***Design Choices and Object-Oriented Decomposition:**

1. Elevator Class:

- Fields:
  - upPassengers and downPassengers: PriorityQueues to hold passengers traveling in the up and down directions, respectively.
  - currentFloor: Integer representing the current floor of the elevator.
  - maxFloor: Maximum number of floors in the building.
  - capacity: Maximum number of passengers the elevator can hold.
  - up: Boolean indicating the current direction of the elevator.
  - upDestination and downDestination: PriorityQueues to store destination floors for passengers traveling up and down.
- Methods:
  - unload(int currentFloor, int currentTick, SimulationResult result): if the elevator arrives at the current floor, unload passengers at the current floor and update the result.
  - unloadPassengers(PriorityQueue<Passenger> passengers, int currentFloor, int currentTick, SimulationResult result): Unloads passengers from a specific queue at the current floor.
  - removeCurrentFloorFromDestination(PriorityQueue<Integer> destinations, int currentFloor): Removes the current floor from the destination queues.
  - isAvailableToLoad(int floor, boolean upRequest): Checks if the elevator is available to load passengers based on direction and capacity.
  - load(Passenger passenger): Loads a passenger into the elevator, updating direction and destination queues.
  - getCurrentFloor(): Returns the current floor of the elevator.
  - getElevatorDirection(): Returns the direction of the elevator.
  - isAvailableToRequest(int floor, boolean upRequest): Checks if the elevator is available to fulfill a passenger request.
  - passengerRequests(Passenger passenger): Processes a passenger request and updates elevator state.
  - travel(): Moves the elevator to the next floor based on loaded passengers and pending requests.
  - isRunning(): Checks if the elevator is in operation.
  - findNextDestination(): Determines the next floor based on passenger requests.
  - toString(): Generates a string representation of the elevator.

2. Floor Class:

- Fields:
  - up, down: Queues to manage passengers going up and down, respectively.
  - floorNumber: The floor number.
  - possibleDestinations: List of possible destination floors.

- Methods:
  - getUpQueue(), getDownQueue(): Returns the queues for passengers going up and down, respectively.
  - getFloorNumber(): Returns the floor number.
  - generatePassenger(): Generates a new random passenger with a destination floor.
  - load(Passenger passenger): Loads a passenger into the appropriate queue based on the direction.
  - possibleDestinationFloors(int maxFloor): Populates the list of possible destination floors.
  - toString(): Generates a string representation of the floor.

3. Passenger Class:

- Fields:
  - startFloor, destinationFloor: The starting and destination floors of the passenger.
  - startTime, endTime: The time the passenger arrives at a floor and exits the elevator.
  - up: Boolean indicating if the passenger is going up.
  - uuid: Unique identifier for the passenger.
- Methods:
  - getStartFloor(), getDestinationFloor(): Returns the starting and destination floors.
  - getStartTime(), getEndTime(): Returns the start and end times of the passenger.
  - setEndTime(int endTime), setStartTime(int startTime): Sets the end and start times of the passenger.
  - goingUp(): Checks if the passenger is going up.
  - getConveyanceTime(): the overall duration from the passenger's arrival until reaching the destination floor.
  - compareTo(Passenger o): Compares passengers based on destination floor.
  - toString(): Generates a string representation of the passenger.

4. PropertyReader Class:

- Methods:
  - Reads and validates properties from a file, sets default values if necessary, and initializes and runs the simulation.

5. ElevatorSimulation Class:

- Fields:
  - elevators, floors: Lists of elevators and floors.
  - tick, duration: Simulation time and duration.
  - passengerProbability: Probability of a new passenger arriving in each tick.
  - result: SimulationResult object to collect and calculate statistics.
- Methods:

- runSimulation(): Runs the elevator simulation based on ticks, handling passenger generation, loading, unloading, and elevator movement.

6. SimulationResult Class:
- Fields:
    - Tracks total passengers, total conveyance time, longest and shortest times.
- Methods:
    - addPassenger(int conveyanceTime): Updates statistics when a passenger completes their journey.
    - calculateStatistics(): Calculates and prints average, longest, and shortest conveyance times.

***Data structure choices:**
1. Elevator Class
    - upPassengers, downPassengers (PriorityQueue): PriorityQueues are used to maintain passengers in ascending order of their destination floors. This allows for efficient retrieval of the next passenger to unload, prioritizing the one with the closest destination floor. The use of PriorityQueue ensures that passengers are processed in a way that minimizes travel distance.
    - upDestination, downDestination (PriorityQueue): Similar to passenger queues, destination floors are stored in PriorityQueues. This choice ensures that the next destination floor is easily accessible, supporting the logic for the elevator's movement. The ordering facilitates quick identification of the next floor to visit.
2. Floor Class
    - up, down (Queue): Queues are chosen to represent passengers waiting on a floor. The use of Queue aligns with the First-Come-First-Serve principle, ensuring that passengers are loaded into elevators in the order they arrive.
    - possibleDestinations (List): A List is used to store possible destination floors for a specific floor. This choice allows for straightforward iteration through the list when generating random destination floors for passengers. The ordering is not critical in this context.
3. ElevatorSimulation Class
    - elevators, floors (List): Lists are chosen to store elevators and floors. The use of lists provides flexibility in dynamically managing these entities during the simulation. Lists allow for easy iteration of elements.

***Instructions on how to get your implementation compiled and running correctly**
- Loading Properties from the File:
    - If providing a command-line argument:

- Ensure that the path to the property file is provided as a command-line argument when running the program.
- Example: This is how I executed it on my laptop through Eclipse: src\\for_elevator\\myFile.properties. Please customize it according to your setup.