Mia Harang

Doyle Chism

Sam Gumm

Kenny Jia Hui Leong

Mani Raj Rejinthala

Computer Science 362

Project topic: Fashion

Iteration 02

Departments [Use Cases]

HQ	Design	Manufacturing	Marketing
collateGlobalReport	designProduct Create MarketingDesign approveDesign transferDesign	createProduct ReceiveDesign transferProudct	advertiseDesign advertiseEvent distributeCatalog

Inventory	Modeling	Treasury	HR
updateInventory -addProducts -deleteProducts RegisterProduct RegisterStorage	haveEvent scheduleFittings trainModels	processPayroll adjustSalary	hireCandidate handleEmployee handleCandidate

Committee	Sales	Security	
createCatalog	ProcessNewOrder HandlingOrderReturn RegisterRetailer	assignSecurityPersonnel manageSecurityAudits	

Marketing Department

[Mia Harang]

Use Case #1:

advertiseEvent

Actor(s): [Head of Marketing], Team Managers, Modeling Department

Preconditions:

1. The modeling department has an event they want to advertise
2. All venues, models, celebrities, and brand deals for the event have been booked
3. The event has been logged into the storage system
4. Some models are available to advertise the event

Main Success Scenario:

1. The Modeling Department has communicated to the Head of Marketing that there is an event that they want to advertise
2. The Modeling Department sends over the event details
3. Depending on the event type, celebrities, and brand deals, the Head of Marketing will meet with the Team Managers [Video, Social Media, Editing, Distribution] to determine what kind of advertisement they want to do (1 or more).
    a. Billboard/Poster Add: photo of model/celebrity wearing branded/non-branded garments on billboard/poster [Editing & Distribution]
    b. Magazine Add: [Specific for Photoshoots] photos gathered from photoshoot, added into the branded magazine (i.e. Vogue, etc) [Editing & Distribution]
    c. Social Media Add: photo of model/celebrity wearing branded/non-branded garments on a social media platform, as well as information about the event. [Editing & Social Media]
    d. Commercial Add: video add of model/celebrity wearing the specific branded garment, with audio and visual descriptions of the event. [Editing & Video]
4. Once one or more add types have been chosen for the event, each team that is required to create the add type works together to schedule a photo shoot/video event with the modeling department.
5. After the event takes place, the editing team will take the images/videos gathered from the event and edit them into an ad format, sending them to either the video, social media, or distribution teams.

6. The Video, Social Media, or Distribution Teams will distribute the ads to their respective ad platforms [i.e. social media platform, billboard, poster in the mall, branded magazine, etc.]
7. The HOM will store the advertisement in the file system.

Alternative Flows:

- No model is available for an advertisement:
    - The head will talk to modeling to see if any schedules can be re-arranged. If not, a temporary model will be hired for the advertisement
- Event gets canceled
    - If advertisements have been distributed, create a new ad to announce cancelation. Otherwise, stop the advertisement process as soon as possible, and remove data from the file system.
- A brand deal/celebrity backs out
    - If advertisements have been distributed, announce their departure immediately and update the advertisement accordingly. Otherwise, halt all advertisement creations and update them to match the current information

Use Case #2:

advertiseDesign

Actor(s): [Head of Marketing], Team Managers, Design Department

Preconditions:

1. The design department has an event they want to advertise
2. The design has been finalized and approved by HR
3. The design is logged into the system
4. Some models are available to advertise the design

Main Success Scenario:

1. The design department communicates with the marketing department and informs them that they have a new design to be advertised.
2. The design department sends the design details and information (if applicable) on how they want it advertised and displayed on the model.
3. Depending on the specifications given by the design department, the Head of Marketing will meet with the Team Managers [Video, Social Media, Editing, Distribution] to determine what kind of advertisement they want to do (1 or more).
    a. Ad types and teams dedicated to them are the same as advertiseEvent

4. Once an ad type has been chosen, the teams assigned to the project schedule a fitting with a model from the modeling department.
    a. The modeling department will request a custom product for the model from the manufacturing department
5. After the fitting takes place, the teams assigned to the advert, finish creating said advert
6. The Video, Social Media, or Distribution Teams will distribute the ads to their respective ad platforms [i.e. social media platform, billboard, poster in the mall, branded magazine, etc.]
7. The HOM will store the advertisement in the file system.

Alternative Flows:

- No model is available for an advertisement:
    - The head will talk to modeling to see if any schedules can be re-arranged. If not, a temporary model will be hired for the advertisement
- Design gets recalled
    - If advertisements have been distributed, create a new ad to announce cancelation. Otherwise, stop the advertisement process as soon as possible. Remove any records of it.

HR Department

[Sam Gumm]

Use Case #1:

manageEmployeeRecords

Actor(s):

- HR Assistant
- HR Manager
- Store Manager
- IT Specialist

Preconditions:

- Employee data is consistently formatted to the expectations
- Mainframe system works
- Security measures have been implemented

Main Success Scenario:

- Store Managers Collect Employee Information
- Employee information, including new hires, terminations, promotions, and updates, is compiled using standardized forms.
- Forms are sent to headquarters for processing.
- HR Assistant Inputs Employee Data
- The system checks for inconsistencies, missing fields, or format errors and flags issues.
- The HR Assistant reviews errors, works with Store Managers to resolve them, and ensures data accuracy.
- Once validated, the system updates the centralized employee database with new or modified records.
- Employee records, including paper trails or scanned documents, are securely stored in the system for reference.

Alternative Flows:

- 4a. Data Format Issues Detected
    - The system flags improperly formatted data.
    - HR Assistant works with Store Managers to correct and resubmit data.
- 5a. Incomplete Employee Information
    - System flags missing information.
    - HR Assistant contacts Store Managers to obtain missing details.

Use Case #2:

manageCandidateRecords

Actor(s):

- HR Manager
- Hiring Manager
- IT Specialist
- Candidates

Preconditions:

- The job description, qualifications, and responsibilities are defined and approved
- Hiring budget is established
- Candidate management system is operational

Main Success Scenario:

1. HR Manager creates and posts job listing, which includes details about qualifications, responsibilities, and salary range.
2. Candidates submit applications and HR adds them to Applied folder
3. Applications are checked for completeness and adherence to format requirements.
4. Any flagged issues are communicated to candidates for resolution.
5. HR Manager moves any Candidates moving forward in the hiring process to PENDING, which indicates an interview needs to be conducted.
6. After the interview, the HR Manager moves the candidate to the APPROVED folder to indicate they are ready for the next stage in the hiring process.
7. The Hiring Manager looks through the APPROVED folder to decide which candidates are to move on to the HIRING folder.
8. The HR Assistant manually transfers the information of candidates who accepted their assigned position into the system as a newly hired employee.

Alternative Flows:

- 3a. Incomplete Application
    - System notifies candidates to provide missing details.
- 5a/6a. Rejected candidates are sent to the REJECTED folder to be deleted.
- 7a. No Suitable Candidates
    - HR Manager reviews the pool or reposts the vacancy.
- 8a. Candidate Declines Offer
    - HR Manager offers the position to the next best candidate.

Design Department

[Doyle Chism]

Use Case #1:

ManageDesignCommunication

Primary Actor: Head of Design
Supporting Actor: Head of Marketing, Head of Manufacturing**,** Head of Modelling
Preconditions: The Design Team has all the required materials to produce a design. The Design Team also has the ability to send their designs to other departments after creating the design.
Main Success Scenario:

1. The Design Team creates a list of sketches to be reviewed by the Head of Design.
2. The Head of Design looks through all the sketches and selects which one they seem best fit.
3. The Head of Design sets a Final Design for Manufacturing based on the sketches.
4. The Head of Design then sets the marketing specifications based on the Final Design that was selected for Manufacturing.
5. After the Marketing design is set, the design team beings creating the specifications needed for Marketing the product.
6. The Head of Design receives a custom design from the modelling department.
7. The Head of Design and the Desing Team customizes and creates the custom design that the modelling team needs.
8. After the Head of Design sets the Final Design and Custom Design for Manufacturing then they will send the specifications to the Manufacturing Department to create the product.
9. After the Head of Design sets the Marketing Design based on the Final Design they will send the specifications to the Marketing Department to begin marketing the product.

Alternative Flows:

1-2A: The Head of Design either selects a sketch or does not verify the sketch to become a final design.

2B. Only one Final Design will be selected for Manufacturing at a time.

3-4A: The Head of Design either verifies the sketch or makes changes to the current sketch to meet the requirements they see fit.

5A: The marketing design is set based on the Final Design for Manufacturing

5B: The Marketing Design is set based on targetAudience, price, seasonType, and description. Given a brief explanation on why these are marketable items.

7A: The head of Design begins setting the specifications needed for the custom design.

7B: The specifications that are different from creating a normal product is exact in. and width measurements to fit the model body type.

8A: The head of design will receive the manufacturing product and either change modifications of the final design or tell them to make the product again.

8B: They will repeat steps 3-5A until they achieve the final product they are looking for.

9A: The Marketing Department does not like the marketing idea, so they send it back to be modified.

9B: Repeat steps 5A-5B to fit what the marketing department desires.

Manufacturing Department

[Doyle Chism]

Use Case #2:

ManageProduction

Actor(s): Head of Manufacturing

Supporting Actor: Manufacturing Manager, Head of Inventory. Head of Modelling, Head of Design

Preconditions: The Design Team has approved the created design t so it can begin being produced by the manufacturer. The manufacturer has all the necessary raw materials needed for production. The manufacturing has the ability to distribute items to the according departments.

Main Success Scenario:

1. The Manufacturing Department receives a Final Design or a custom Design from the Design Department.
2. The Head of Manufacturing receives the design and evaluates the specifications.
3. The Head of Manufacturing alerts the Manufacturing Manager to collect the raw materials for the product.
4. The Manufacturing manager receives the specifications and retrieves the materials needed for the product.
5. The Head of Manufacturign verifies the raw materials retrieved by the manufacturing manager.
6. The Manufacturing manager begins creating the product with the raw materials.
7. The product is created and collected by the manufacturing manager and is given

to the Head of Manufacturing for processing.

8. The Head of Manufacturing verifies if the product is up to quality standards for the Final Design or Custom Design.
9. The Head of Manufacturing sends the Custom Design to the Modelling Department.
10. The product is created it is stored in Inventory Department.

Alternative Flows:

5A: The raw materials are not the correct materials for the design so the manager has to retrieve the correct type/amount.

6A: The machine is not on, so it has to be turned on to begin production.

8A: The product does not meet quality standards to it has to be made again repeating steps 4-7.

9A: The model does not fit in the product created, so it has to be remade and sent back to the manufacturer; repeating steps 4-7.

9B: The product is not the desired style or color that the modelling team wanted, so it is sent back to the manufacturer, repeating steps 4-7.

Security Department

[Kenny Jia Hui Leong]

Use Case: assignSecurityPersonnel
Level:
Primary Actor: Head of Security (HoS)
Stakeholders & Interests:

- HoS: Wants fast and accurate processing of security personnel task assignments, wants no mistake in task assignments, can result in disruptions during events or product theft. This leads to reputational and financial losses.
- HoDepartments: Wants to make security personnel requests and receive them. Requirements can be changed depending on size of event or importance of supply/product.
- CEO/Headquarters: Wants security personnel to be at the right places at the right times to prevent operation disruptions or product damages.
- Security personnel: Wants to receive fast and accurate schedules of security task assignments.

Preconditions: security personnel data and security requests are complete and up-to-date

Main success scenario:

1. Departments submit new security requests and the system records it.
2. HoS enters updated security personnel information and retrieves existing lists from the system.
3. System records new personnel information if applicable.
4. HoS receives and views details about upcoming events or security requests from departments in the system, e.g. type of event, priority level, number of personnel required, etc.
5. HoS goes through security requests and assigns security personnel accordingly based on priority and needs.
6. System records data and presents security personnel schedules with complete information, e.g. assigned personnel, time, location, tasks, special requirements, and etc.
7. HoS repeats 5-6 until done.
8. HoS verifies assignment schedules and confirms in system.
9. System stores assignment schedules to repository.
10. System sends assignment schedules to departments.
11. System sends assignment schedules to security personnel.
12. HoS closes system.

Extensions:
2a. An emergency event is issued

1. System flags and presents emergency for specific event.
2. HoS performs immediate schedule override.

3. System presents all available or assigned security personnel.
4. HoS reschedules available or assigned security personnel to specific emergency event.
5. System receives new data and updates security assignment schedules accordingly.
6. System sends changed assignment schedules to security personnel.
7. System sends changed assignment schedules to departments.

5a. Insufficient security personnel for assignment
1. System flags and presents status on insufficient security personnel for assignment.
2. HoS overrides insufficient status and continues with personnel assignment with less security.
   2a. HoS chooses to reassign security personnel from schedules of lesser priority.
      1. System receives new request and updates both new and changed assignment schedules.
3. System receives request and updates schedule with overridden requirement.
4. HoS continues updating system with schedule data.

5b. Receives existing security request with changed priority level or requirements.
1. System flags and presents specific request with changed priority level or requirements.
2. HoS views flagged security request and adjusts new security requirements accordingly based on new priority or requirements.
3. System receives new information and updates changed security schedules.
4. HoS continues updating system with schedule data.

Use Case: manageSecurityAudits
Level:
Primary Actor: Head of Security (HoS)
Stakeholders & Interests:
- HoS: Wants fast and accurate processing of security audit team task assignments, wants no task assignment scheduling errors, can result in late audits on important company security infrastructure and systems. Leading to significant financial losses and reputational damage to the company.
- HoDepartments: Wants regular security audits to ensure all security infrastructure and systems are up-to-date to protect department operations from disruptions.
- CEO/Headquarters: Wants all security infrastructure and systems to be up-to-date to prevent any operational disruptions. Wants to prevent financial losses from operational disruptions and reputational losses from data breaches.
- Audit team: Wants to receive fast and accurate schedules of auditing tasks.
- Security team: Wants to receive information for vulnerabilities from cases of failed audits.

