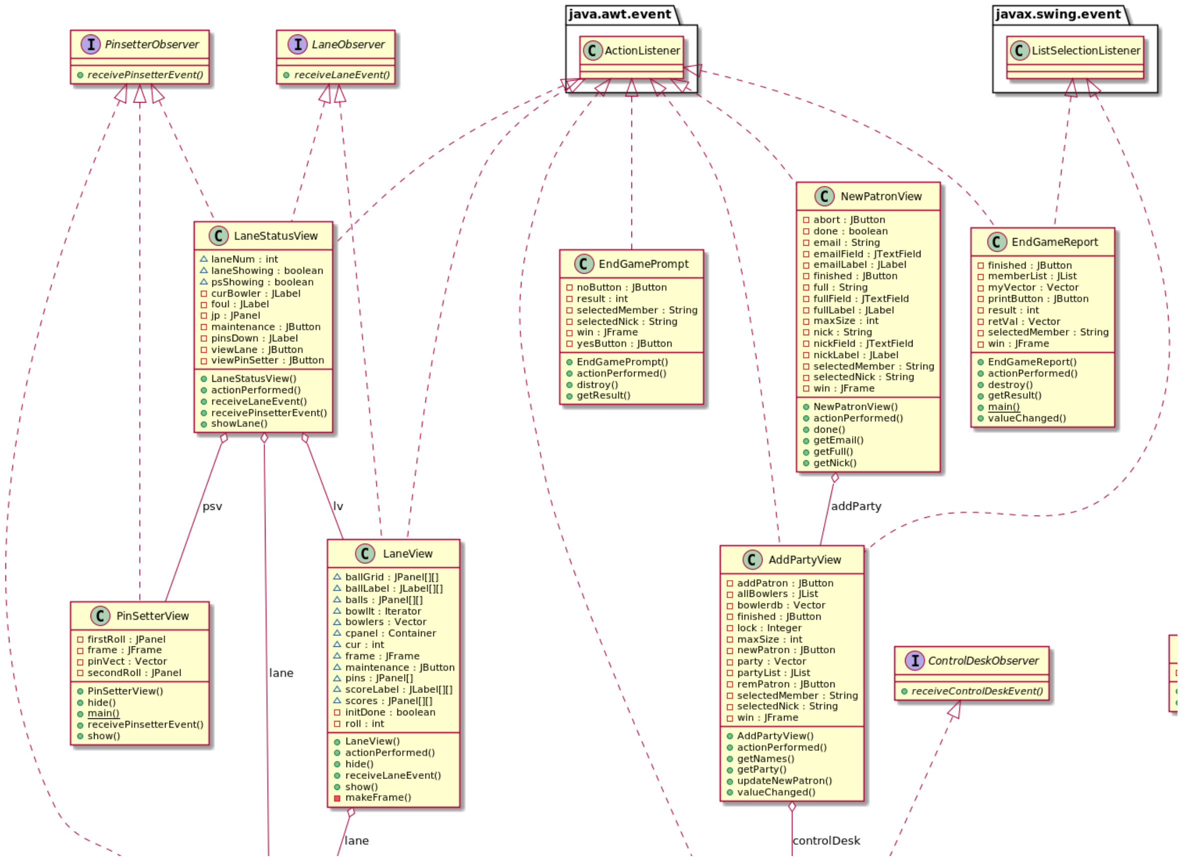
Name	**Modules worked on**	**No.of hours put**
Akshay Phate	Design of the refactored application Segregating the classes into packages Creating Class diagrams for the existing and refactored design	10 hrs
Haridasu Yaswanth	Analysis of pre-refactored design Identifying the code smells and removing them Creating Sequence diagrams for the existing and refactored design	8-9 hrs
Darshit Khant	Removing useless classes Removing redundancy in user interface (views)	6-7 hrs
Vishal Patel	Creating utility class for score generation Extracting methods	6-7 hrs

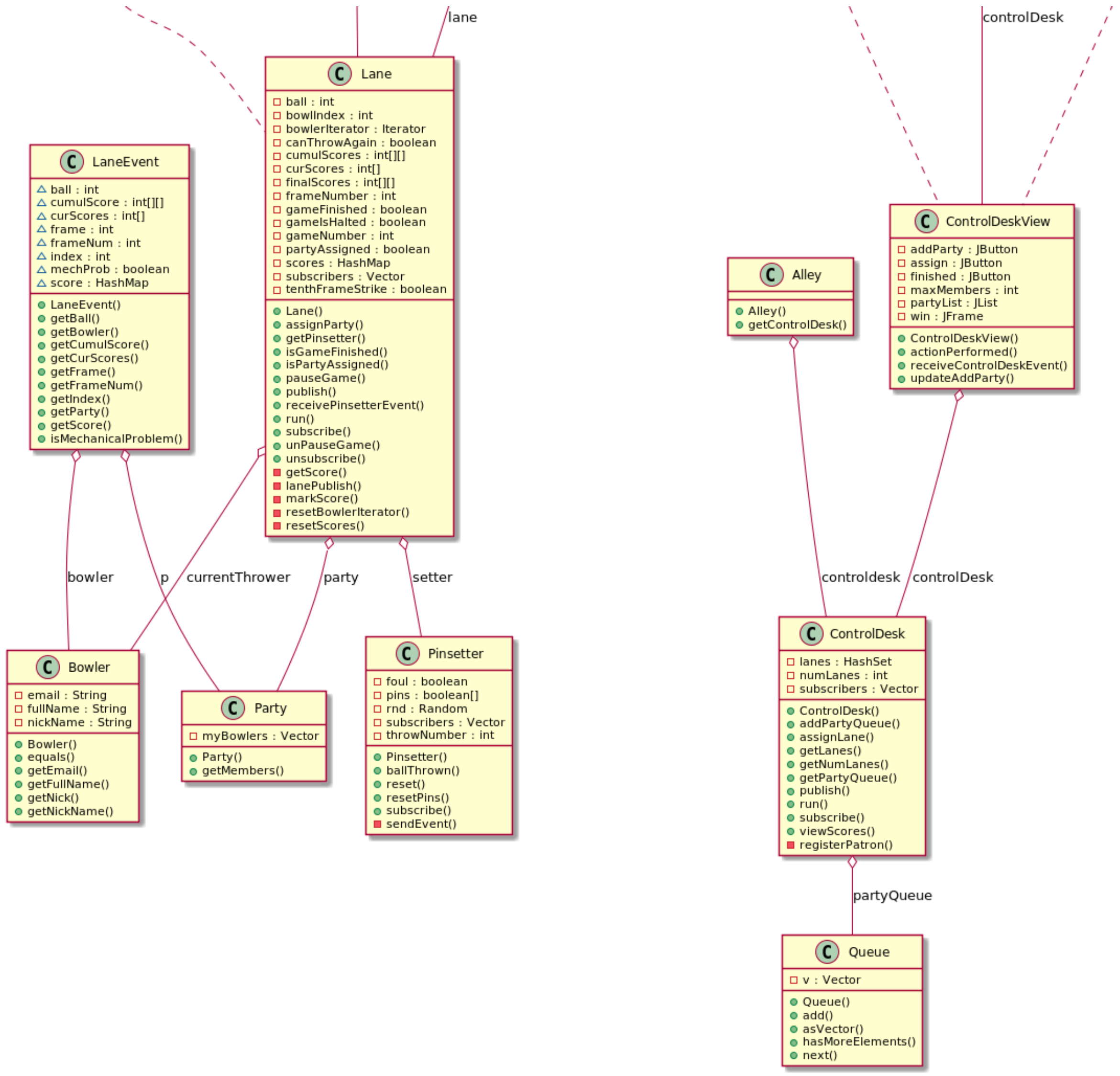
Overview:

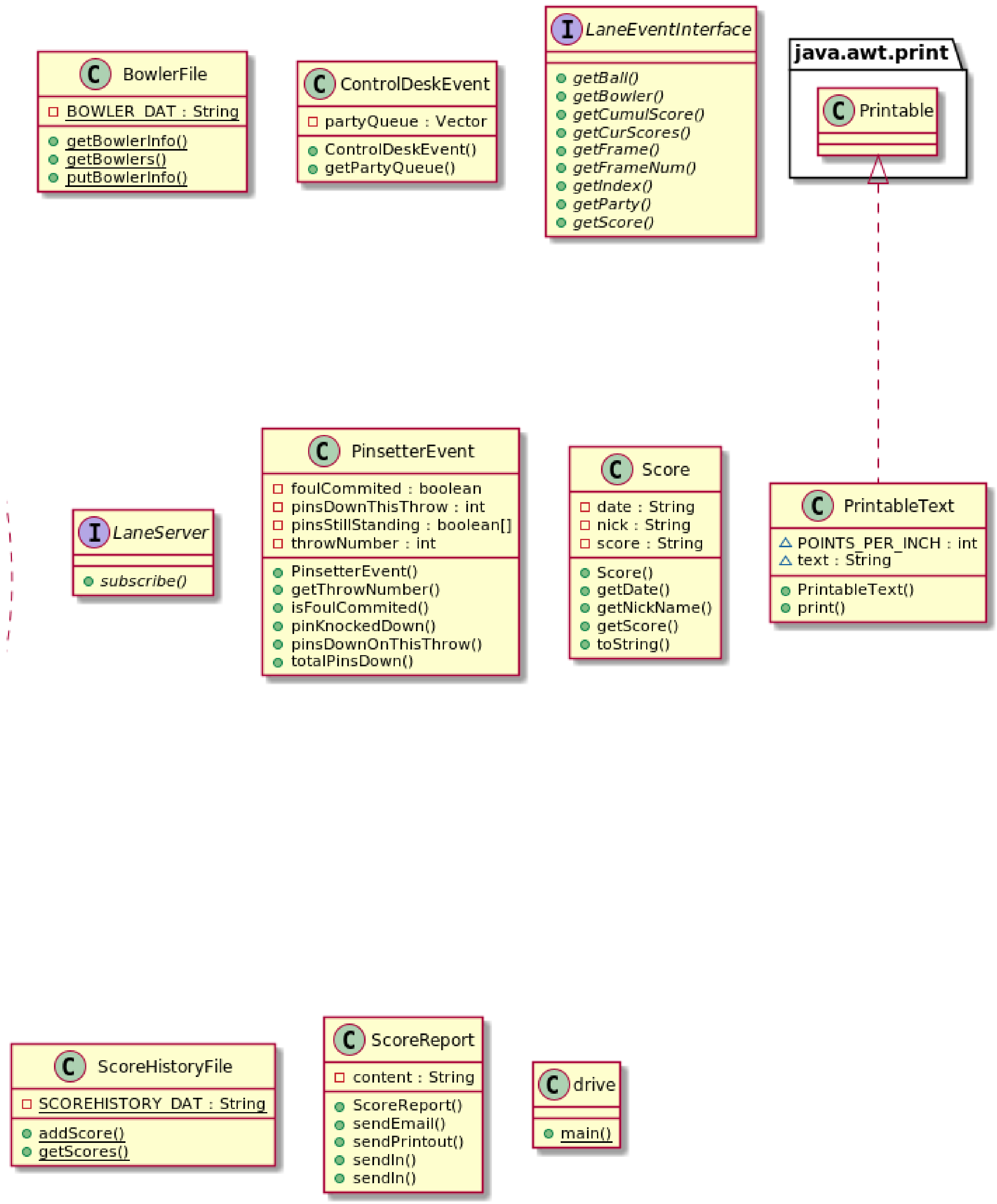
The project is a working prototype of a bowling alley management system. The system was developed in an earlier version of Java (Java 5) that did not support templates. A bowling alley consists of many lanes. When the bowlers enter the alley, they check in at the control desk for a lane assignment. Bowlers may check in as a party so that they will be assigned to the same lane. Once a party has started bowling, the control desk monitors the number of frames completed by each bowler. When a party has checked out at the control desk after completing the game, a report is generated containing the bowlers' scores.

