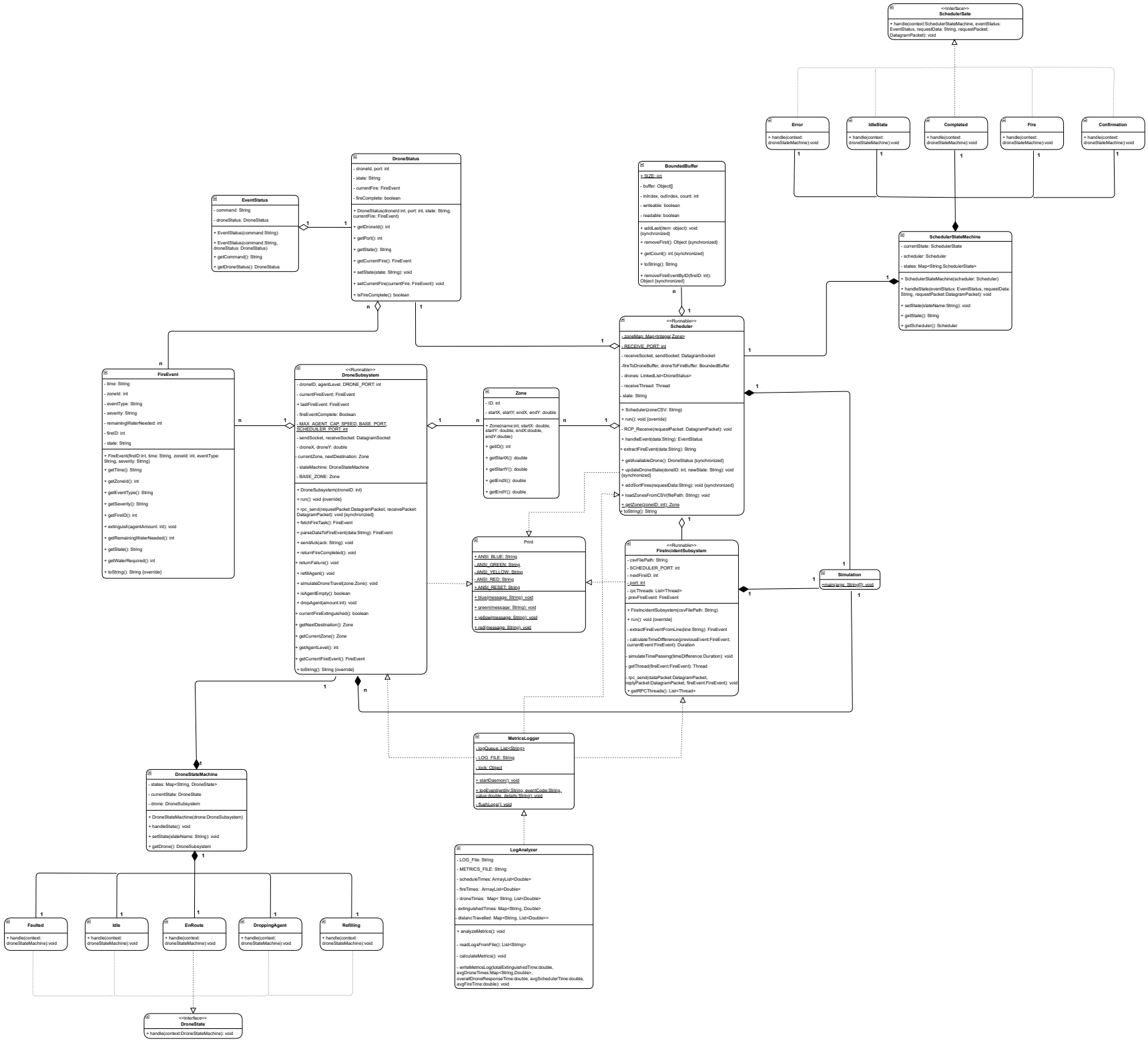
<<interface>>
SchedulerSate
+ handle(context:SchedulerStateMachine, eventStatus: EventStatus, requestData: String, requestPacket: DatagramPacket): void
Error
+ handle(context: droneStateMachine):void
IdleState
+ handle(context: droneStateMachine):void
Completed
+ handle(context: droneStateMachine):void
Fire
+ handle(context: droneStateMachine):void
Confirmation
+ handle(context: droneStateMachine):void
SchedulerStateMachine
- currentState: SchedulerState
- scheduler: Scheduler
- states: Map<String,SchedulerState>
+ SchedulerStateMachine(scheduler: Scheduler)
+ handleState(eventStatus: EventStatus, requestData: String, requestPacket:DatagramPacket): void
+ setState(stateName:String): void
+ getState(): String
+ getScheduler(): Scheduler
DroneStatus
- droneId, port: int
- state: String
- currentFire: FireEvent
- fireComplete: boolean
+ DroneStatus(droneId:int, port: int, state: String, currentFire: FireEvent)
+ getDroneId(): int
+ getPort(): int
+ getState(): String
+ getCurrentFire(): FireEvent
+ setState(state: String): void
+ setCurrentFire(currentFire: FireEvent): void
+ isFireComplete(): boolean
EventStatus
- command: String
- droneStatus: DroneStatus
+ EventStatus(command:String)
+ EventStatus(command:String, droneStatus: DroneStatus)
+ getCommand(): String
+ getDroneStatus(): DroneStatus
BoundedBuffer
+ SIZE: int
- buffer: Object[]
- inIndex, outIndex, count: int
- writeable: boolean
- readable: boolean
+ addLast(item: object): void {synchronized}
+ removeFirst(): Object {synchronized}
+ getCount(): int {synchronized}
+ toString(): String
+ removeFireEventByID(fireID: int): Object {synchronized}
<<Runnable>>
Scheduler
- zoneMap: Map<Integer Zone>
- RECEIVE_PORT: int
- receiveSocket, sendSocket: DatagramSocket
-fireToDroneBuffer, droneToFireBuffer: BoundedBuffer
- drones: LinkedList<DroneStatus>
- receiveThread: Thread
- state: String
+ Scheduler(zoneCSV: String)
+ run(): void {override}
- RCP_Receive(requestPacket: DatagramPacket): void
+ handleEvent(data:String): EventStatus
+ extractFireEvent(data:String): String
+ getAvailableDrone(): DroneStatus {synchronized}
+ updateDroneState(droneID: int, newState: String): void {synchronized}
+ addSortFires(requestData:String): void {synchronized}
+ loadZonesFromCSV(filePath: String): void
+ getZone(zoneID: int): Zone
+ toString(): String
FireEvent
- time: String
- zoneId: int
- eventType: String
- severity: String
- remainingWaterNeeded: int
- fireID: int
- state: String
+ FireEvent(fireID:int, time: String, zoneId: int, eventType: String, severity: String)
+ getTime(): String
+ getZoneId(): int
+ getEventType(): String
+ getSeverity(): String
+ getFireID(): int
+ extinguish(agentAmount: int): void
+ getRemainingWaterNeeded(): int
+ getState(): String
+ getWaterRequired(): int
+ toString(): String {override}
<<Runnable>>
DroneSubsystem
- droneID, agentLevel, DRONE_PORT: int
- currentFireEvent: FireEvent
+ lastFireEvent: FireEvent
- fireEventComplete: Boolean
- MAX_AGENT_CAP,SPEED, BASE_PORT, SCHEDULER_PORT: int