Preconditions: security auditing team data is complete and up-to-date

Main success scenario:
1. Security audit team submits completed security audit reports and the system records it.
2. HoS enters updated list of security audit personnel.

3. System records new audit personnel if applicable and displays all audit personnel data.
4. HoS receives and views completed or pending security audits reported for different departments from the system, e.g. audit report status, audits soon to be conducted, failed audits that require immediate attention, etc.
5. HoS goes through scheduled security audits for various departments and assigns audit personnel accordingly based on type of audit, deadline, or priority level.
6. System records data and presents security audit personnel schedules with complete information, e.g. assigned personnel, department, type of audit, specific instructions, and etc.
7. HoS repeats 5-6 until done.
8. HoS verifies auditing schedules and confirms in system.
9. System stores auditing schedules to repository.
10. System sends auditing schedules to department to notify them of pending audit.
11. System sends auditing schedules to security audit personnel.
12. HoS closes system.

Extensions:

2a. Audit failure that indicates critical vulnerability

1. System flags and presents vulnerability for specific department as gathered from failed audit report.
2. HoS creates an immediate emergency event detailing vulnerabilities after reviewing failed audit report.
3. System records the created emergency event and sends it to the Security team.
4. HoS reassigns and reschedules audit personnel who authored the failed report to assist the Security team.
5. System sends pending audit notification to specific department.
6. System receives new data and updates audit personnel schedule accordingly.
7. System sends updated security audit schedule to audit personnel.

5a. Insufficient audit personnel to conduct security audit.

1. System flags and presents status on insufficient audit personnel for security audit.
2. HoS overrides insufficient status and proceeds security audit with less personnel.
   2a. HoS chooses to reassign audit personnel from future or lower priority audits.
      a. System receives new request and updates both new and changed audit schedules.
3. System receives request and updates audit schedule with overridden requirement.
4. HoS continues updating system with audit schedule data.

Sales Department

[Mani Raj Rejinthala]

1) Use Case: ProcessNewOrder

Actor: Sales Representative

Precondition:

1. Sales department can access the inventory records.

Main Success Scenario:

1. A Retailer contacts the Sales Representative or meet the sales representative at office to place an order.
2. The Sales Representative logs into the system and select sales department
3. Then selects the “New Order Request” option.
4. The Retailer provides their retailer ID and the sales representative enters the id.
5. The system retrieves the retailer's details, such as name and location.
6. The Sales Representative verifies the retailer information
7. Retailer requests to add certain quantity of a list items to the order
8. Sales representative requests the product list at a particular storage
9. System displays the product list of a particular storage
10. Sales representative checks the product id for each of the requested item
11. Sales representative enters the requested fashion items ID’s along with their quantities.
12. The system confirms the availability of the requested items and asks for the confirmation to proceed.
13. The system generates a preliminary order summary.
14. The Sales Representative reviews the order summary with the Retailer.
15. The Sales Representative confirms and submits the order.
16. The system generates a unique order ID, and updates the inventory.
17. The System prompts do you need receipt and provides two options yes/no
18. Sales representative asks the retailer and enter the input.
19. Sales representative finishes the order request and logs out the system.

Extensions:

1. If the retailer requests the receipt:
   a. The System prompts do you need receipt and provides two options yes/no
   b. Sales representative enters Yes.
   c. System prints the receipt.
   d. Sales representative provides the receipt to the retailer.
2. If the Retailer ID is not valid:

a. The Sales Representative enters the retailer ID
b. The system prompts the ID is Invalid.
c. System prompts to re-enter correct retailer ID
    i. Sales representative verifies the ID with the retailer and renters the ID
    ii. System validates the ID and fetches the stored retailer information
    iii. Sales representative verifies the fetched details by enquiring with retailer.
    iv. Sales representative confirms the check.

2) Use Case: Handling Returns

Actor: Sales Representative

Precondition:

1. Sales department can access the inventory records.

Main Success Scenario:

1. A Retailer contacts the Sales Representative or meet the sales representative at office.
2. A Retailer requests to return items of an Order.
3. Sales Representative ask for order receipt.
4. Retailer provides the order receipt
5. The Sales Representative logs into the system and selects the sales department
6. Then, selects “Return Request”
7. System requests the order id
8. The Sales Representative enter the order id from the receipt
9. System confirms the order is still in return period and requests the return items details
10. Retailer informs product id (listed on receipt) and quantity of items to be returned
11. Sales representative enters the product Id’s and quantity
12. System verifies the items with the order information stored and prompts to confirm the condition of the items being returned.
13. Sales Representative inspects the items being requested for return and confirms to proceed.
14. System updates the order details of the returned items and updates the inventory to reflect the returned items.
15. System displays the return summary

16. Sales representative reviews the return summary with the retailer.
17. The Sales Representative requests the updated receipt.
18. System prints the updated receipt after excluding the returned items.
19. Sales Representative provides the updated receipt to retailer.

Extensions:

1. If the items requested for return belongs to an order whose return period is finished
   a. System prompts the items belong to the provided order Id can't be returned
   b. Sales representative verifies the dates and informs the retailer that the return can't completed.
   c. Sales representative rejects the return request.

2. If the quantity of an item returning exceeds the quantity of item bought
   a. Sales representative enters the list of return requested items details
   b. System displays the items and their quantity which are not in accordance with the order information stored in records.
   c. Sales representative reviews the information and rejects the return of those items.

3) Use Case: Register Retailer

Primary Actor: Sales Representative

Precondition: None

Main Success Scenario:

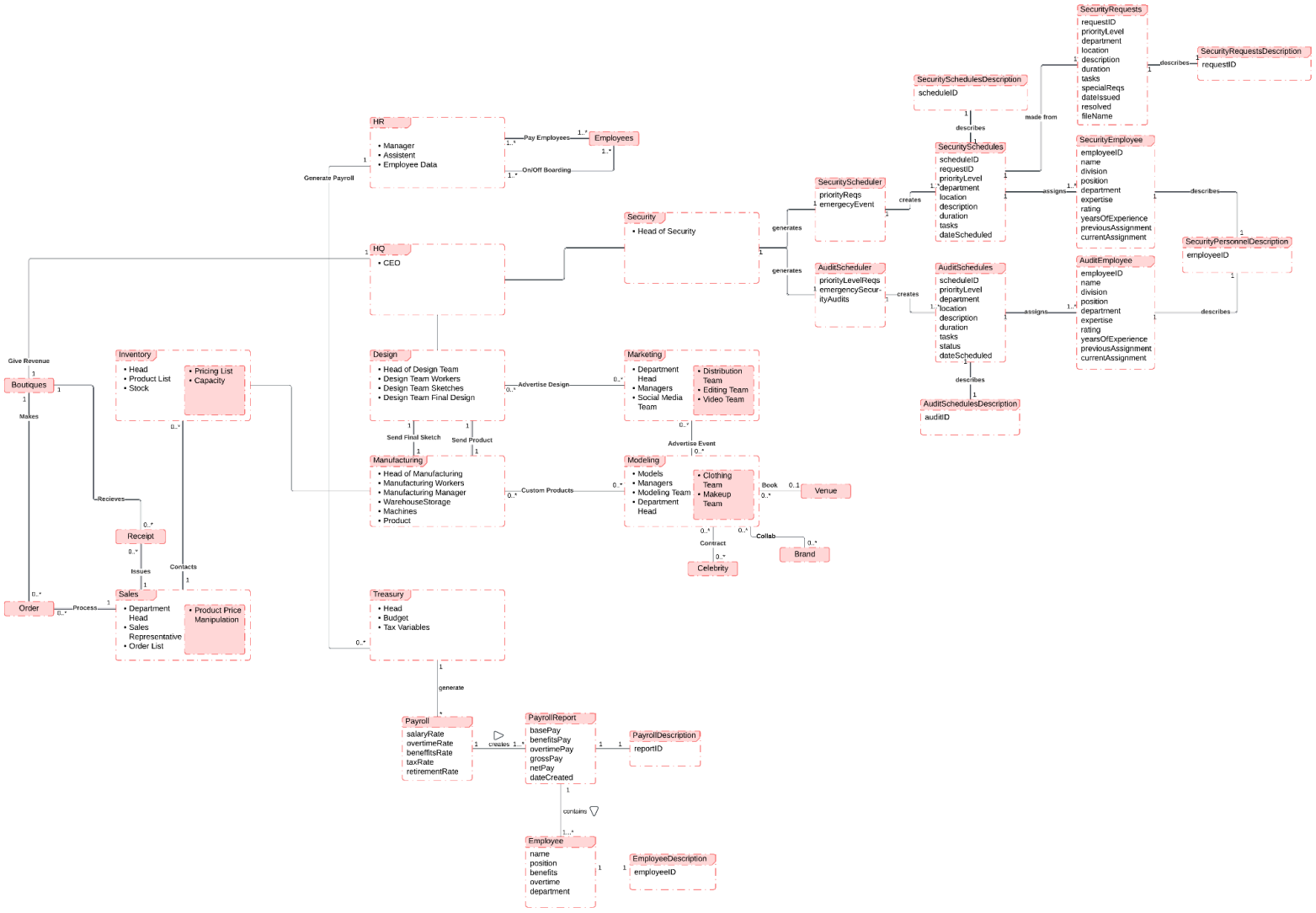
1. A Retailer contacts the Sales Representative or meet the sales representative at office.
2. The Sales Representative logs into system and selects the sales department
3. The Sales Representative selects the "Register New Retailer" option
4. System prompts to enter the retailer details
5. Retailer provides their name and location
6. The Sales Representative enter Retailer Name and Location
7. The system validates the entered data to ensure there are no duplicates or missing fields.

8. The Sales Representative confirms the registration.
9. The system saves the retailer's details, assigns a unique Retailer ID.
10. The system generates a registration confirmation, including the Retailer ID, name and location, which the Sales Representative provides to the retailer.
11. The Sales Representative completes the registration process and logs out.

Extensions:

1. If the system detects missing or invalid information:
    a. The system prompts to enter the incomplete fields
    b. The Sales Representative corrects the information and resubmits the form.
    c. The system revalidates the corrected information and proceeds the registration.
2. If a duplicate retailer is detected
    a. The system notifies the Sales Representative of the potential duplicate and prompts to confirm whether still interested to continue where it provides yes/no.
    b. The Sales Representative checks the existing records to confirm
        i. if it is indeed a duplicate
            I. Sales representative selects no and cancel the registration.
            II. System acknowledges the input and displays the retailer information from the records
            III. Sales representative provides the existing Id to the retailer
        ii. If it is a new registration with similar name.
            I. Sales representative selects yes and continues the registration.
            II. System acknowledges the input and displays the new registered retailer information
            III. Sales representative provides the generated retailer Id to the retailer

Domain Model



Information Expert

Marketing/Modeling:

The information expert for these departments is the department head, and the department itself. For example, the department head can create events, or ask the managers to schedule fittings. The department takes the events/fittings created by the head and stores them in the file system. Any other department requesting information from these departments must go through the department to do so. In each department file, there are different public methods that other departments can call to receive information from the file system, parsed into an object for ease of use. The department is defined in the App file, which is accessible to anyone.

Sales

1. addOrderRequest: The Information expert for addOrderRequest is saleManage as it has order and retailer details. In detail, salesMange creates the instances of order and retailer objects and calls the necessary methods of those to fulfill the order.

2. ReturnOrderRequest: The Information expert for ReturnOrderRequest is saleManage as it has order and retailer details. In detail, salesMange has the list of order instances and associated retailer instances so it can manage the order request.

3. RegisterRetailer: The information Expert for RegisterRetailer is salesManage as it knows the retailer. It creates the new instance of retailer with the provided retailer information.

4. ChangeProductPrice: The information Expert for ChangeProductPrice is SalesInventoryRequest as it creates the StorageManage instance which has product details and price alter methods. So it calls the changePrice methods using the StorageManage instance and updates the price of a specific product.

HR

1. Employee: The information expert for Employee is employeeRecordManager, which interacts with and performs operations on the Employee data. It additionally manages collections of employee information and knows about how and where to store them
2. Candidate: The information expert for Candidate is candidateRecordManager, as it interacts with and performs operations on all Candidate objects. It knows how to read and write information from Candidate txt files.
3. File System: the information expert for managing files and the operations to read and write to them is the fileStorageHR class, which maintains the responsibility of handling all file operations for Candidate and Employee objects, through the RecordManagers.

Design

1. **manageDesignCommunication**: The information expert for manageDesignCommunication is Design. The Design department is in charge of multiple forms of communication between other departments such as Modelling, Marketing, and Manufacturing. The Design Department will receive a request from the Modelling department to create a custom design for one of the models, this design is then sent to manufacturing to be created. The design Team also decided on a final design that they want to produce. This design is then sent to the manufacturing department to be created, then sent to the marketing department to begin advertising the product.

Manufacturing

1. **manageProdution:** The information expert for manageProduction is Manufacturing. The Manufacturing department is responsible for accepting designs from the Design team and making products for Modelling department and consumers. After receiving the design they will being production with the required raw materials. When the product is created it is sent to Inventory and the Modelling Department to be verified further. Once the Product is verified they will begin production on the required amount needed.