Class Diagrams for pre-refactored design:

SRC's Class Diagram

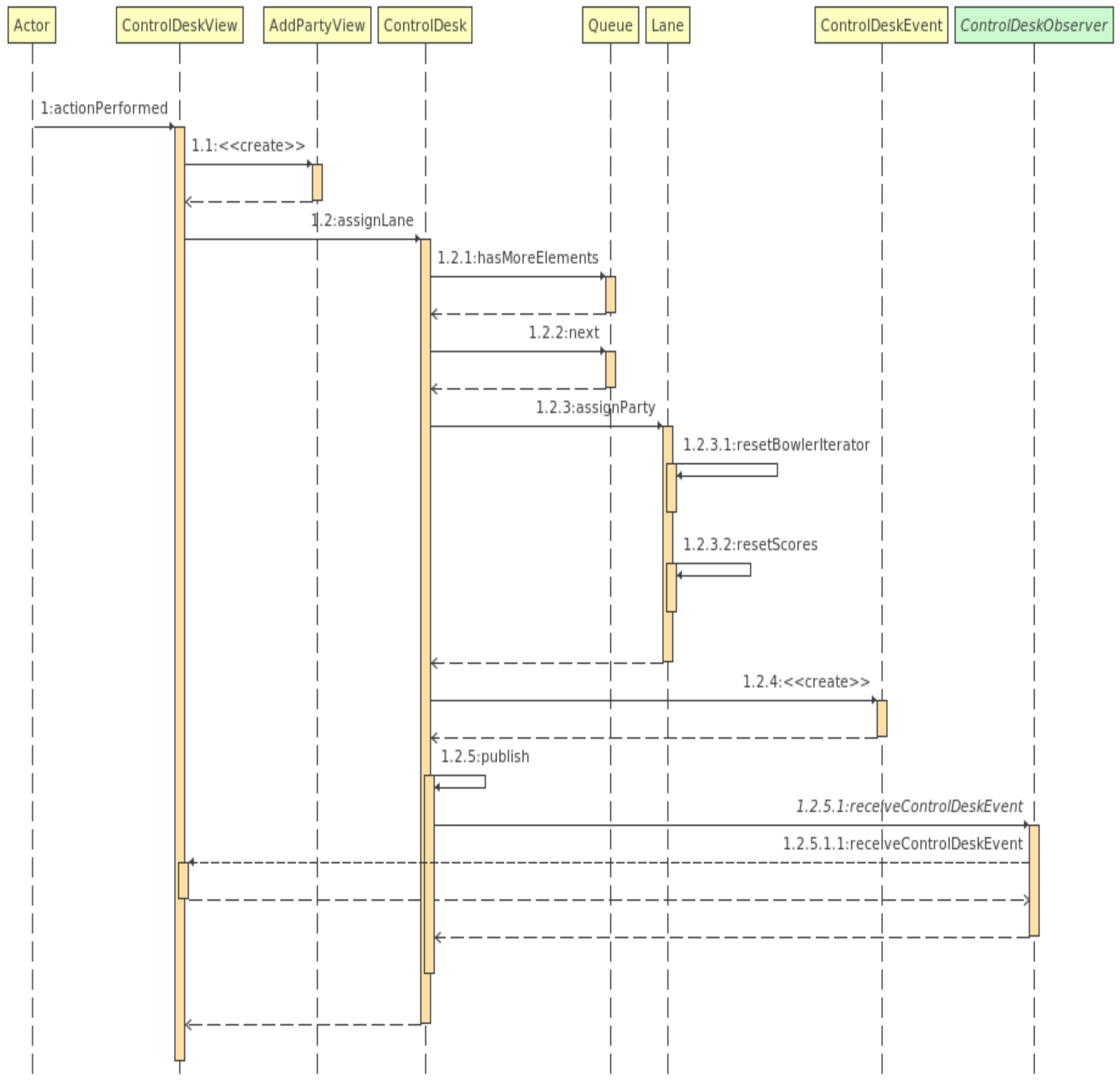



lane
Lane
- ball : int
- bowlIndex : int
- bowlerIterator : Iterator
- canThrowAgain : boolean
- cumulScores : int[][]
- curScores : int[]
- finalScores : int[][]
- frameNumber : int
- gameFinished : boolean
- gameIsHalted : boolean
- gameNumber : int
- partyAssigned : boolean
- scores : HashMap
- subscribers : Vector
- tenthFrameStrike : boolean
+ Lane()
+ assignParty()
+ getPinsetter()
+ isGameFinished()
+ isPartyAssigned()
+ pauseGame()
+ publish()
+ receivePinsetterEvent()
+ run()
+ subscribe()
+ unPauseGame()
+ unsubscribe()
- getScore()
- lanePublish()
- markScore()
- resetBowlerIterator()
- resetScores()
LaneEvent
~ ball : int
~ cumulScore : int[][]
~ curScores : int[]
~ frame : int
~ frameNum : int
~ index : int
~ mechProb : boolean
~ score : HashMap
+ LaneEvent()
+ getBall()
+ getBowler()
+ getCumulScore()
+ getCurScores()
+ getFrame()
+ getFrameNum()
+ getIndex()
+ getParty()
+ getScore()
+ isMechanicalProblem()
bowler
p
currentThrower
party
setter
Bowler
- email : String
- fullName : String
- nickName : String
+ Bowler()
+ equals()
+ getEmail()
+ getFullName()
+ getNick()
+ getNickName()
Party
- myBowlers : Vector
+ Party()
+ getMembers()
Pinsetter
- foul : boolean
- pins : boolean[]
- rnd : Random
- subscribers : Vector
- throwNumber : int
+ Pinsetter()
+ ballThrown()
+ reset()
+ resetPins()
+ subscribe()
- sendEvent()
controlDesk
ControlDeskView
- addParty : JButton
- assign : JButton
- finished : JButton
- maxMembers : int
- partyList : JList
- win : JFrame
+ ControlDeskView()
+ actionPerformed()
+ receiveControlDeskEvent()
+ updateAddParty()
Alley
+ Alley()
+ getControlDesk()
controldesk
controlDesk
ControlDesk
- lanes : HashSet
- numLanes : int
- subscribers : Vector
+ ControlDesk()
+ addPartyQueue()
+ assignLane()
+ getLanes()
+ getNumLanes()
+ getPartyQueue()
+ publish()
+ run()
+ subscribe()
+ viewScores()
- registerPatron()
partyQueue
Queue
- v : Vector
+ Queue()
+ add()
+ asVector()
+ hasMoreElements()
+ next()

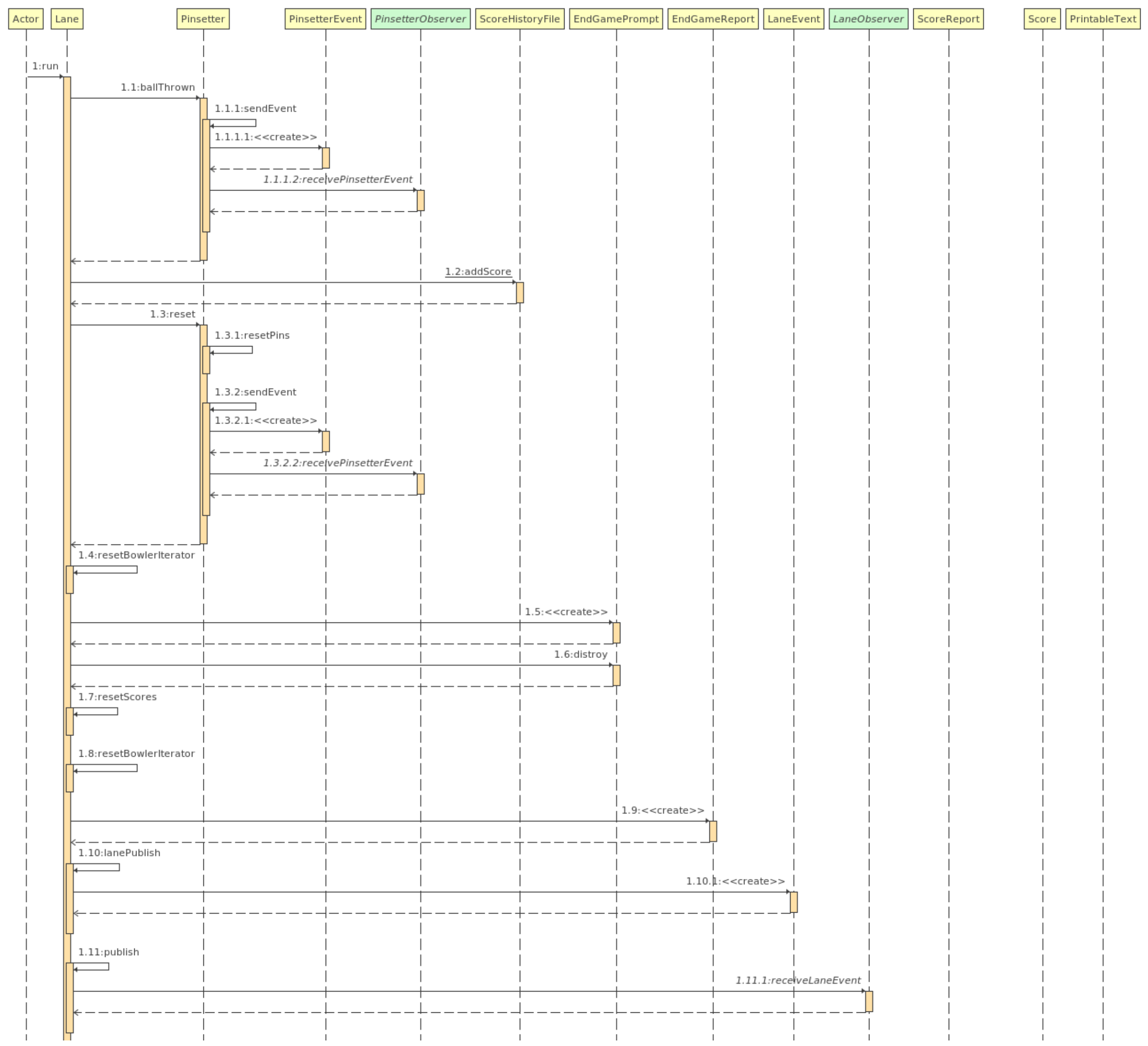

BowlerFile
BOWLER_DAT : String
getBowlerInfo()
getBowlers()
putBowlerInfo()
ControlDeskEvent
partyQueue : Vector
ControlDeskEvent()
getPartyQueue()
LaneEventInterface
getBall()
getBowler()
getCumulScore()
getCurScores()
getFrame()
getFrameNum()
getIndex()
getParty()
getScore()
java.awt.print
Printable
LaneServer
subscribe()
PinsetterEvent
foulCommited : boolean
pinsDownThisThrow : int
pinsStillStanding : boolean[]
throwNumber : int
PinsetterEvent()
getThrowNumber()
isFoulCommited()
pinKnockedDown()
pinsDownOnThisThrow()
totalPinsDown()
Score
date : String
nick : String
score : String
Score()
getDate()
getNickName()
getScore()
toString()
PrintableText
POINTS_PER_INCH : int
text : String
PrintableText()
print()
ScoreHistoryFile
SCOREHISTORY_DAT : String
addScore()
getScores()
ScoreReport
content : String
ScoreReport()
sendEmail()
sendPrintout()
sendln()
sendln()
drive
main()

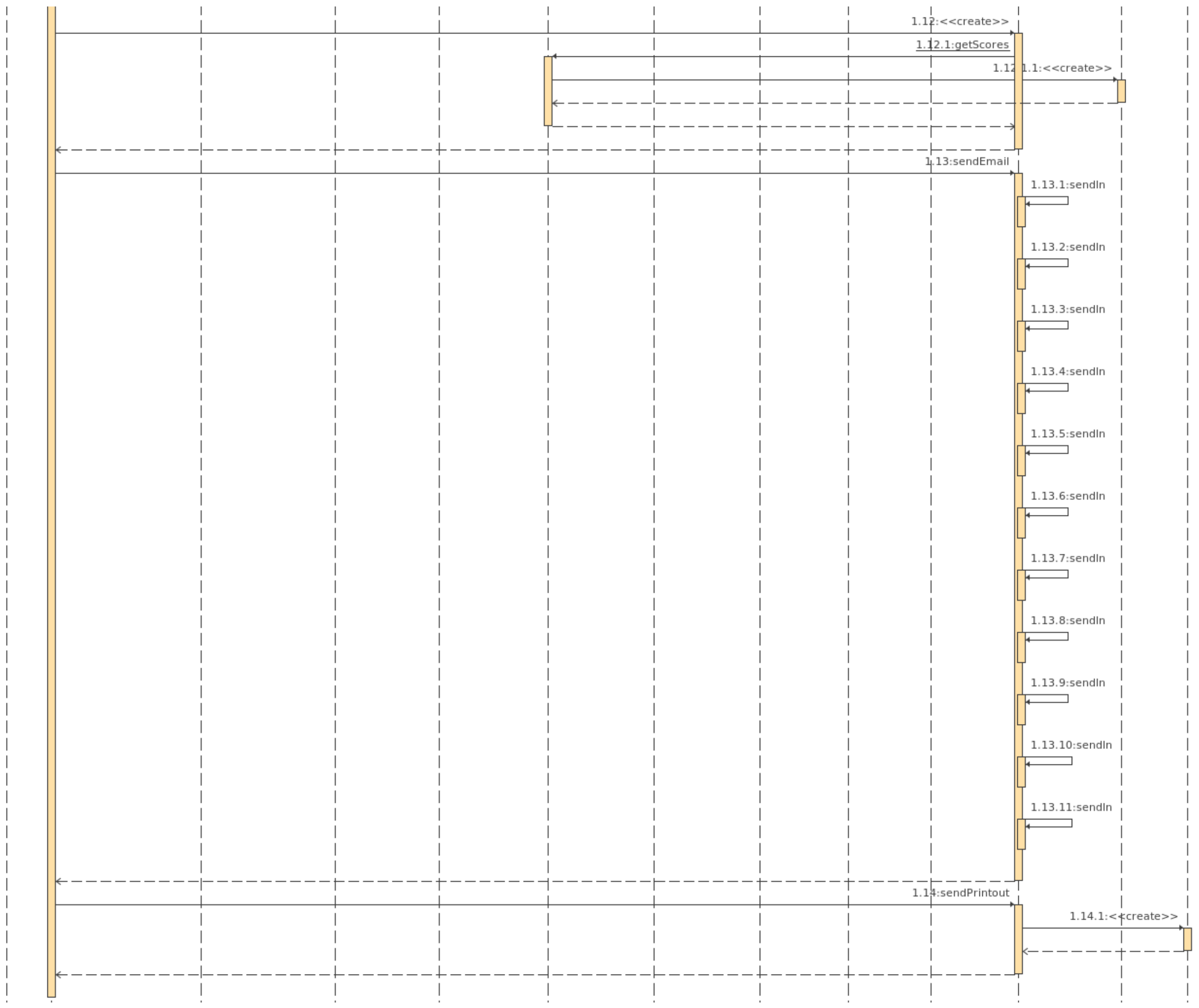
Sequence Diagrams for pre-refactored design:

ControlDeskView actionPerformed():

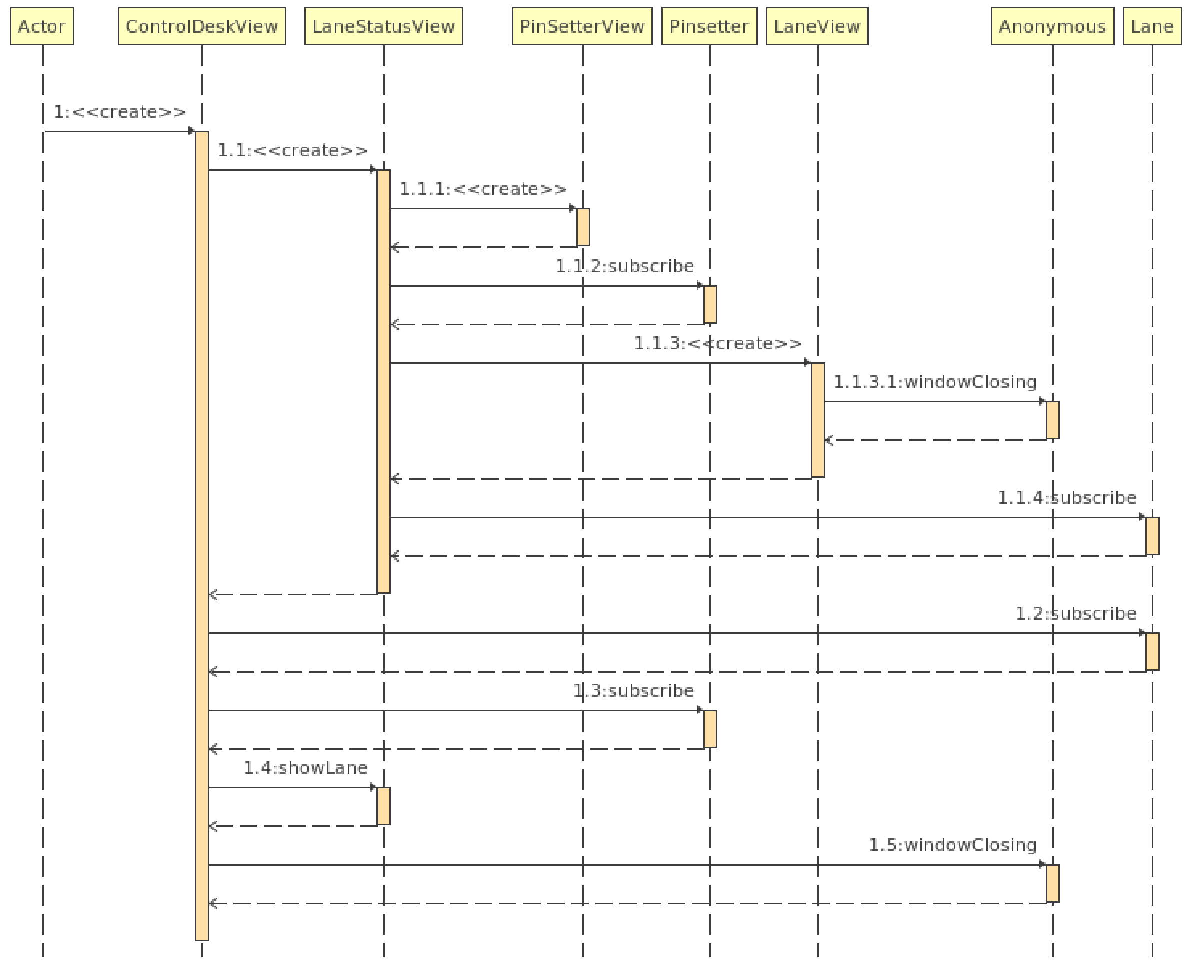


Lane run():

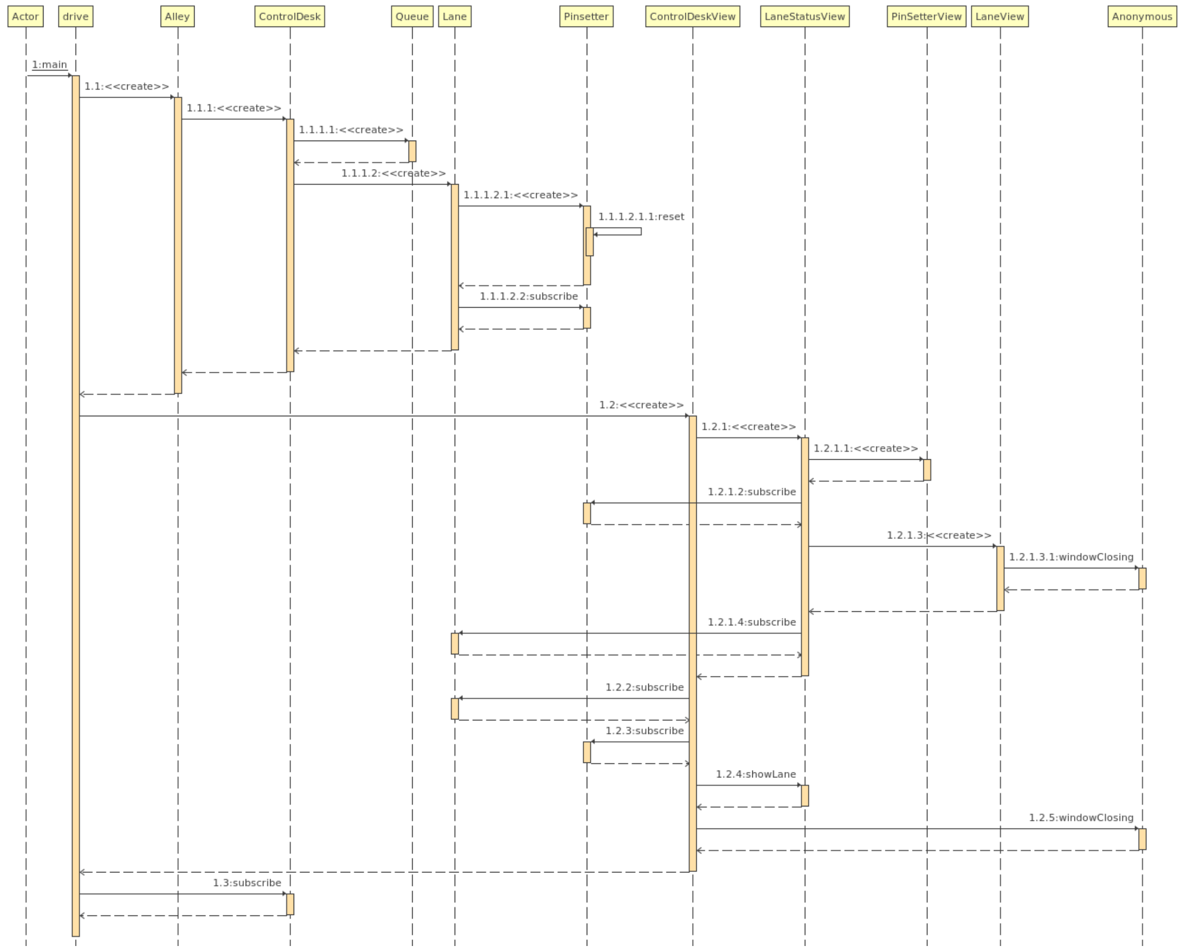

Actor
Lane
Pinsetter
PinsetterEvent
PinsetterObserver
ScoreHistoryFile
EndGamePrompt
EndGameReport
LaneEvent
LaneObserver
ScoreReport
Score
PrintableText
1:run
1.1:ballThrown
1.1.1:sendEvent
1.1.1.1:<<create>>
1.1.1.2:receivePinsetterEvent
1.2:addScore
1.3:reset
1.3.1:resetPins
1.3.2:sendEvent
1.3.2.1:<<create>>
1.3.2.2:receivePinsetterEvent
1.4:resetBowlerIterator
1.5:<<create>>
1.6:distroy
1.7:resetScores
1.8:resetBowlerIterator
1.9:<<create>>
1.10:lanePublish
1.10.1:<<create>>
1.11:publish
1.11.1:receiveLaneEvent


1.12:<<create>>
1.12.1:getScores
1.12.1.1:<<create>>
1.13:sendEmail
1.13.1:sendln
1.13.2:sendln
1.13.3:sendln
1.13.4:sendln
1.13.5:sendln
1.13.6:sendln
1.13.7:sendln
1.13.8:sendln
1.13.9:sendln
1.13.10:sendln
1.13.11:sendln
1.14:sendPrintout
1.14.1:<<create>>

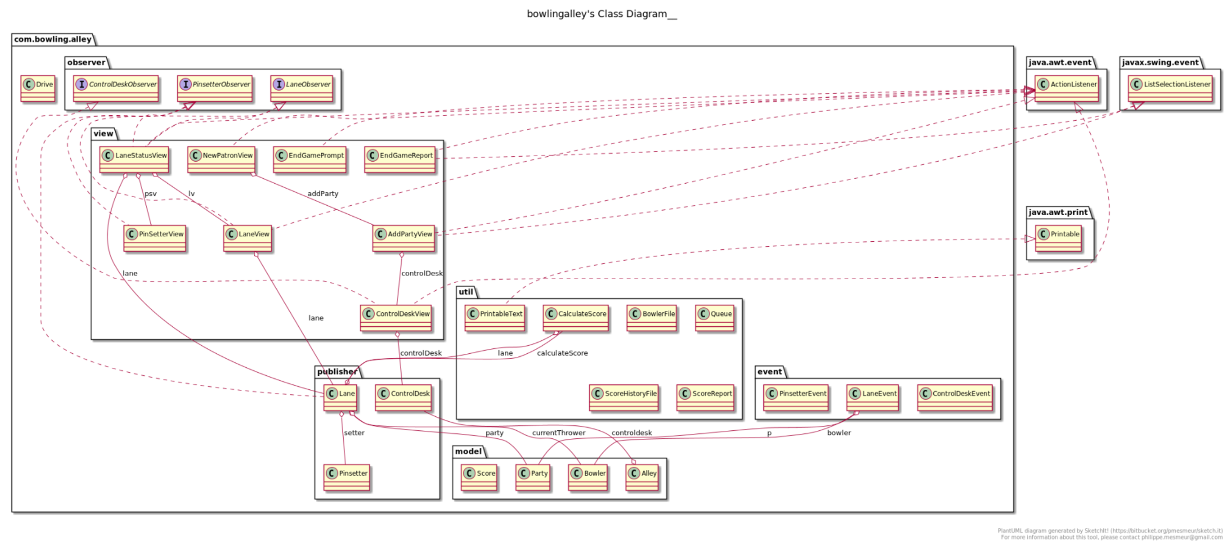
ControlDeskView constructor():



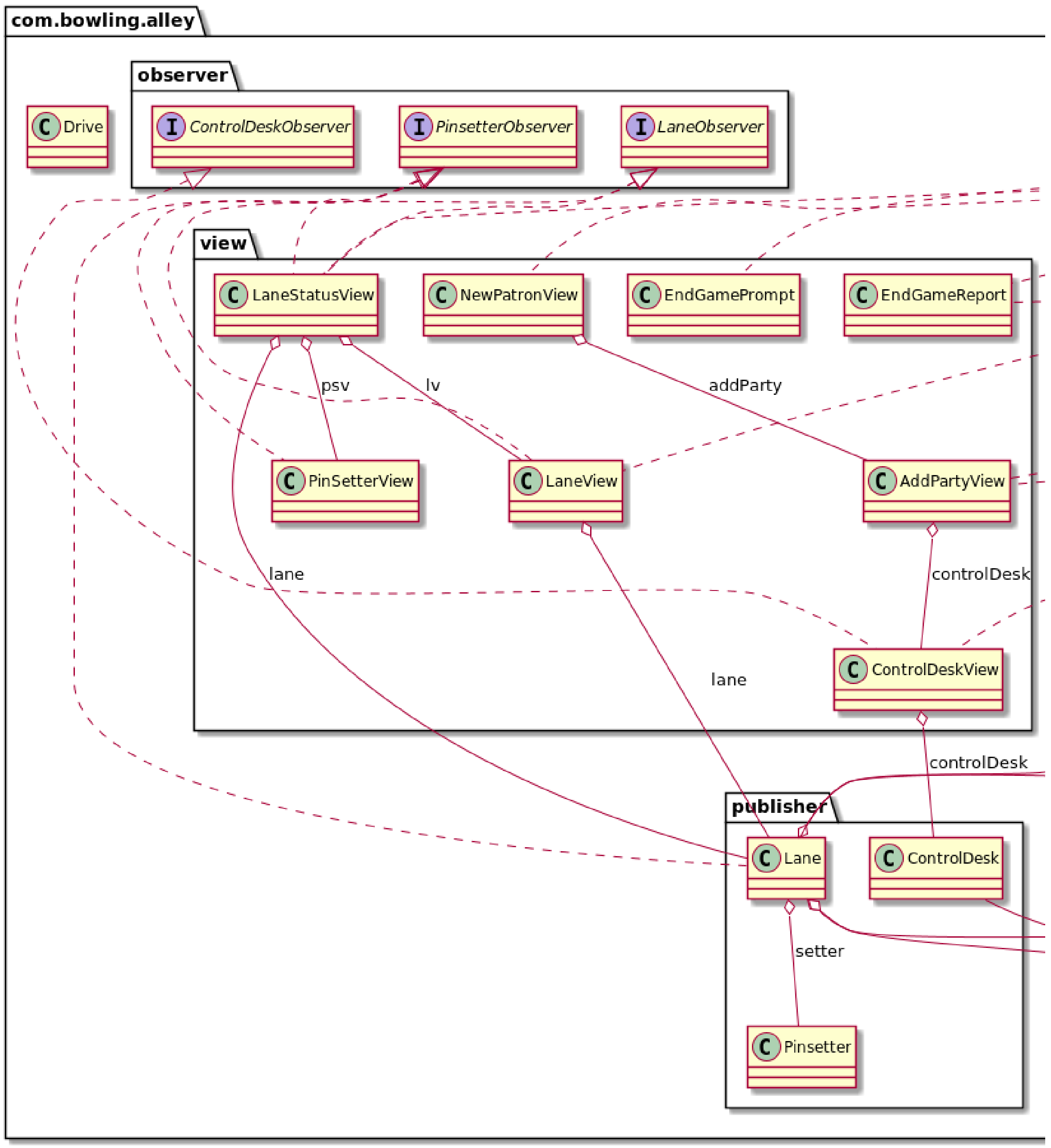
Drive main():