- sendSocket, receiveSocket: DatagramSocket
- droneX, droneY: double
- currentZone, nextDestination: Zone
- stateMachine: DroneStateMachine
- BASE_ZONE: Zone
+ DroneSubsystem(droneID: int)
+ run(): void {override}
+ rpc_send(requestPacket:DatagramPacket, receivePacket: DatagramPacket): void {synchronized}
+ fetchFireTask(): FireEvent
+ parseDataToFireEvent(data:String): FireEvent
+ sendAck(ack: String): void
+ returnFireCompleted(): void
+ returnFailure(): void
+ refillAgent(): void
+ simulateDroneTravel(zone:Zone): void
+ isAgentEmpty(): boolean
+ dropAgent(amount:int): void
+ currentFireExtinguished(): boolean
+ getNextDestination(): Zone
+ getCurrentZone(): Zone
+ getAgentLevel(): int
+ getCurrentFireEvent(): FireEvent
+ toString(): String {override}
Zone
- ID: int
- startX, startY, endX, endY: double
+ Zone(name:int, startX: double, startY: double, endX:double, endY:double)
+ getID(): int
+ getStartX(): double
+ getStartY(): double
+ getEndX(): double
+ getEndY(): double
Print
+ ANSI_BLUE: String
-ANSI_GREEN: String
-ANSI_YELLOW: String
- ANSI_RED: String
+ ANSI_RESET: String
+ blue(message: String): void
+ green(message: String): void
+ yellow(message: String): void
+ red(message: String): void
<<Runnable>>
FireIncidentSubsystem
- csvFilePath: String
- SCHEDULER_PORT: int
- nextFireID: int
- port: int
- rpcThreads: List<Thread>
- prevFireEvent: FireEvent
+ FireIncidentSubsystem(csvFilePath: String)
+ run(): void {override}
- extractFireEvent(FromLine(line:String): FireEvent
- calculateTimeDifference(previousEvent:FireEvent, currentEvent:FireEvent): Duration
- simulateTimePassing(timeDifference:Duration): void
- getThread(fireEvent:FireEvent): Thread
- rpc_send(dataPacket:DatagramPacket, replyPacket:DatagramPacket, fireEvent:FireEvent): void
+ getRPCThreads(): List<Thread>
Simulation
+main(args: String[]): void
MetricsLogger
- logQueue: List<String>
- LOG_FILE: String
- lock: Object
+ startDaemon(): void
+ logEvent(entity:String, eventCode:String, value:double, details:String): void
-flushLogs(): void
LogAnalyzer
- LOG_File: String
- METRICS_FILE: String
- scheduleTimes: ArrayList<Double>
- fireTimes: ArrayList<Double>
- droneTimes: Map< String, List<Double>
- extinguishedTimes: Map<String, Double>
- distancTravelled: Map<String, List<Double>>
+ analyzeMetrics(): void
- readLogsFromFile(): List<String>
- calculateMetrics(): void
- writeMetricsLog(totalExtinguishedTime:double, avgDroneTimes:Map<String,Double>, overallDroneResponseTime:double, avgSchedulerTime:double, avgFireTime:double): void
DroneStateMachine
- states: Map<String, DroneState>
- currentState: DroneState
- drone: DroneSubsystem
+ DroneStateMachine(drone:DroneSubsystem)
+ handleState(): void
+ setState(stateName: String): void
+ getDrone(): DroneSubsystem
Faulted
+ handle(context: droneStateMachine):void
Idle
+ handle(context: droneStateMachine):void
EnRoute
+ handle(context: droneStateMachine):void
DroppingAgent
+ handle(context: droneStateMachine):void
Refilling
+ handle(context: droneStateMachine):void
<<interface>>
DroneState
+ handle(context:DroneStateMachine): void