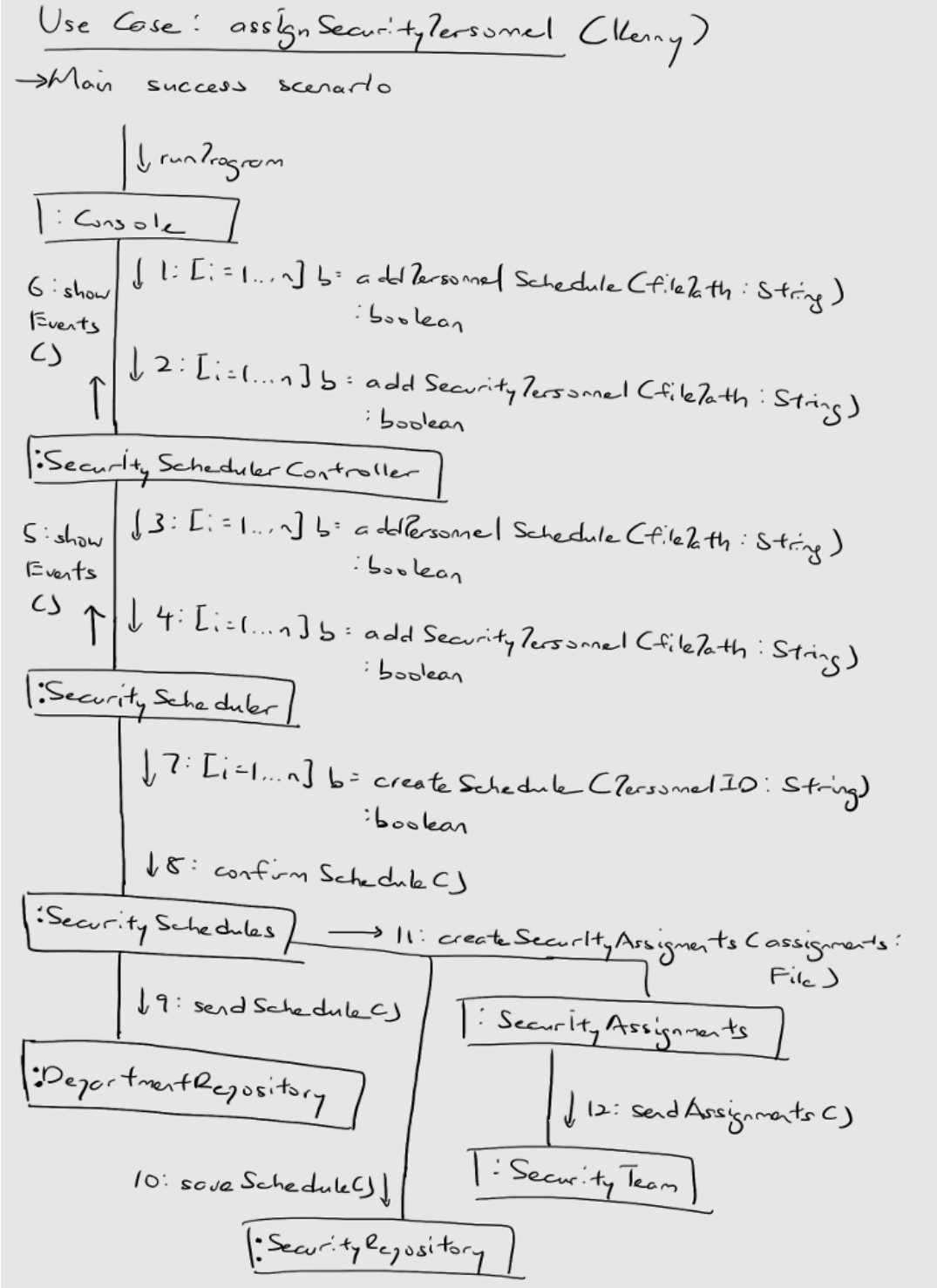
Security

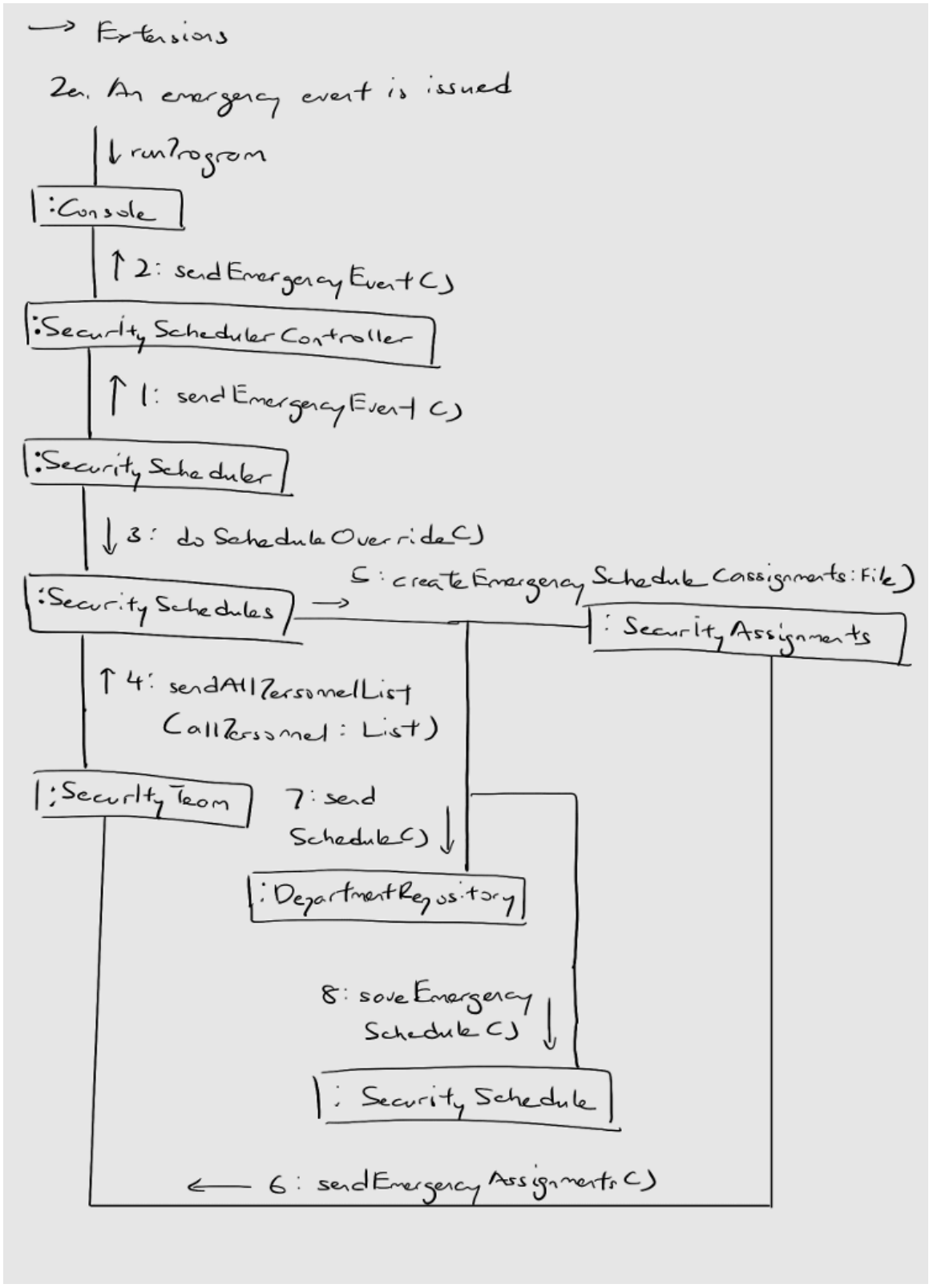
1. **assignSecurityPersonnel**
   The information expert for assignSecurityPersonnel is SecurityScheduler. It knows SecurityEmployee, SecurityRequests, and SecuritySchedules. The method assigned to SecurityScheduler will initiate the creation of a security schedule by retrieving SecurityEmployee and SecurityRequests instances from the repository, then putting them into SecuritySchedules.

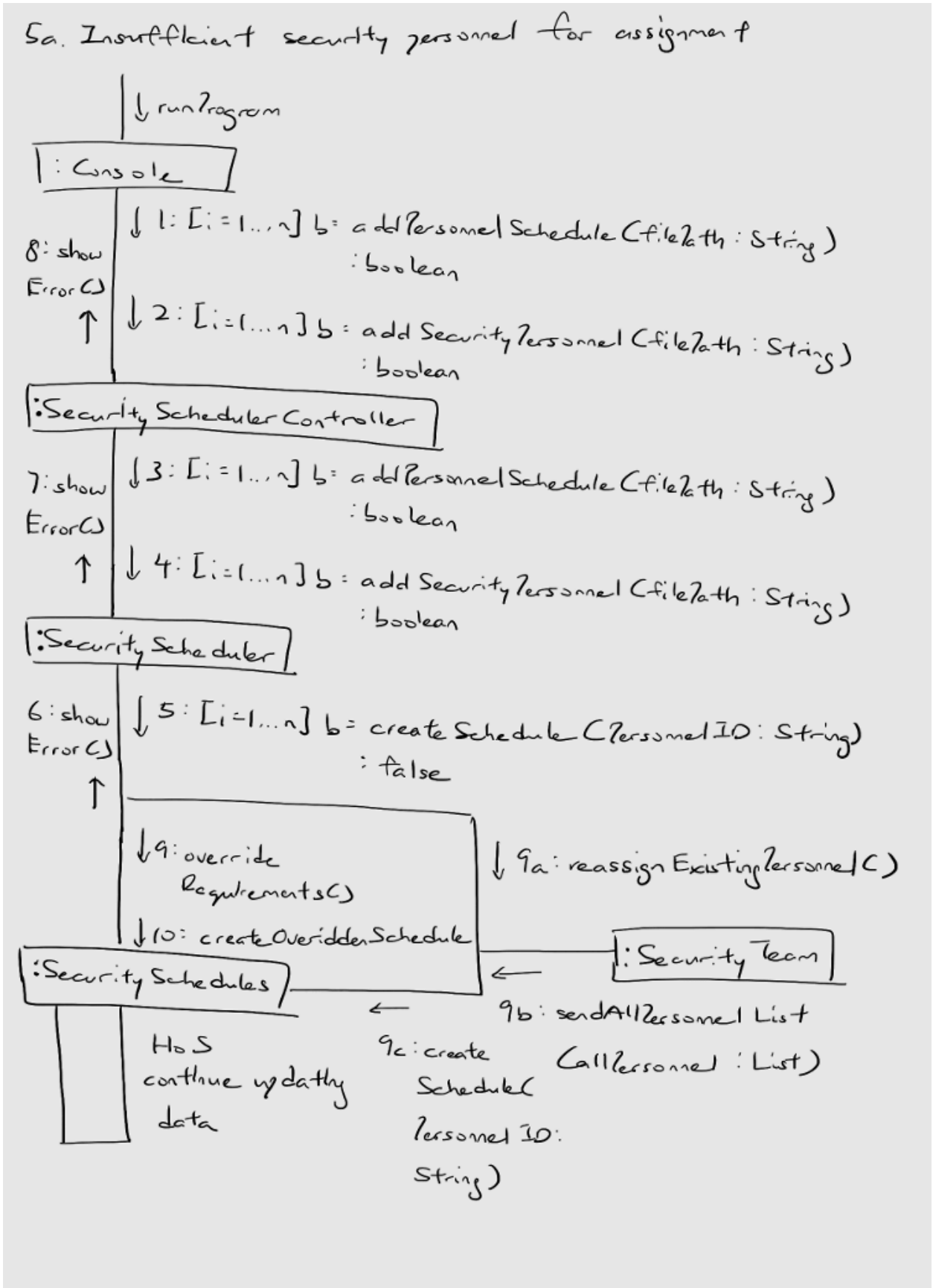
2. **manageSecurityAudits**
   The information expert for manageSecurityAudits is AuditScheduler. It knows AuditEmployee and AuditSchedules. The method assigned to AuditScheduler will initiate the creation of a security audit by retrieving AuditEmployee instances from the repository, then putting them into AuditSchedules.

Interaction Diagrams (Communication)