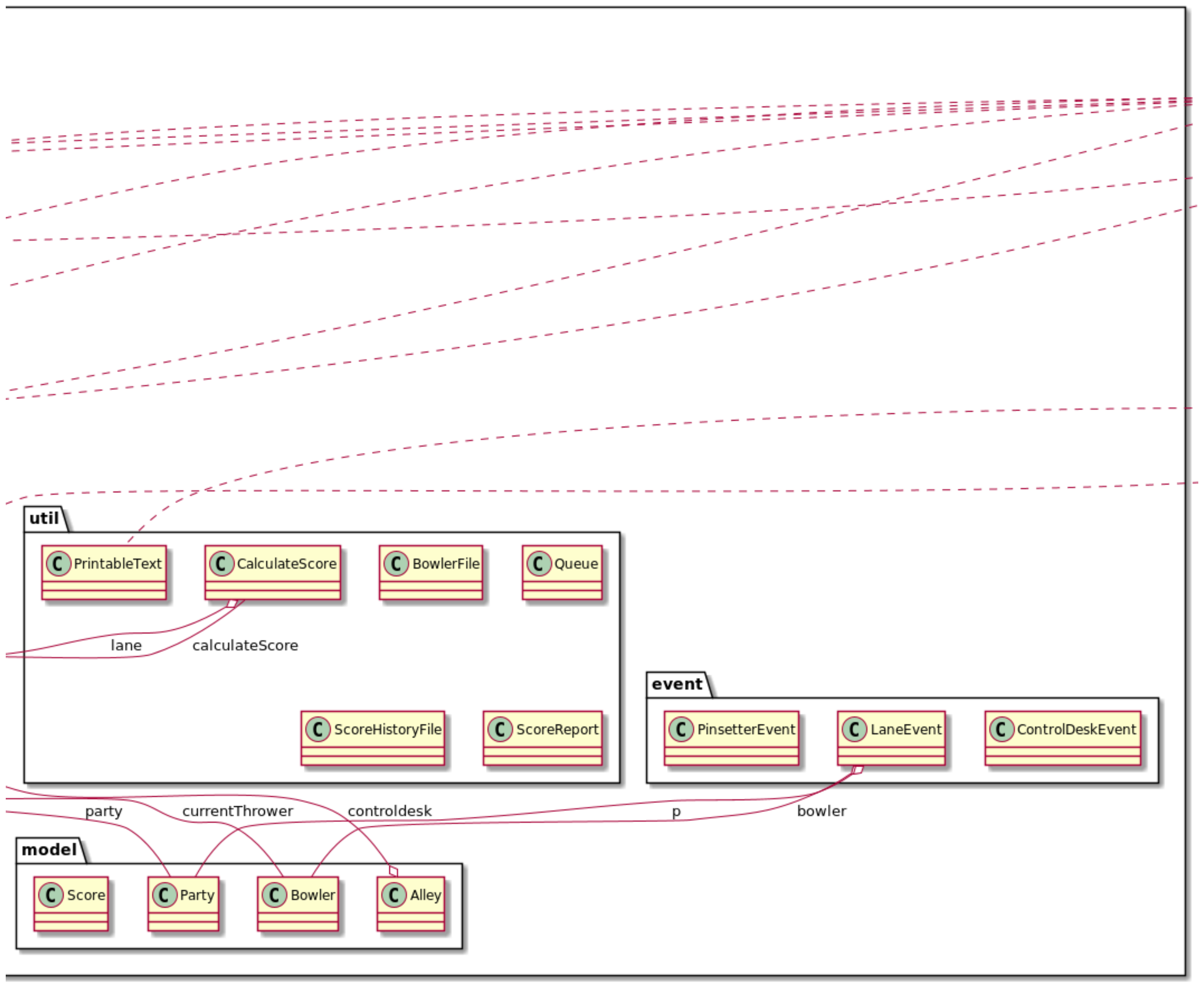

Actor
drive
Alley
ControlDesk
Queue
Lane
Pinsetter
ControlDeskView
LaneStatusView
PinSetterView
LaneView
Anonymous
1:main
1.1:<<create>>
1.1.1:<<create>>
1.1.1.1:<<create>>
1.1.1.2:<<create>>
1.1.1.2.1:<<create>>
1.1.1.2.1.1:reset
1.1.1.2.2:subscribe
1.2:<<create>>
1.2.1:<<create>>
1.2.1.1:<<create>>
1.2.1.2:subscribe
1.2.1.3:<<create>>
1.2.1.3.1:windowClosing
1.2.1.4:subscribe
1.2.2:subscribe
1.2.3:subscribe
1.2.4:showLane
1.2.5:windowClosing
1.3:subscribe

Class Diagrams for refactored design:

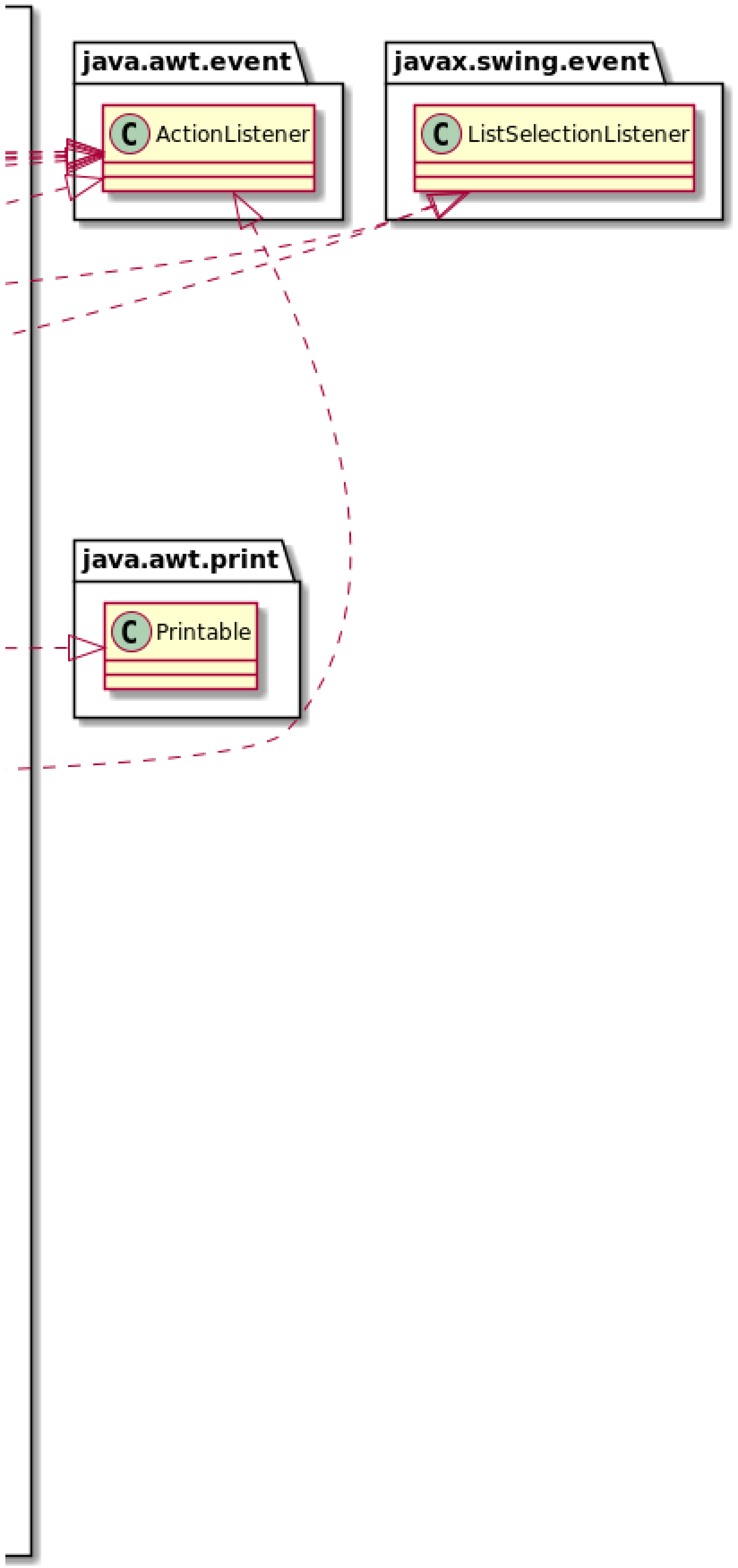


Dividing the image into three separate images and displaying them below for better clarity purpose:

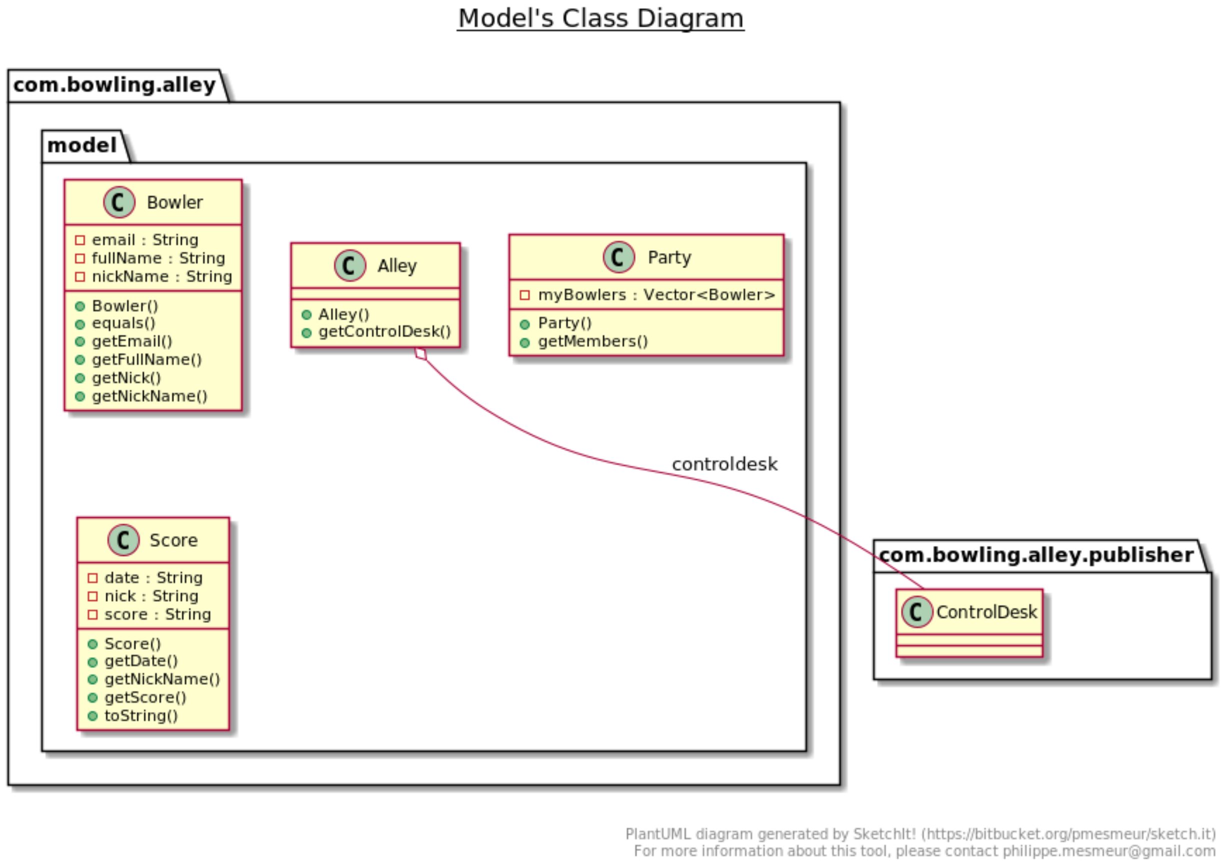

com.bowling.alley
observer
Drive
ControlDeskObserver
PinsetterObserver
LaneObserver
view
LaneStatusView
NewPatronView
EndGamePrompt
EndGameReport
psv
lv
addParty
PinSetterView
LaneView
AddPartyView
lane
controlDesk
lane
ControlDeskView
controlDesk
publisher
Lane
ControlDesk
setter
Pinsetter



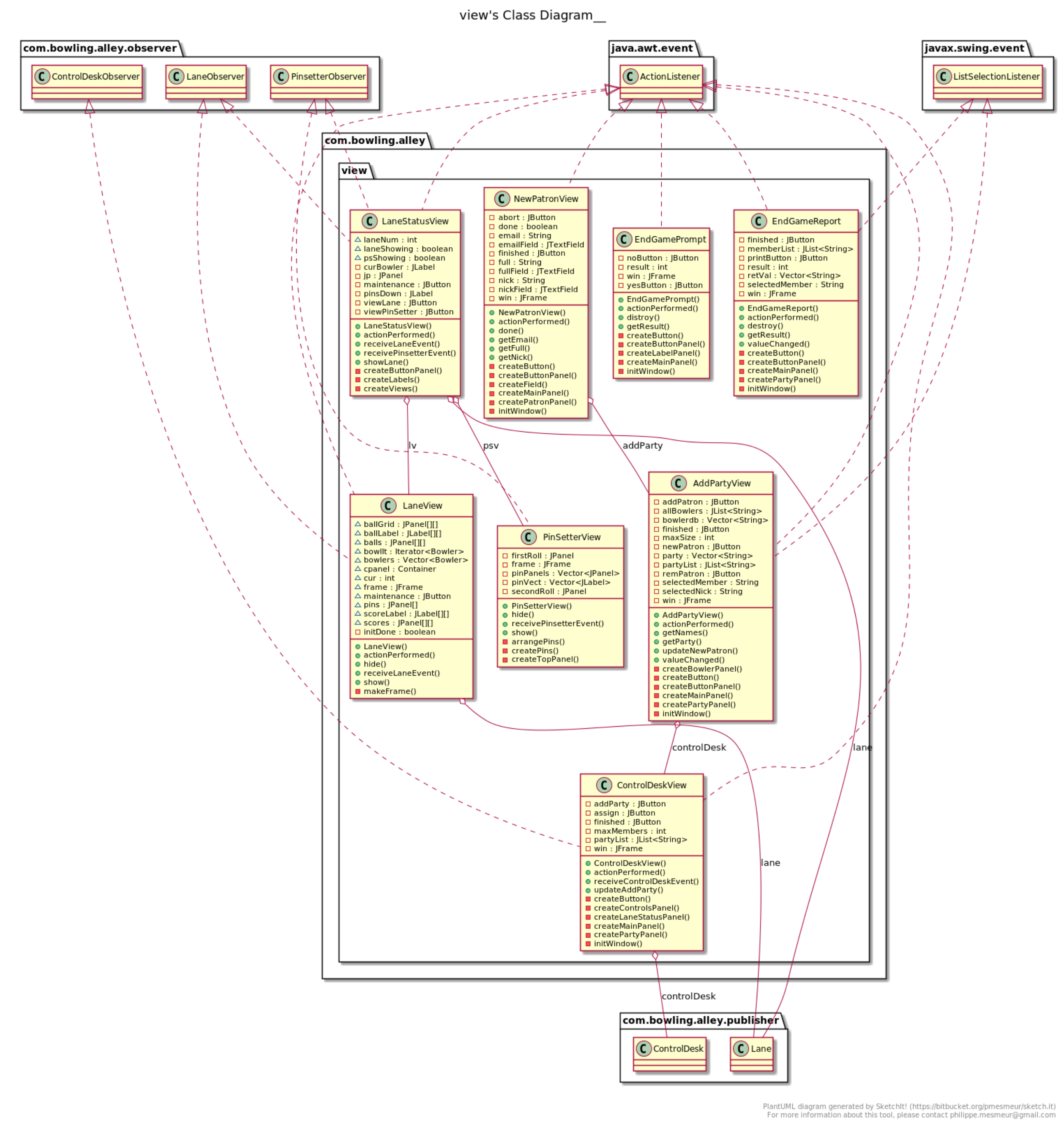
PlantUM
For mo


java.awt.event
ActionListener
javax.swing.event
ListSelectionListener
java.awt.print
Printable

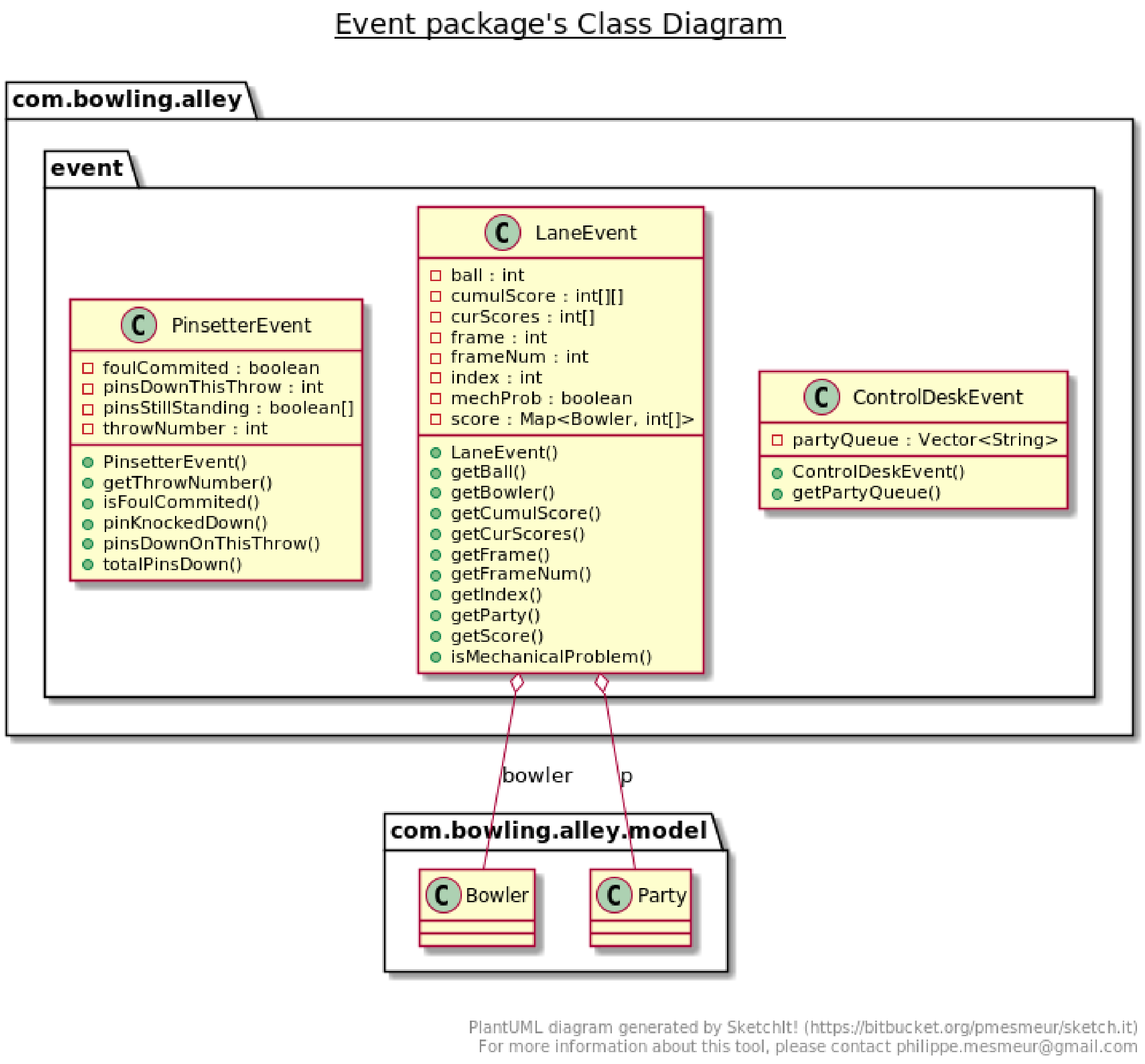
Class diagram for model package:



Class diagram for view package:

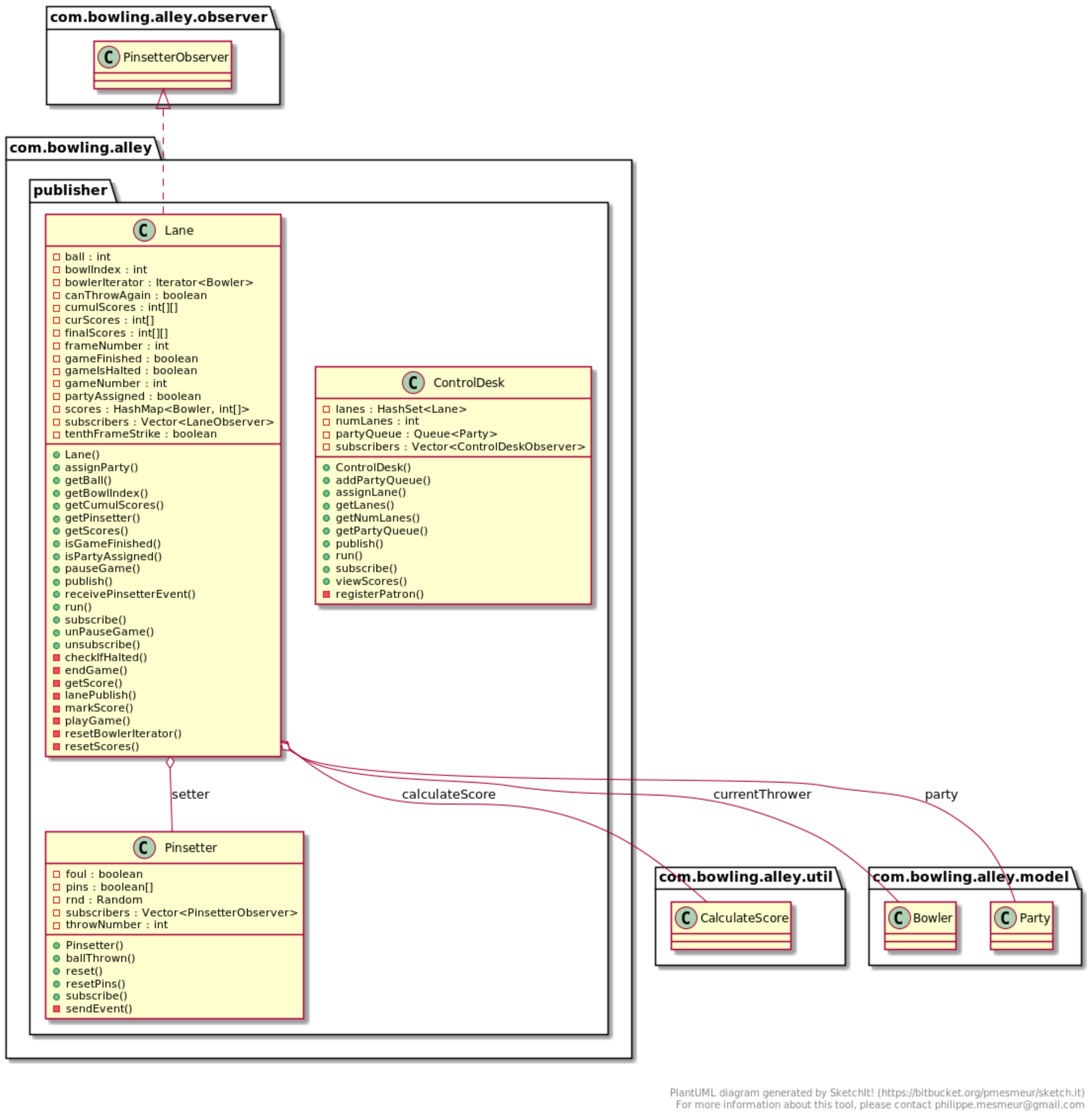


Class diagram for event package:



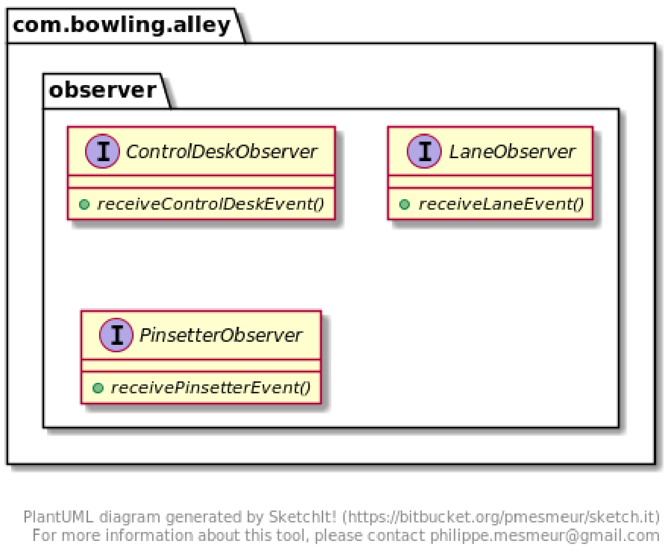
Class diagram for publisher package:

Publisher's Class Diagram



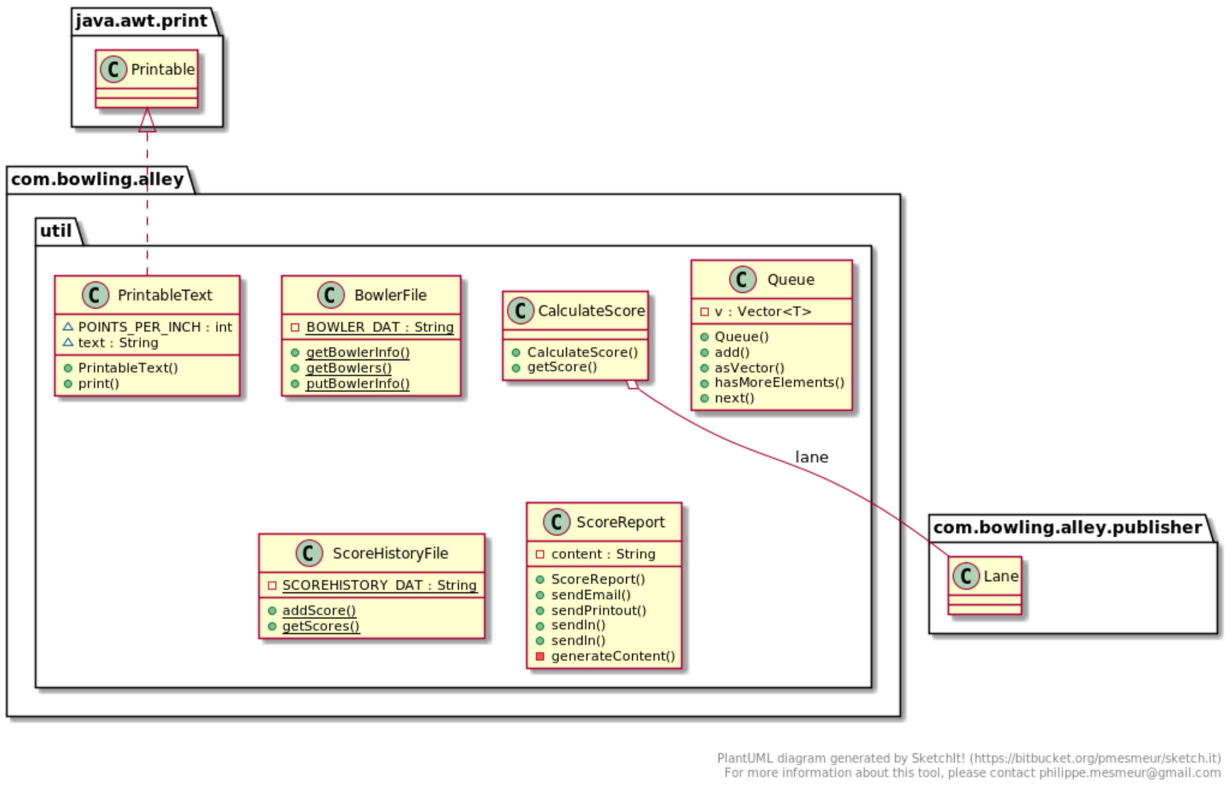
Class diagram for observer package:

Observer's Class Diagram


com.bowling.alley
observer
ControlDeskObserver
receiveControlDeskEvent()
LaneObserver
receiveLaneEvent()
PinsetterObserver
receivePinsetterEvent()
PlantUML diagram generated by SketchIt! (https://bitbucket.org/pmesmeur/sketch.it)
For more information about this tool, please contact philippe.mesmeur@gmail.com

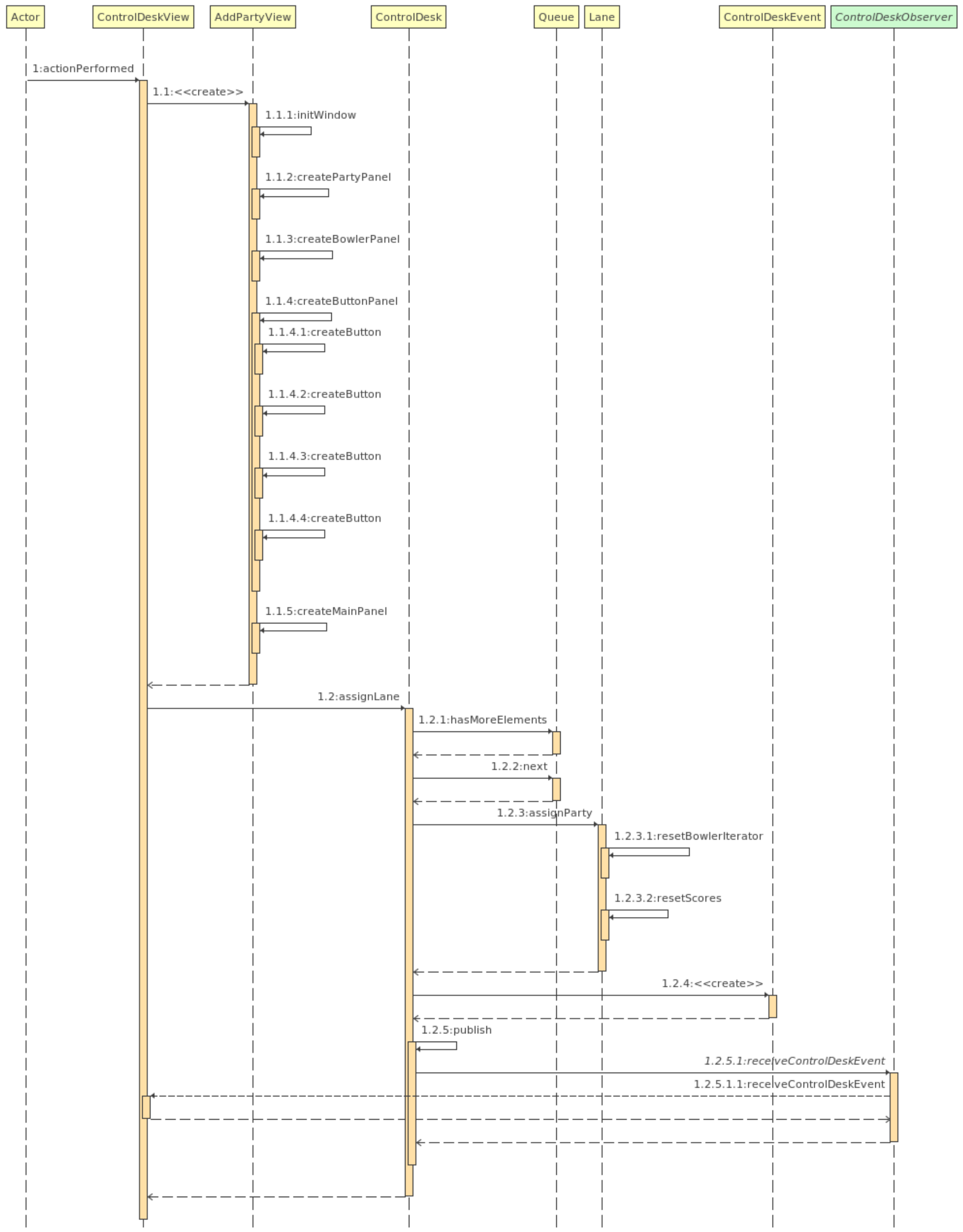
Class diagram for util package:

Util's Class Diagram

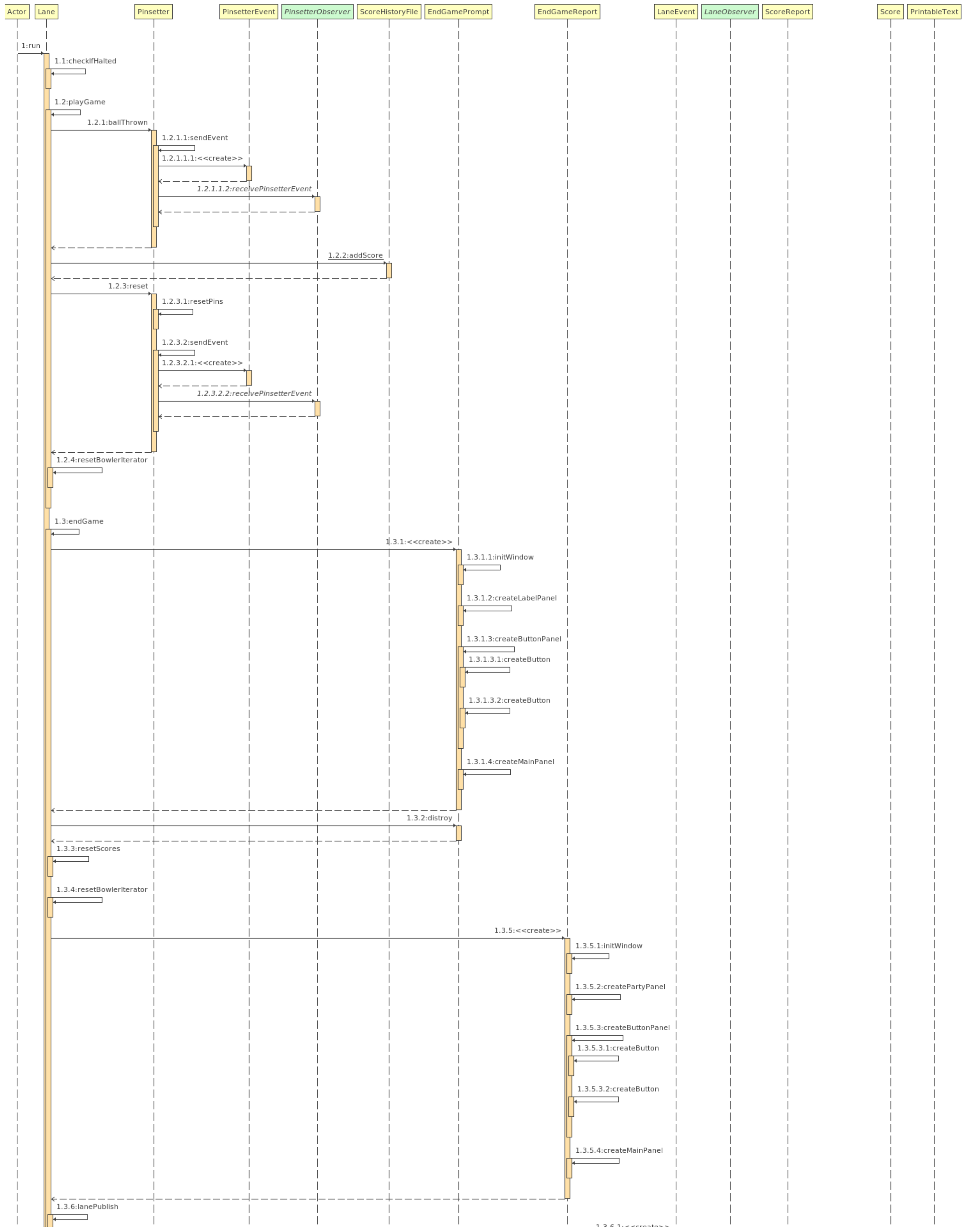


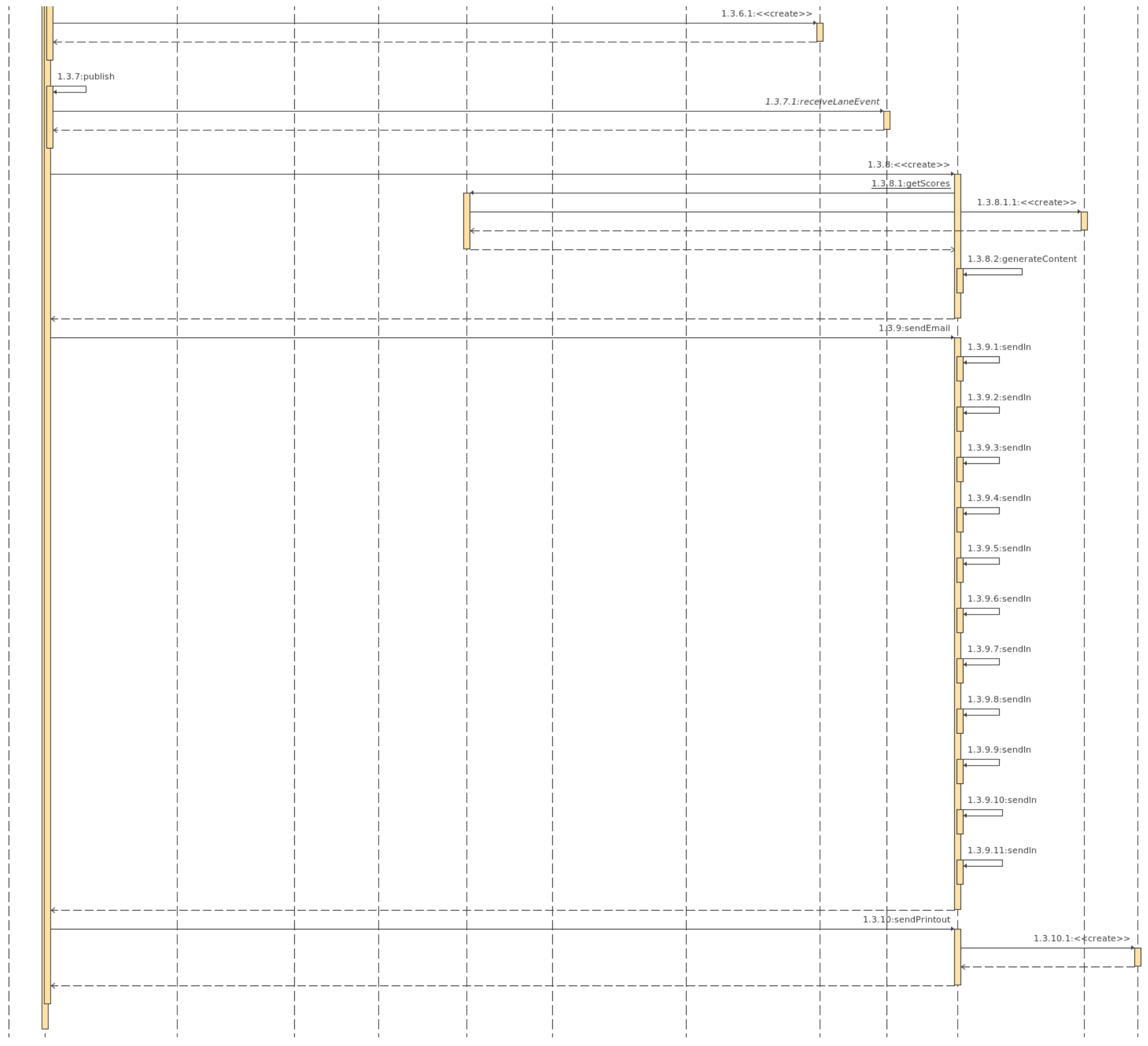
Sequence Diagrams for pre-refactored design:

ControlDeskView actionPerformed():

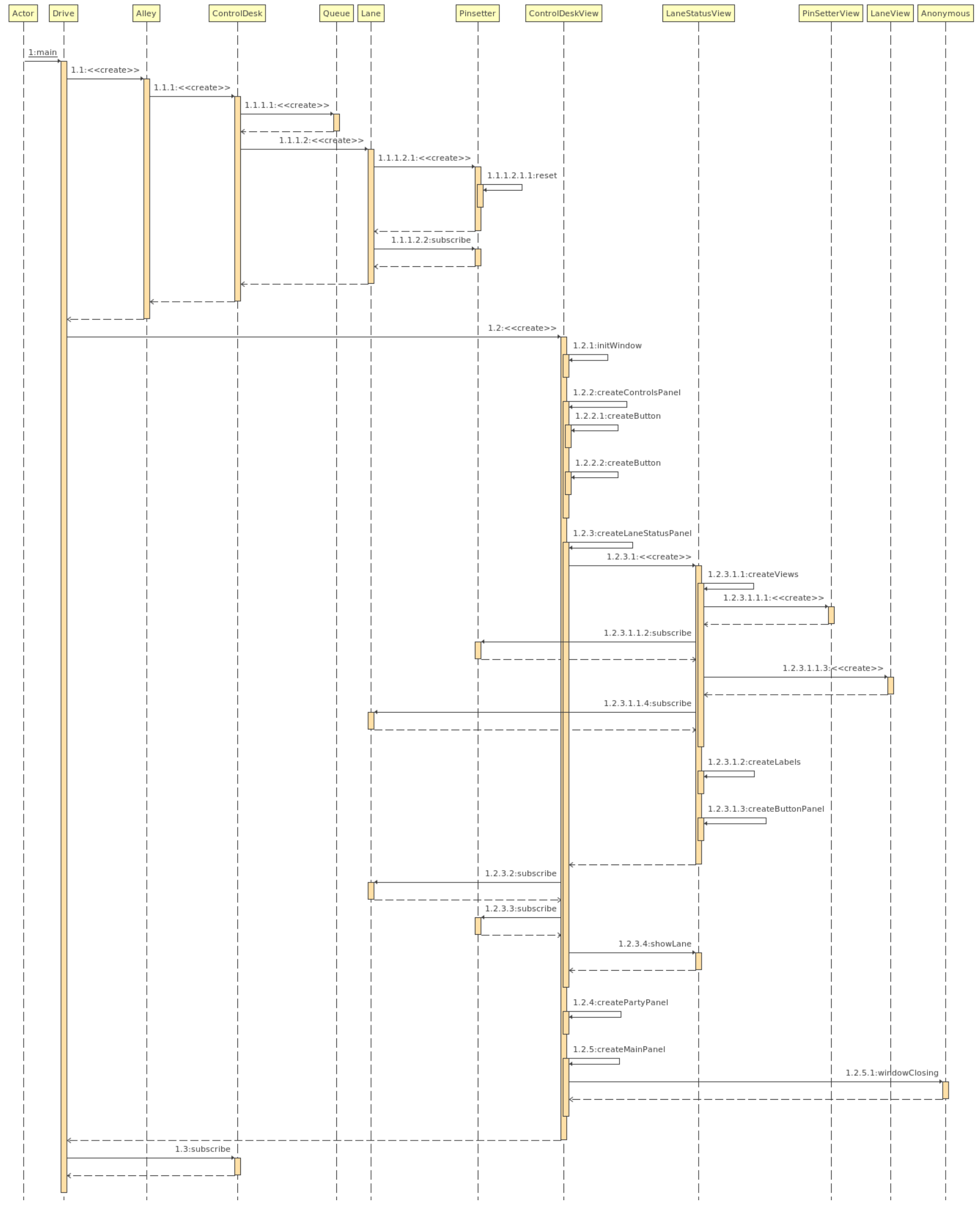


Lane run():

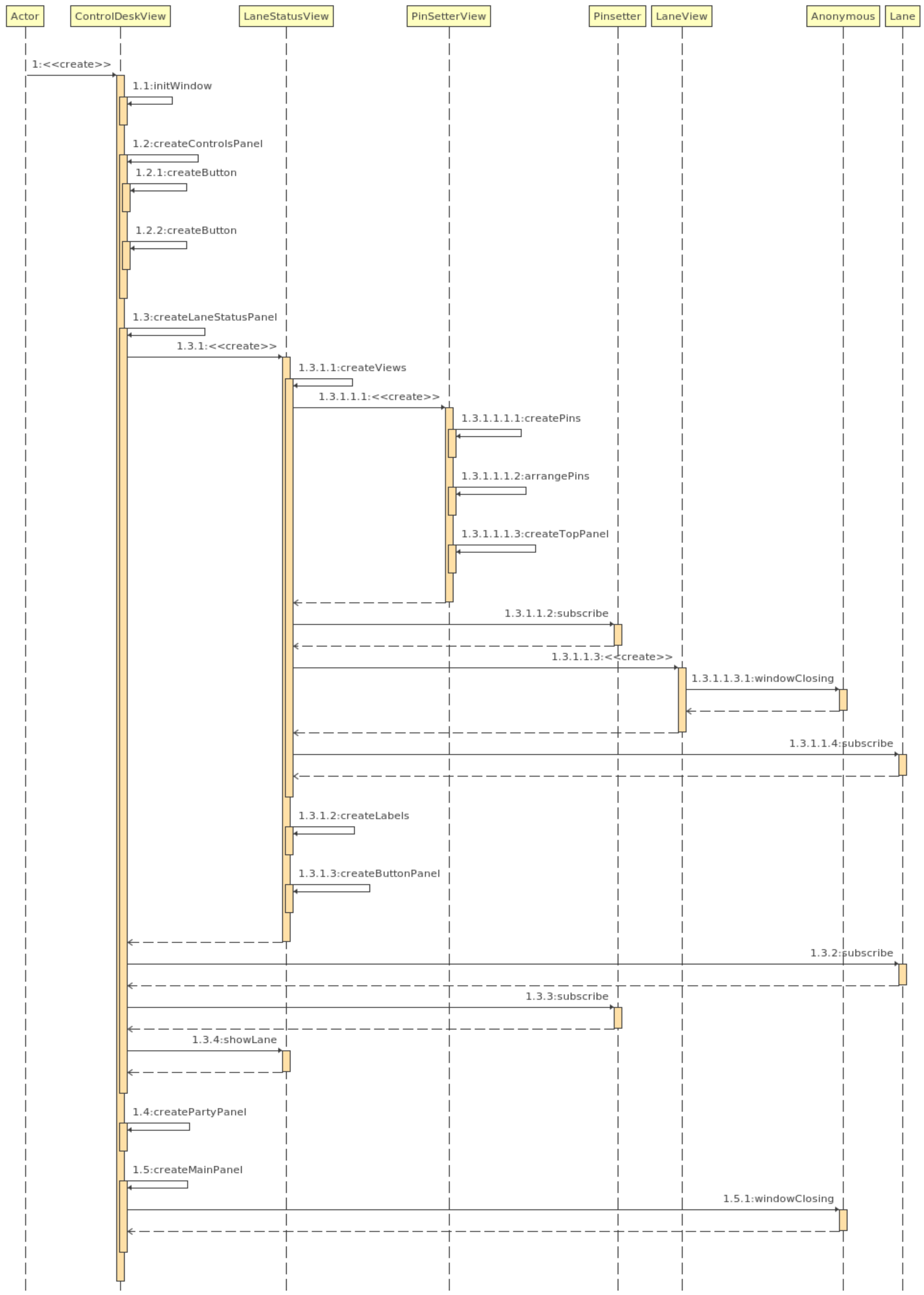

Actor
Lane
Pinsetter
PinsetterEvent
PinsetterObserver
ScoreHistoryFile
EndGamePrompt
EndGameReport
LaneEvent
LaneObserver
ScoreReport
Score
PrintableText
1:run
1.1:checkIfHalted
1.2:playGame
1.2.1:ballThrown
1.2.1.1:sendEvent
1.2.1.1.1:<<create>>
1.2.1.1.2:receivePinsetterEvent
1.2.2:addScore
1.2.3:reset
1.2.3.1:resetPins
1.2.3.2:sendEvent
1.2.3.2.1:<<create>>
1.2.3.2.2:receivePinsetterEvent
1.2.4:resetBowlerIterator
1.3:endGame
1.3.1:<<create>>
1.3.1.1:initWindow
1.3.1.2:createLabelPanel
1.3.1.3:createButtonPanel
1.3.1.3.1:createButton
1.3.1.3.2:createButton
1.3.1.4:createMainPanel
1.3.2:distroy
1.3.3:resetScores
1.3.4:resetBowlerIterator
1.3.5:<<create>>
1.3.5.1:initWindow
1.3.5.2:createPartyPanel
1.3.5.3:createButtonPanel
1.3.5.3.1:createButton
1.3.5.3.2:createButton
1.3.5.4:createMainPanel
1.3.6:lanePublish


1.3.6.1:<<create>>
1.3.7:publish
1.3.7.1:receiveLaneEvent
1.3.8:<<create>>
1.3.8.1:getScores
1.3.8.1.1:<<create>>
1.3.8.2:generateContent
1.3.9:sendEmail
1.3.9.1:sendln
1.3.9.2:sendln
1.3.9.3:sendln
1.3.9.4:sendln
1.3.9.5:sendln
1.3.9.6:sendln
1.3.9.7:sendln
1.3.9.8:sendln
1.3.9.9:sendln
1.3.9.10:sendln
1.3.9.11:sendln
1.3.10:sendPrintout
1.3.10.1:<<create>>

Drive main():


Actor
Drive
Alley
ControlDesk
Queue
Lane
Pinsetter
ControlDeskView
LaneStatusView
PinSetterView
LaneView
Anonymous
1:main
1.1:<<create>>
1.1.1:<<create>>
1.1.1.1:<<create>>
1.1.1.2:<<create>>
1.1.1.2.1:<<create>>
1.1.1.2.1.1:reset
1.1.1.2.2:subscribe
1.2:<<create>>
1.2.1:initWindow
1.2.2:createControlsPanel
1.2.2.1:createButton
1.2.2.2:createButton
1.2.3:createLaneStatusPanel
1.2.3.1:<<create>>
1.2.3.1.1:createViews
1.2.3.1.1.1:<<create>>
1.2.3.1.1.2:subscribe
1.2.3.1.1.3:<<create>>
1.2.3.1.1.4:subscribe
1.2.3.1.2:createLabels
1.2.3.1.3:createButtonPanel
1.2.3.2:subscribe
1.2.3.3:subscribe
1.2.3.4:showLane
1.2.4:createPartyPanel
1.2.5:createMainPanel
1.2.5.1:windowClosing
1.3:subscribe

ControlDeskView Constructor():



Responsibility of Classes:

S.No	Class	Responsibility
1.	Alley	It creates a control desk with the given number of lanes.
2.	Bowler	It contains the details of each bowler such as full name, nick name and email.
3.	BowlerFile	It retrieves the bowlers data from the database and creates bowler objects with that data and returns those bowler objects.
4.	Control Desk	It consists of a collection of lane objects and registers the bowlers. It creates a party from bowlers and stores them in the waiting queue. It assigns a free lane to the parties in the waiting queue.
5.	ControlDeskView	It is responsible for displaying the party queues, lane statuses and control center, and it is also the first view that is displayed after running the project.
6	Lane	It initializes a new lane object and creates a thread which assigns parties to the lane, records and calculates the score for each bowler, reset the scores, publishes events to the subscribers, pausing and unpausing the game.