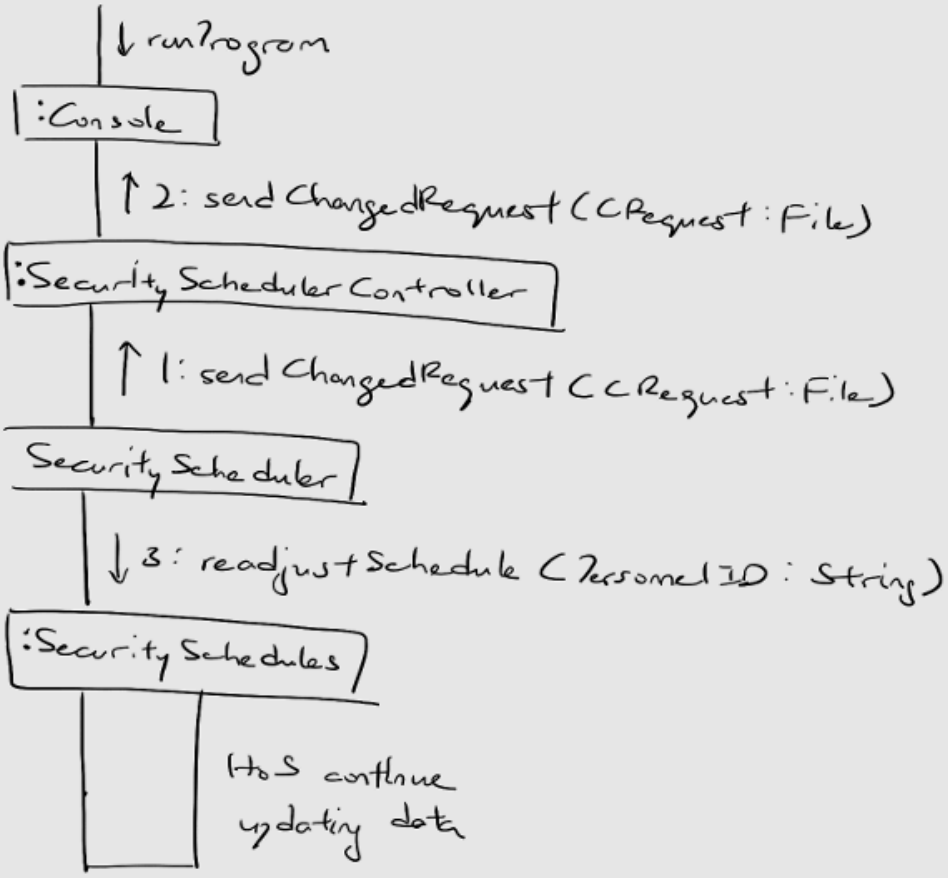
Security

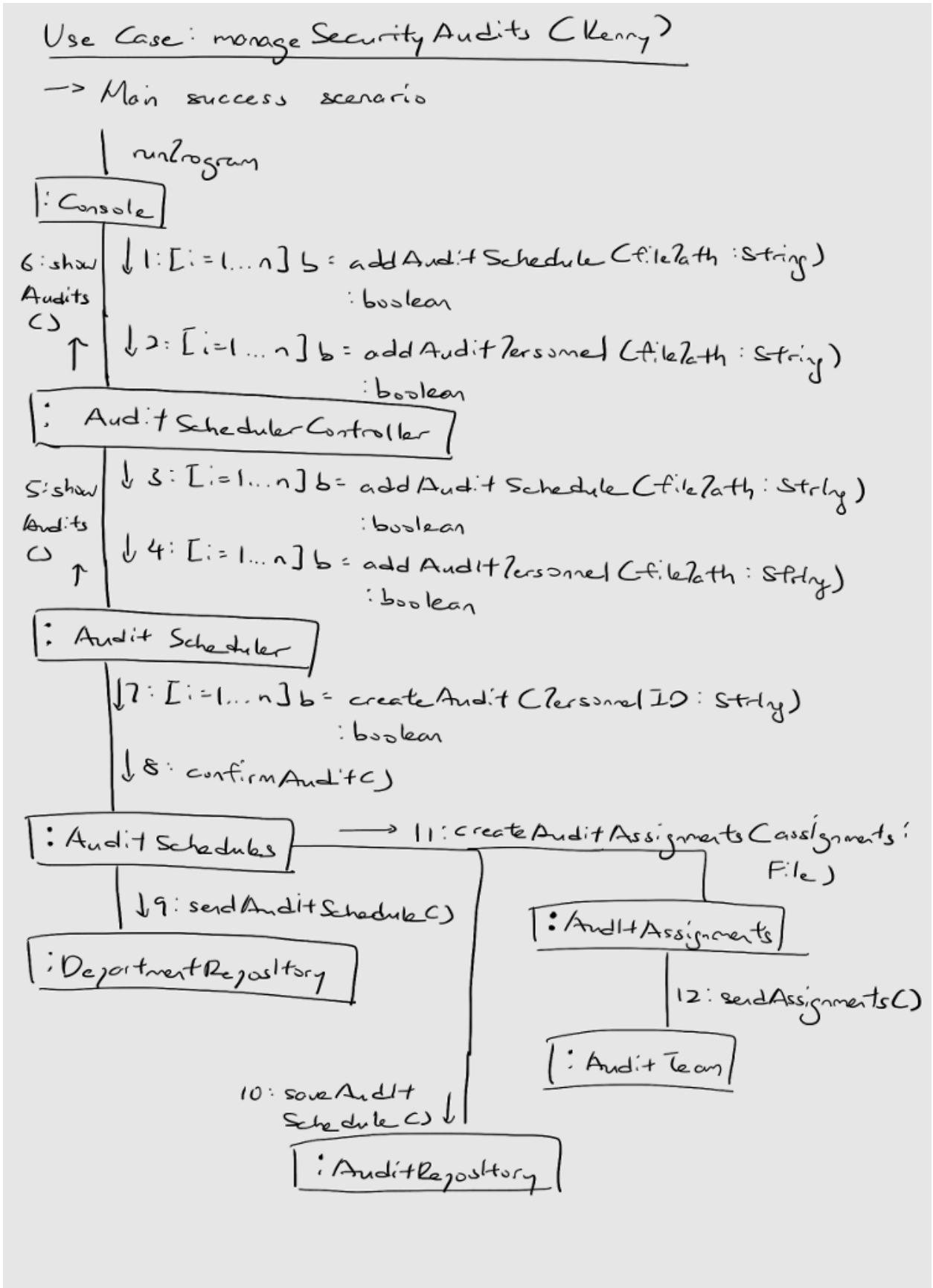



→ Extensions
2a. An emergency event is issued
runProgram()
:Console
2: sendEmergencyEvent()
:SecuritySchedulerController
1: sendEmergencyEvent()
:SecurityScheduler
3: doScheduleOverride()
:SecuritySchedules
5: createEmergencyScheduleAssignments:File()
:SecurityAssignments
4: sendAllPersonnelList(allPersonnel: List)
:SecurityTeam
7: send Schedule()
:DepartmentRepository
8: saveEmergency Schedule()
:SecuritySchedule
6: sendEmergencyAssignments()


5a. Insufficient security personnel for assignment
run Program
: Console
1: [i = 1...n] b = addPersonnelSchedule(filePath : String) : boolean
8: show Error()
2: [i = 1...n] b = addSecurityPersonnel(filePath : String) : boolean
:Security Scheduler Controller
3: [i = 1...n] b = addPersonnelSchedule(filePath : String) : boolean
7: show Error()
4: [i = 1...n] b = addSecurityPersonnel(filePath : String) : boolean
:Security Scheduler
5: [i = 1...n] b = createSchedule(PersonnelID : String) : false
6: show Error()
9: overrideRequirements()
9a: reassignExistingPersonnel()
10: createOveriddenSchedule
:Security Team
:Security Schedules
9b: sendAllPersonnelList(allPersonnel : List)
HoS continue updating data
9c: createSchedule(PersonnelID : String)

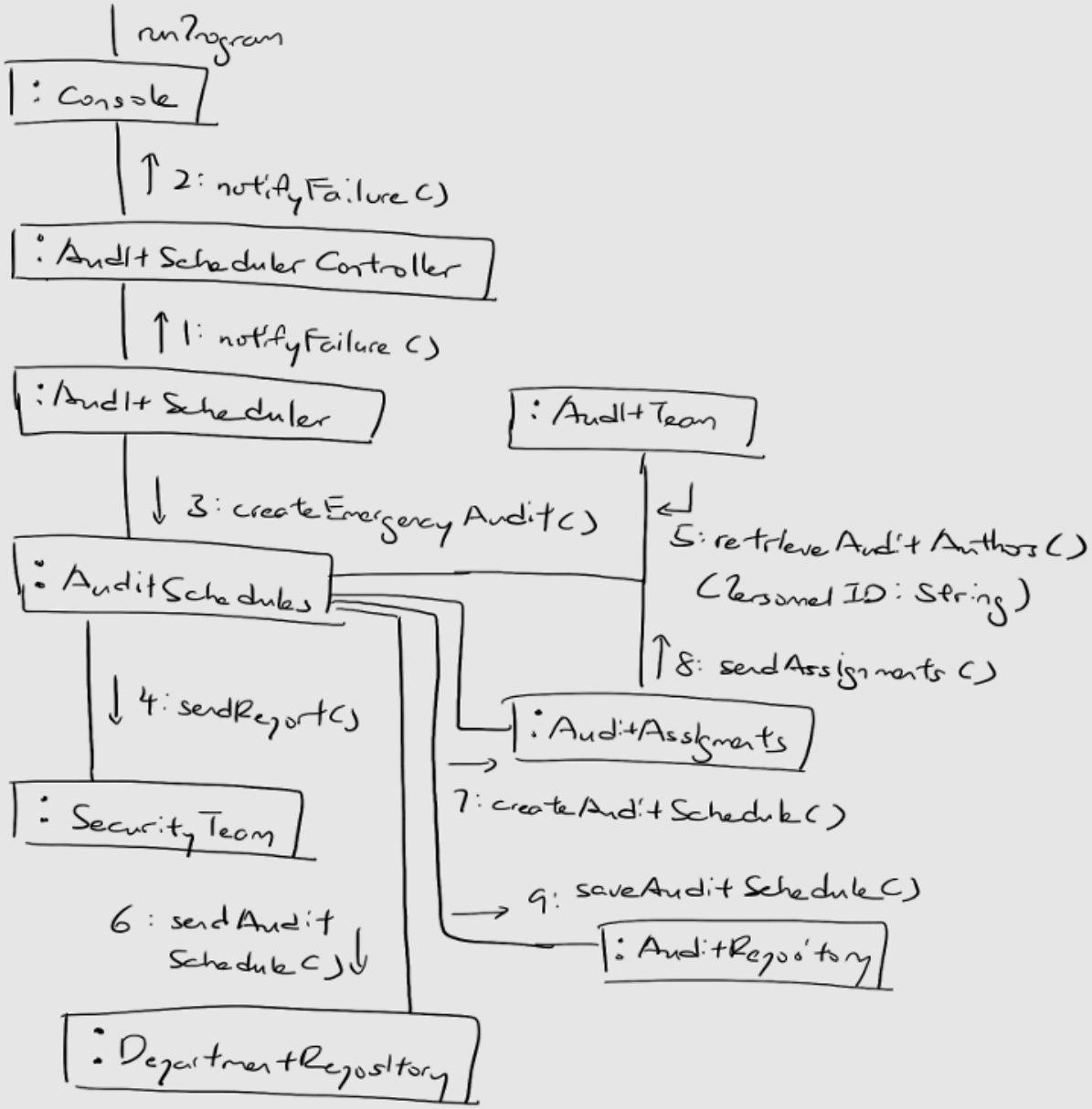
55. Receives existing security request with changed priority level or requirements.

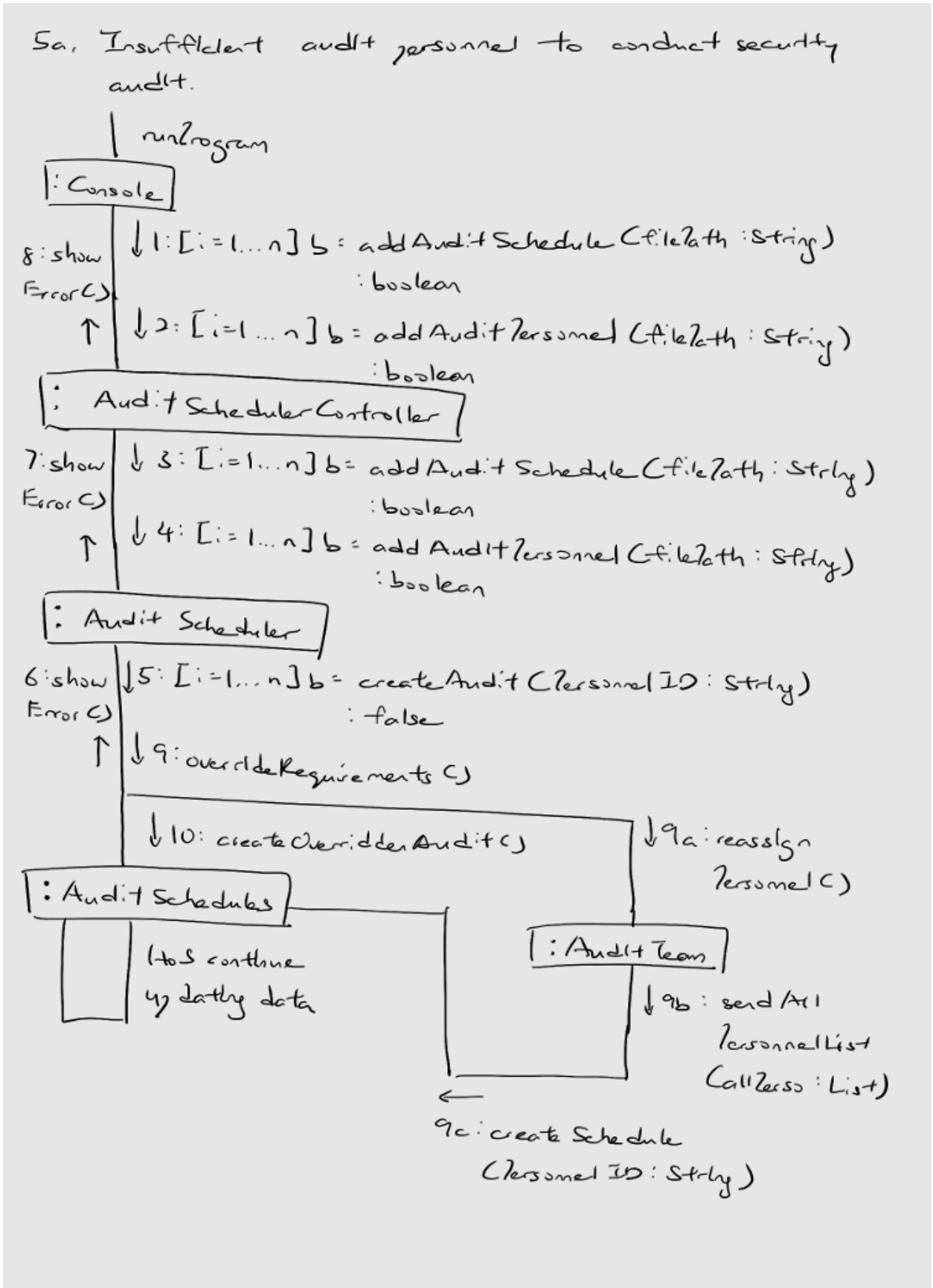



Use Case: manage Security Audits (Kerry)
-> Main success scenario
run Program
: Console
6: show Audits ()
1: [i=1...n] b = add Audit Schedule (file Path : String) : boolean
2: [i=1...n] b = add Audit Personnel (file Path : String) : boolean
: Audit Scheduler Controller
5: show Audits ()
3: [i=1...n] b = add Audit Schedule (file Path : String) : boolean
4: [i=1...n] b = add Audit Personnel (file Path : String) : boolean
: Audit Scheduler
7: [i=1...n] b = create Audit (Personnel ID : String) : boolean
8: confirm Audit ()
: Audit Schedules
11: create Audit Assignments (assignments : File)
9: send Audit Schedule ()
: Department Repository
: Audit Assignments
12: send Assignments ()
: Audit Team
10: save Audit Schedule ()
: Audit Repository

→ Extensions:

2a. Audit failure that indicates critical vulnerability.

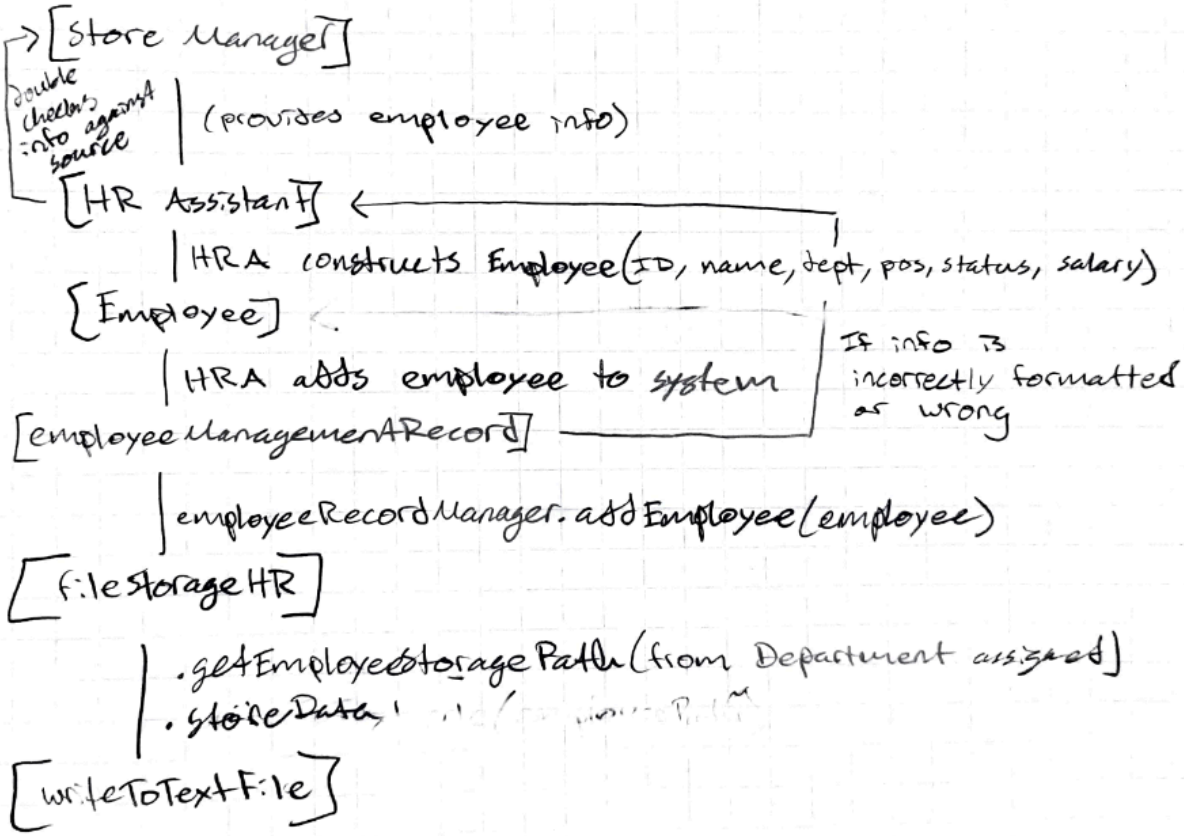


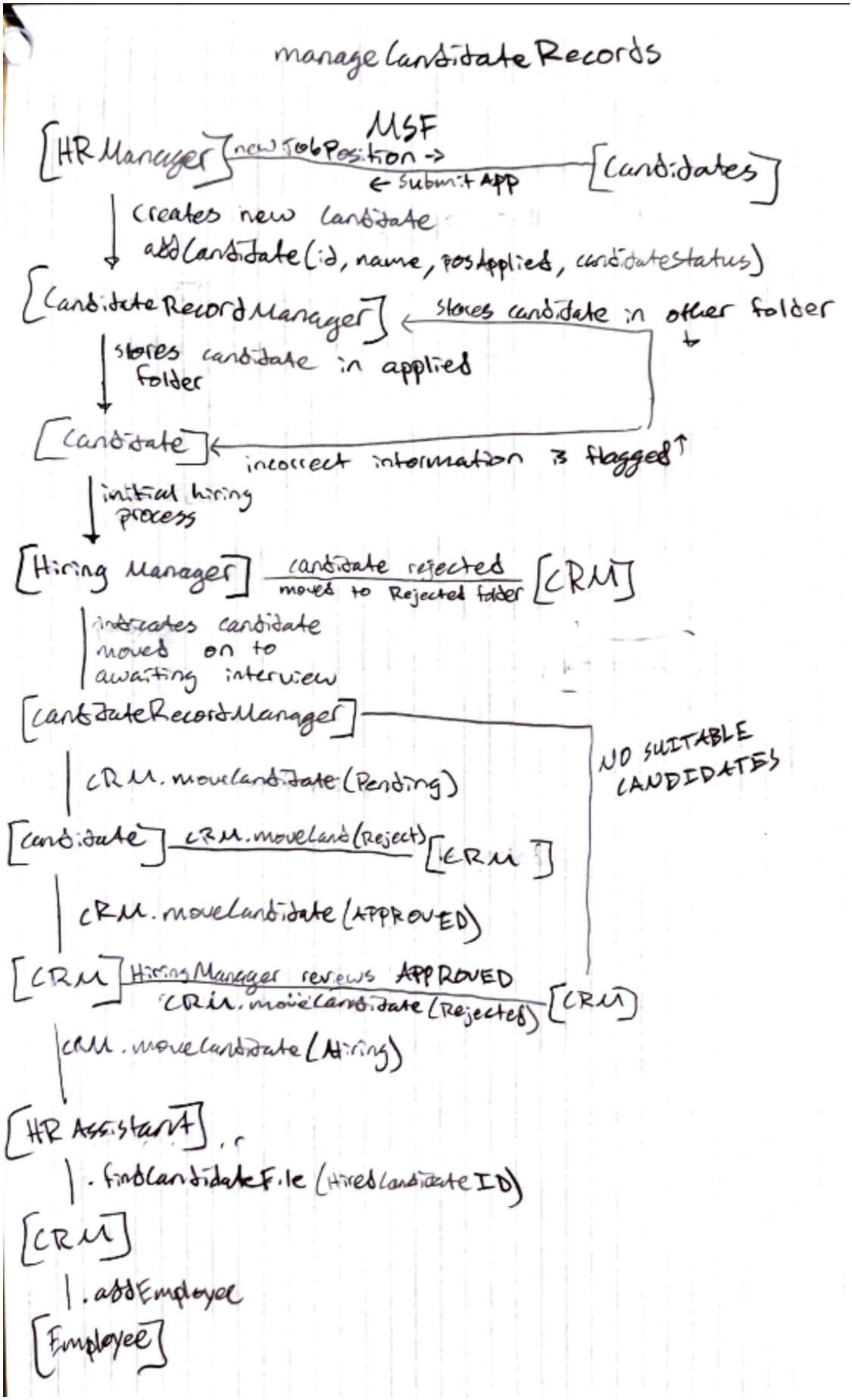

5a. Insufficient audit personnel to conduct security audit.
runProgram
: Console
1: [i=1...n] b = addAuditSchedule(filePath : String) : boolean
2: [i=1...n] b = addAuditPersonnel(filePath : String) : boolean
8: showError()
: AuditSchedulerController
3: [i=1...n] b = addAuditSchedule(filePath : String) : boolean
4: [i=1...n] b = addAuditPersonnel(filePath : String) : boolean
7: showError()
: AuditScheduler
5: [i=1...n] b = createAudit(personnelID : String) : false
9: overrideRequirements()
6: showError()
10: createOverriddenAudit()
9a: reassignPersonnel()
: AuditSchedules
Has continue by adding data
: AuditTeam
9b: sendAuditPersonnelList(callPersons : List)
9c: createSchedule(personnelID : String)

HR:

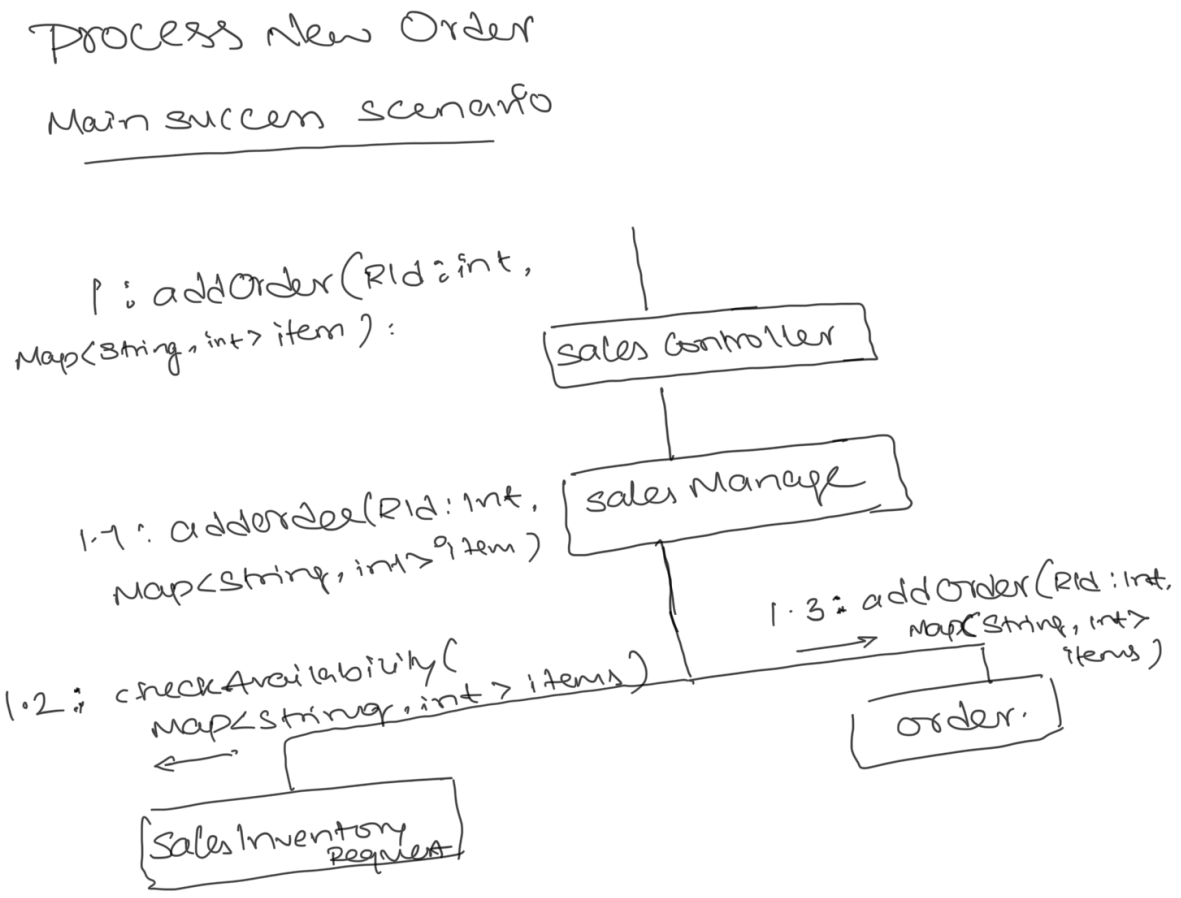
manageEmployeeRecords

MSF

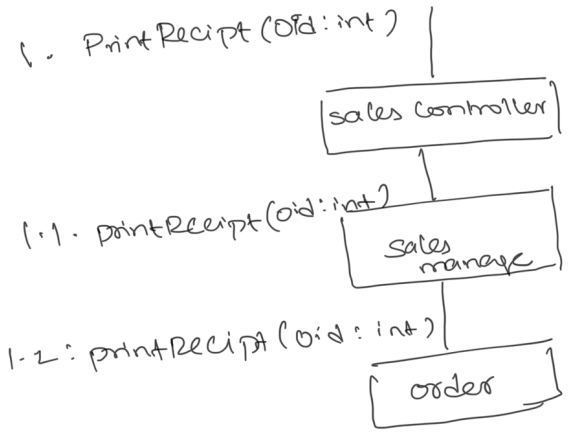



manage Candidate Records
[HR Manager]
new Job Position ->
MSF
← Submit APP
[Candidates]
creates new candidate
addCandidate(id, name, posApplied, candidateStatus)
[Candidate Record Manager]
stores candidate in other folder
stores candidate in applied folder
[Candidate]
incorrect information is flagged
initial hiring process
[Hiring Manager]
candidate rejected
moved to Rejected folder
[CRM]
indicates candidate moved on to awaiting interview
[CandidateRecordManager]
NO SUITABLE CANDIDATES
CRM.moveCandidate(Pending)
[candidate]
CRM.moveCand(Reject)
[CRM]
CRM.moveCandidate(APPROVED)
[CRM]
Hiring Manager reviews APPROVED
CRM.moveCandidate(Rejected)
[CRM]
CRM.moveCandidate(Hiring)
[HR Assistant]
.findCandidateFile(HiredCandidateID)
[CRM]
.addEmployee
[Employee]

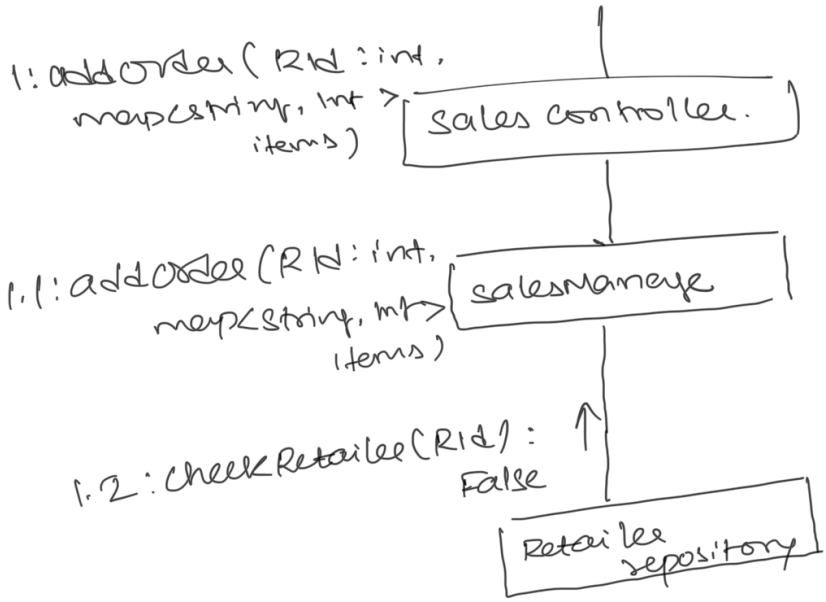
Sales


Process New Order
Main succes scenario
1 : addOrder(RId : int, Map<String, int> item) :
sales Controller
1.1 : addorder(RId : int, Map<String, int> item)
sales Manage
1.2 : checkAvailability(Map<String, int> items)
1.3 : addOrder(RId : int, Map<String, int> items)
order.
SalesInventory Register