7.	LaneView	It is responsible for displaying the frame of each lane separately containing status of each throw and scores of each bowler while playing the game.
8	LaneStatusView	It is responsible for displaying the status of each lane i.e. bowler name and pinsetter in the control desk while playing the game.
9.	NewPatronView	It displays the window containing the fields which are required while adding a new bowler to the bowlers database.
10.	Party	It is responsible for creating a party object from the given list of bowlers.
11.	AddPartyView	It displays the GUI used for creating the parties from the existing bowlers in the database, adding new bowlers to the database and finally adds the party in the party queue.
12.	Pinsetter	It represents the bowling pins and performs operations like resetting the pins, simulates ball throw and sends events to the subscribers.
13.	PinsetterView	It displays the current state of the pinsetter for each lane after the bowler finishes the throw in the control desk.
14.	Drive	It initializes the game and creates a bowling alley with a given number of lanes and

		displays the GUI to the users.
15.	EndGamePrompt	It pops-up at the end of the game asking the user whether to bowl another game or exit the game.
16.	EndGameReport	It displays a report for each party at the end of the game if requested.
17.	CalculateScore	It is responsible for calculating the score of the bowler assigned to a lane.
18.	ScoreReport	It generates the score reports of each bowler and emails the scores to the players if requested.

Narrative of the original design:

It was found from pre-refactored design that the project had all the classes and interfaces residing in the same source folder. No packaging of correlated modules were present. Although the application was working properly, from the readability point of view it was not easy to understand what was really happening. The observer pattern was implemented. Publishers such as Lane, ControlDesk, etc., were responsible to publish events via their corresponding Event classes which are passed to the Corresponding observers. Apart from the overall design, many classes had a lot of code smells such as bloaters, object-oriented abusers etc. The cohesion among classes was found to be on the safer side.

Weaknesses:

The weaknesses in the original design are explained using the code smells:

Type	Problem	Examples
Bloaters	**Long Methods:** Constructor or methods containing too many lines of code, which can be split into separate methods.	EndGameReport.java Lane.java Constructors in the view classes
	Large Class: A class containing many fields/methods/lines of code.	Lane.java
Dispensables	**Duplicate Code:** Two code fragments looking almost identical. It can be fixed using the Extract Method and call the new method in both places.	Constructors in the view classes use the same code for creating and displaying the UI elements
	Lazy class: A state in which the class is lazy with little behavior .i.e. class with only getter and setter methods and no significance.	LaneEvent.java Score.java Queue.java
	Dead code: A variable, parameter, field, method or class that is no longer used.	LanerEventInterface.java LaneServer.java
Couplers	**Feature Envy:** A method that accesses the data of another	Lane.java

	object more than its own data.	
God object	Objects that know too much and have too many responsibilities. It creates tight coupling and increases the challenges in the code maintainability.	ControlDeskView.java LaneStatusView.java

Strengths:

1. **Code readability**, which makes it easier to understand as the class/method names are relevant to the original functionality.
2. The views and its corresponding functionality has been separated among various classes which makes it **less complex**.
3. **High Cohesion** because the related functionality is kept together.
4. Even though the functionality is divided among various classes, there is **no tight coupling** among the classes.

Fidelity:

Fidelity refers to the degree to which a model or simulation reproduces the state and behavior of real world objects. In this project, a prototype of a bowling management system has been developed by following the given requirements. The original design includes all the functionality specified in the design document and the requirements are implemented in such a way that the quality of the software was also not compromised.

Use of Design Pattern:

The original design separates the views and the logic into separate classes without combining them together which makes it somewhat similar to

MVC pattern and it also uses the **Observer behavioral pattern** so that when an object changes its state, or when an event occurs, all of its dependents are notified and updated automatically.

Narrative of the refactored design:

The first step we took was to create a conceptual UML diagram of the application. We listed down the classes which belonged to the related categories. We put all the Plain Old Java Object (POJO) classes into the "model" package. These classes do not have any business logic written in them. We then identified the publishers of the application which were responsible for publishing events and packaged them into a "publisher" package. All the corresponding events which were used by those publishers were packaged into an "event" package. Hence, if a publisher wishes to publish an event, it will do so with the help of one of its Event classes. It is possible that more than one class may wish to listen to events published by a publisher. Hence, we packaged all the interfaces of the corresponding observers of the publishers into an "observer" package. In the application, most of the views are the observers and the action listeners. Those views are also responsible for displaying the GUI of the application. We decided to keep all such views in the "view" package. Apart from these classes, we found out that there are a few helper classes used by other classes which we identified and put into the "utility" package.

This packaging helped establish "separation of concern" into the application. If a new publisher needs to be integrated into the application, we can do so by creating a class in the "publisher" package. Then its corresponding event needs to be created into the event package. A new observer would be created, and the implementing class would listen to the events published by the new publisher. This precisely helps establish an "observer pattern" in the application.

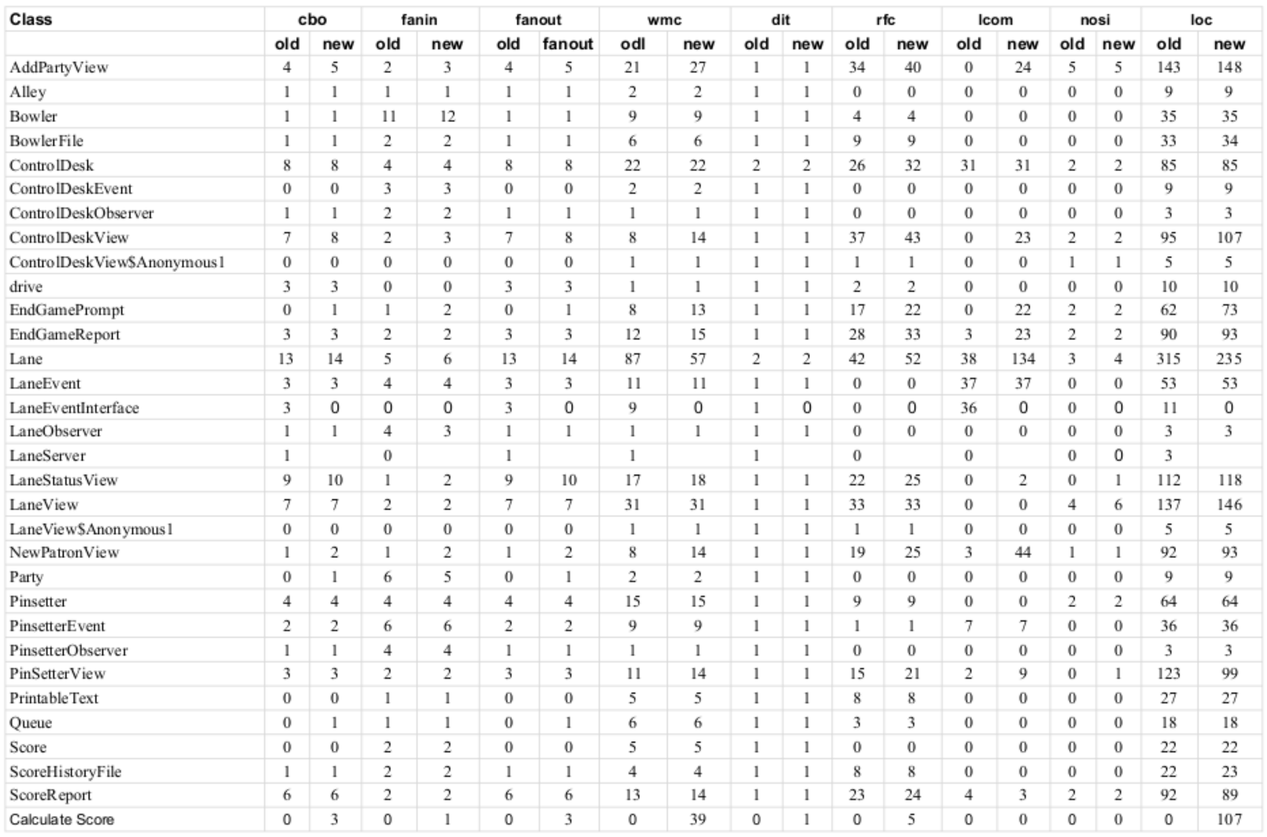
We could have avoided passing events to the observers and could have just passed the whole publisher object to the observer. But this would have violated the "Law of Demeter". Hence, we decided to preserve the original design of passing the event classes. We identified that "LaneServer" class to be redundant and not contributing to the application, hence, decided to remove it.

Metrics and Analysis:

From the point of view of metrics, we identified the classes with tight coupling and less cohesion. Then we picked those classes and tried to refactor them so that the refactored design gives loose coupling and cohesiveness to the system.

We found that our refactored design helped improve a few metrics. We were able to reduce the number of lines of code (LOC) for a few classes. However, for a few classes LOC increased to extract out the methods to increase readability of the code. Segregating the packages into different packages may have increased the overall complexity of the system, but priority was given to the separation of concern.

We also found significant improvement for few of the other metrics as shown in the table:



Class	cbo		fanin		fanout		wmc		dit		rfc		lcom		nosi		loc	
	old	new	old	new	old	fanout	odl	new	old	new	old	new	old	new	old	new	old	new
AddPartyView	4	5	2	3	4	5	21	27	1	1	34	40	0	24	5	5	143	148
Alley	1	1	1	1	1	1	2	2	1	1	0	0	0	0	0	0	9	9
Bowler	1	1	11	12	1	1	9	9	1	1	4	4	0	0	0	0	35	35
BowlerFile	1	1	2	2	1	1	6	6	1	1	9	9	0	0	0	0	33	34
ControlDesk	8	8	4	4	8	8	22	22	2	2	26	32	31	31	2	2	85	85
ControlDeskEvent	0	0	3	3	0	0	2	2	1	1	0	0	0	0	0	0	9	9
ControlDeskObserver	1	1	2	2	1	1	1	1	1	1	0	0	0	0	0	0	3	3
ControlDeskView	7	8	2	3	7	8	8	14	1	1	37	43	0	23	2	2	95	107
ControlDeskView$Anonymous1	0	0	0	0	0	0	1	1	1	1	1	1	0	0	1	1	5	5
drive	3	3	0	0	3	3	1	1	1	1	2	2	0	0	0	0	10	10
EndGamePrompt	0	1	1	2	0	1	8	13	1	1	17	22	0	22	2	2	62	73
EndGameReport	3	3	2	2	3	3	12	15	1	1	28	33	3	23	2	2	90	93
Lane	13	14	5	6	13	14	87	57	2	2	42	52	38	134	3	4	315	235
LaneEvent	3	3	4	4	3	3	11	11	1	1	0	0	37	37	0	0	53	53
LaneEventInterface	3	0	0	0	3	0	9	0	1	0	0	0	36	0	0	0	11	0
LaneObserver	1	1	4	3	1	1	1	1	1	1	0	0	0	0	0	0	3	3
LaneServer	1		0		1		1		1		0		0		0	0	3	
LaneStatusView	9	10	1	2	9	10	17	18	1	1	22	25	0	2	0	1	112	118
LaneView	7	7	2	2	7	7	31	31	1	1	33	33	0	0	4	6	137	146
LaneView$Anonymous1	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	5	5
NewPatronView	1	2	1	2	1	2	8	14	1	1	19	25	3	44	1	1	92	93
Party	0	1	6	5	0	1	2	2	1	1	0	0	0	0	0	0	9	9
Pinsetter	4	4	4	4	4	4	15	15	1	1	9	9	0	0	2	2	64	64
PinsetterEvent	2	2	6	6	2	2	9	9	1	1	1	1	7	7	0	0	36	36
PinsetterObserver	1	1	4	4	1	1	1	1	1	1	0	0	0	0	0	0	3	3
PinSetterView	3	3	2	2	3	3	11	14	1	1	15	21	2	9	0	1	123	99
PrintableText	0	0	1	1	0	0	5	5	1	1	8	8	0	0	0	0	27	27
Queue	0	1	1	1	0	1	6	6	1	1	3	3	0	0	0	0	18	18
Score	0	0	2	2	0	0	5	5	1	1	0	0	0	0	0	0	22	22
ScoreHistoryFile	1	1	2	2	1	1	4	4	1	1	8	8	0	0	0	0	22	23
ScoreReport	6	6	2	2	6	6	13	14	1	1	23	24	4	3	2	2	92	89
Calculate Score	0	3	0	1	0	3	0	39	0	1	0	5	0	0	0	0	0	107

cbo: Coupling Between Object Classes

fanin: In-degree of corresponding graph vertex of the class

fanout: Out-degree of corresponding graph vertex of the class

wmc: Weighted Method Count

dit: Depth of Inheritance Tree

rfc: Response For a Class

lcom: Lack of Cohesion Methods

nosf: Number of static fields

loc: Lines of code

Conclusion:

Removing code smells, repackaging the application, removing redundancy and redesigning the UML class diagram helped improve the overall design and readability of the application. Establishing the separation of concern resulted in making the application extensible for adding new functionalities. Generating metrics helped analyze the existing system and acted as a guide towards improvement of the design.