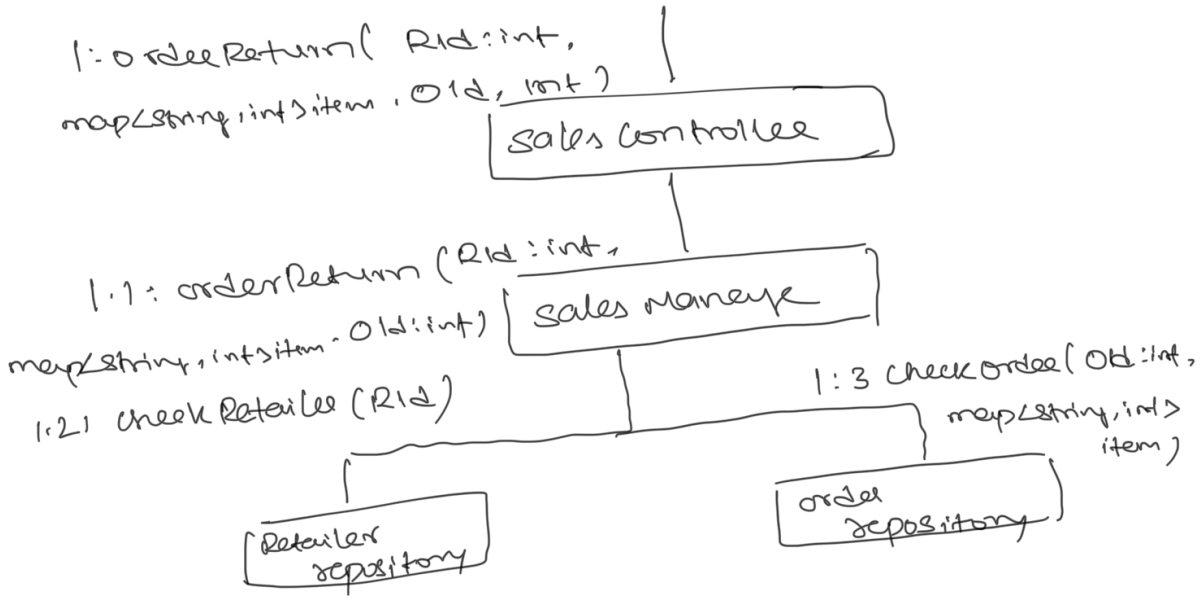
Extension 1



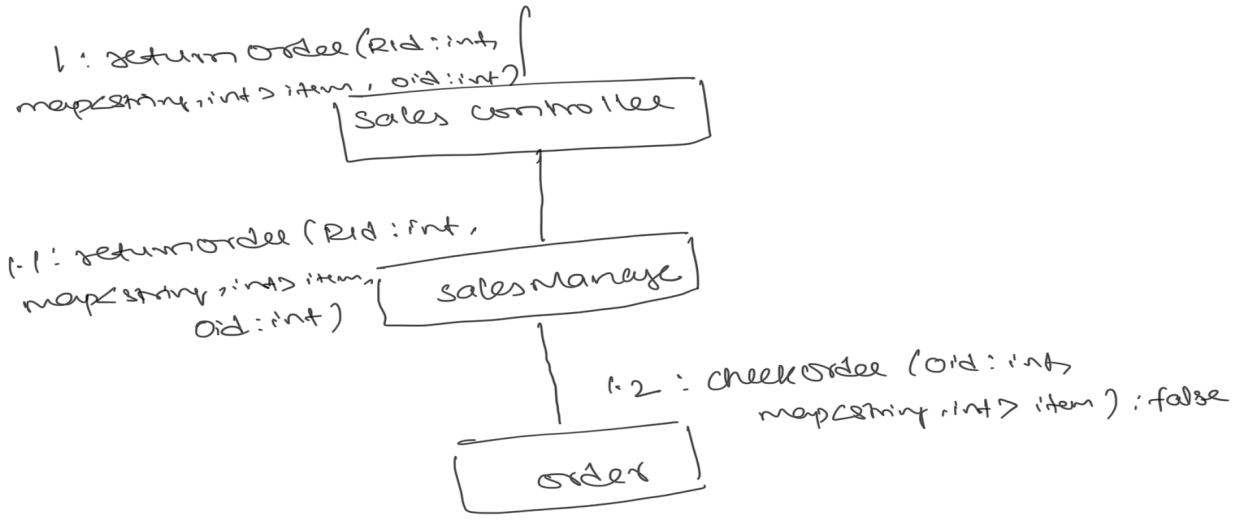
Extension 2

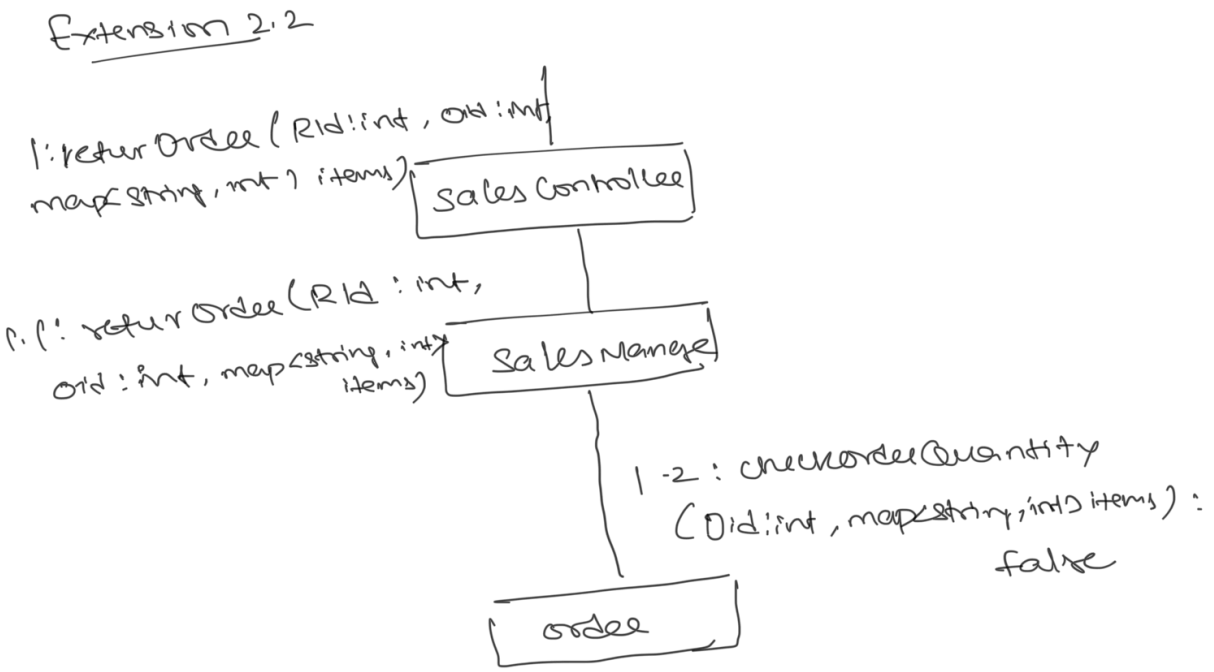


Handling Return:
Main Success Scenario:

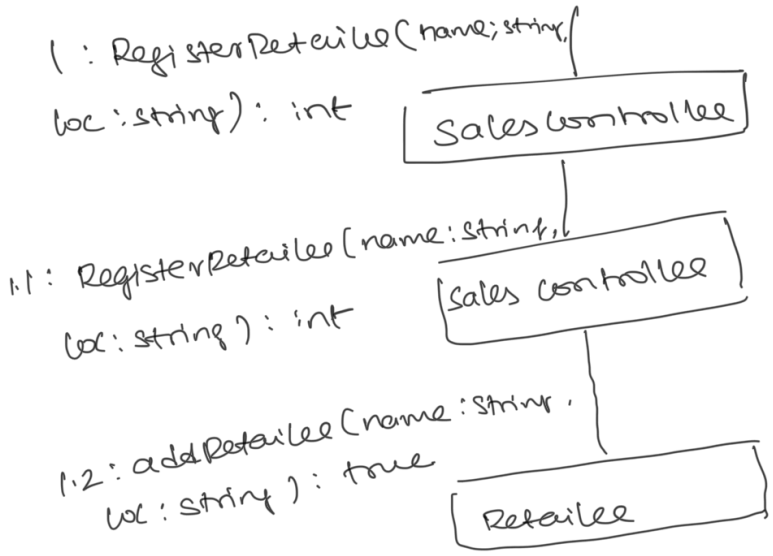


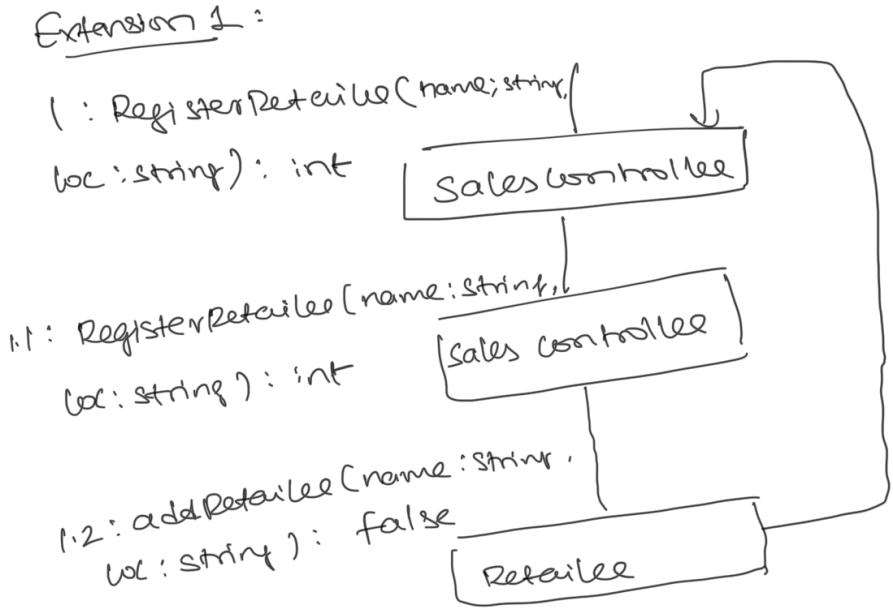
Extension 1:



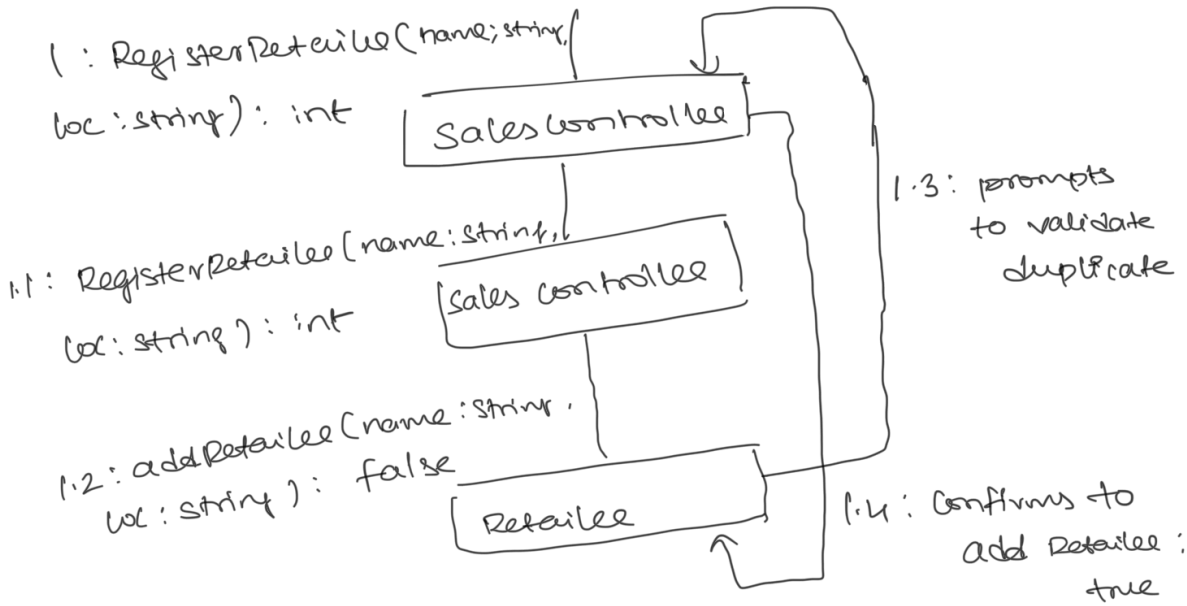


Register Retailer:
Main Success Scenario:




Extension 1 :
1 : RegisterRetailer(name:string,
loc:string) : int
SalesController
1.1 : RegisterRetailer(name:string,
loc:string) : int
SalesController
1.2 : addRetailer(name:string,
loc:string) : false
Retailer

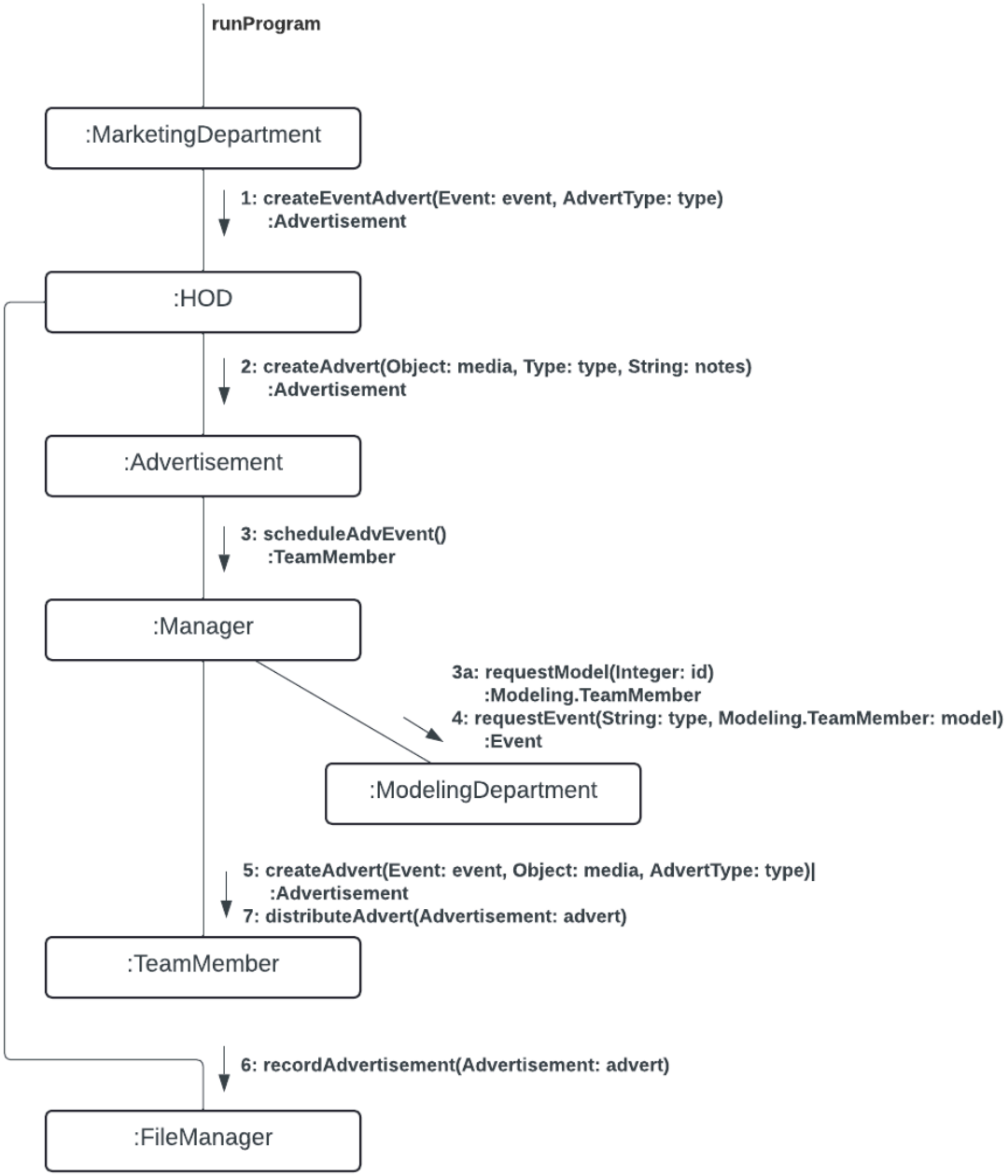
Extension 2:


1 : RegisterRetailer(name:string,
loc:string) : int
SalesController
1.1 : RegisterRetailer(name:string,
loc:string) : int
SalesController
1.2 : addRetailer(name:string,
loc:string) : false
Retailer
1.3 : prompts
to validate
duplicate
1.4 : Confirms to
add Retailer :
true

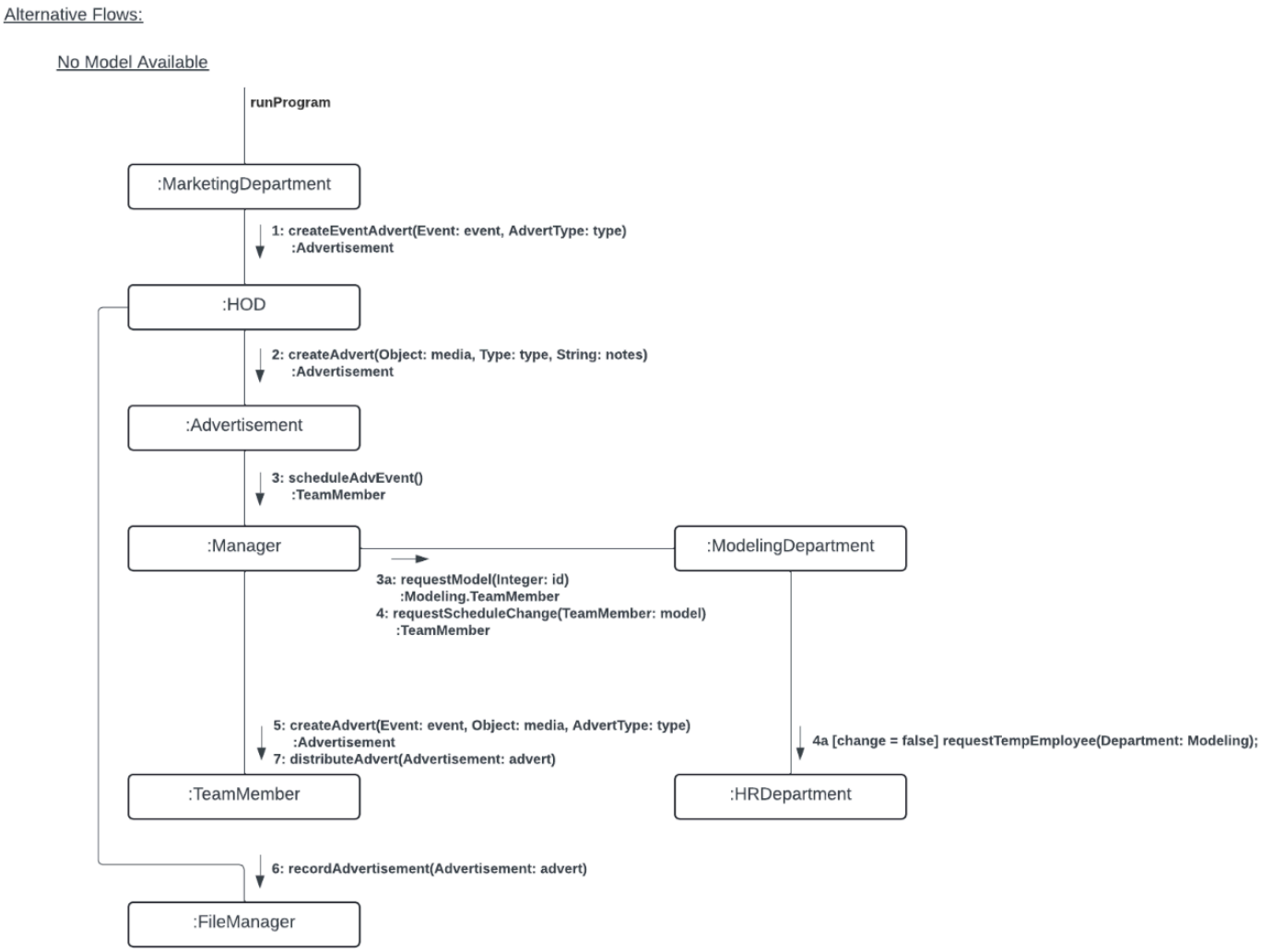
Marketing [Mia]:

Use Case: advertiseEvent

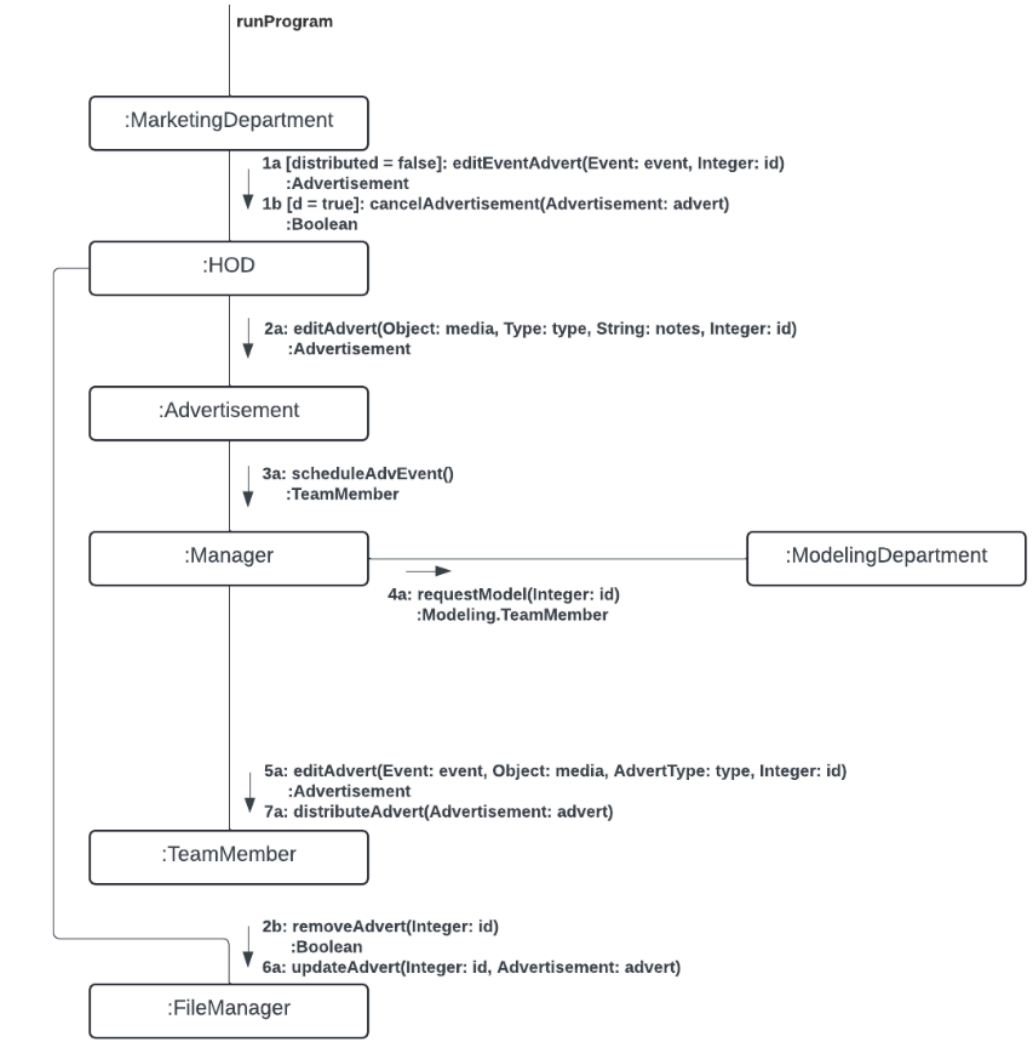
Main Success Scenario:



Alternative Flows:

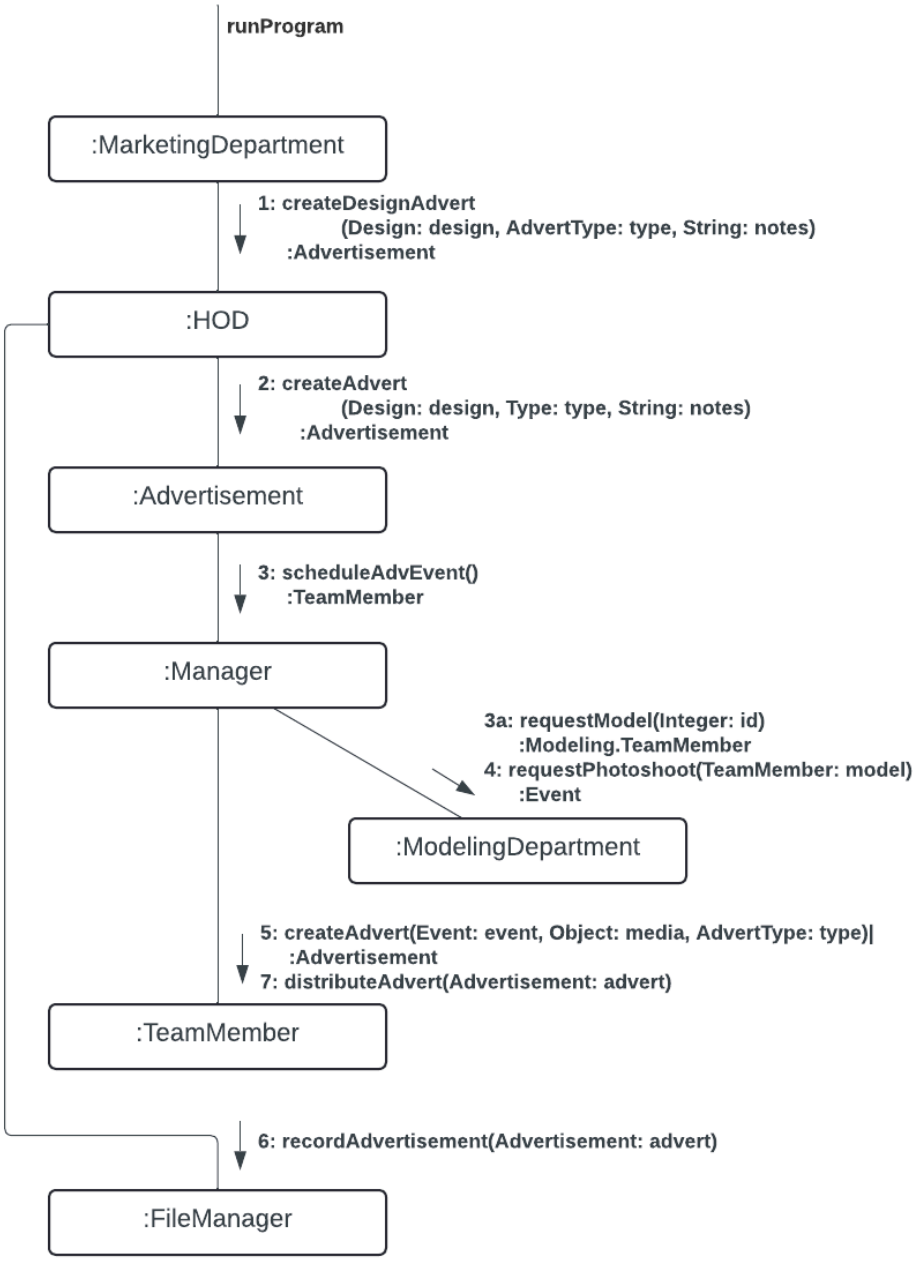


Event Canceled/Brand or Celebrity Backs out



Use Case: advertiseDesign

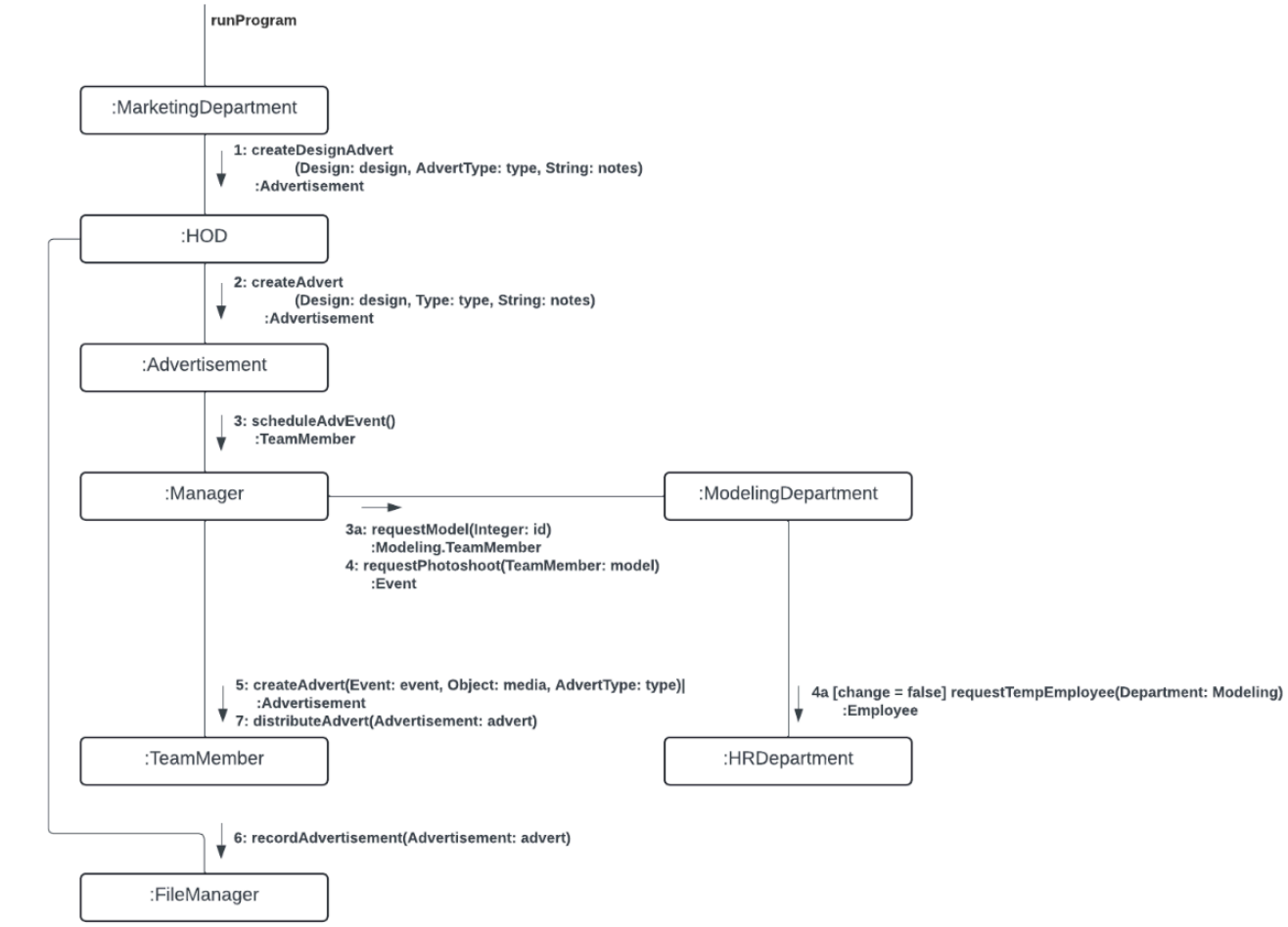
Main Success Scenario:

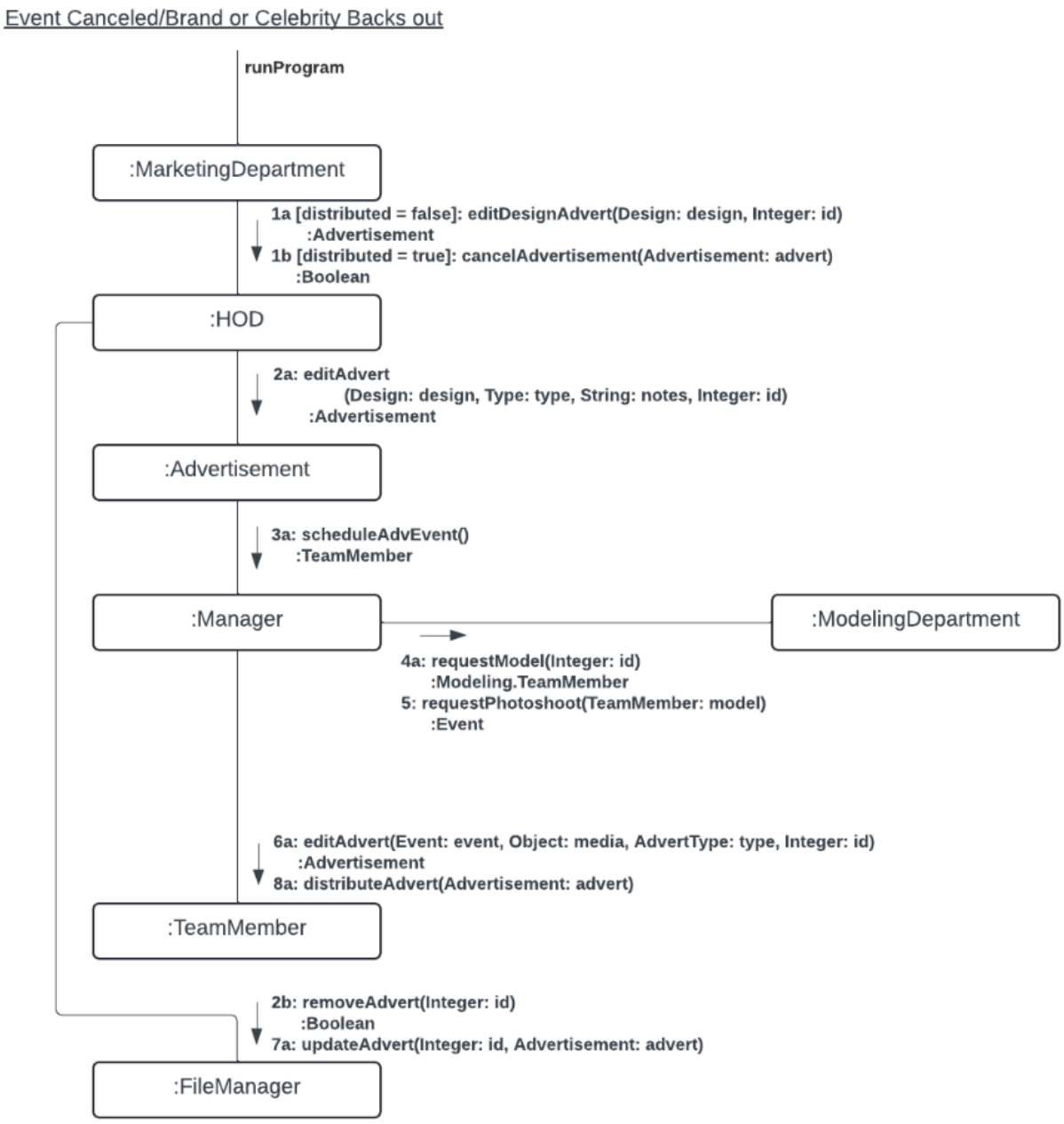


Alternative Flows

Alternative Flows:

No Model Available

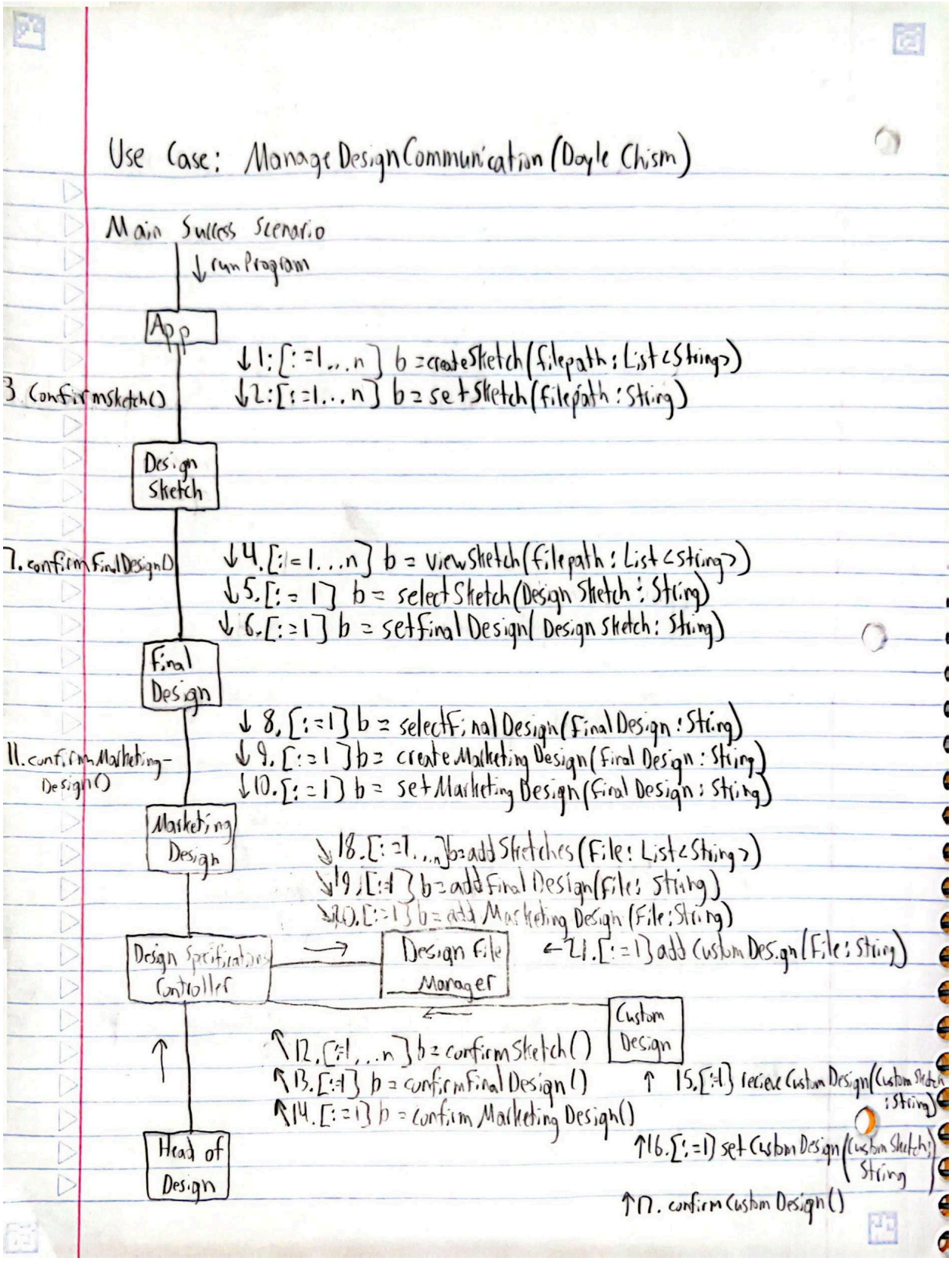




Design [Doyle Chism]

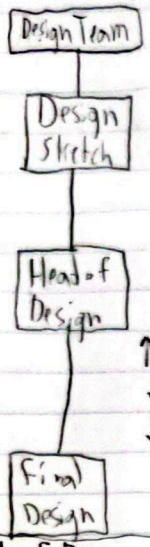
Use Case: Manage Design Communication (Doyle Chism)

Main Success Scenario



Alternate Flows

1-4A:



↓1. [:=1...n] b = createSketch(filepath: List<String>)

↓2. [:=1...n] b = viewSketch(filepath: List<String>)

↓3. [:=1...n] b = selectSketch(DesignSketch: String)

↑↓4. [:=1...n] b = verifySketch(DesignSketch: List<String>) : boolean

↓5. [:=1...n] b = modifySketch(DesignSketch: String)

↓6. [:=1] b = confirmFinalDesign(DesignSketch: String) : boolean

5A-5B:

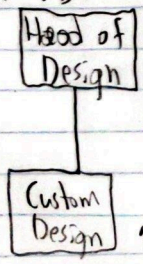
↓1. [:=1] b = selectFinalDesign(FinalDesign: String)

↓2. [:=1] b = createMarketingDesign(FinalDesign: String)

↓3. [:=1] b = setMarketingDesign(MarketingDesign: String)

↑4. [:1] b = confirmMarketingDesign()

7A-7B:



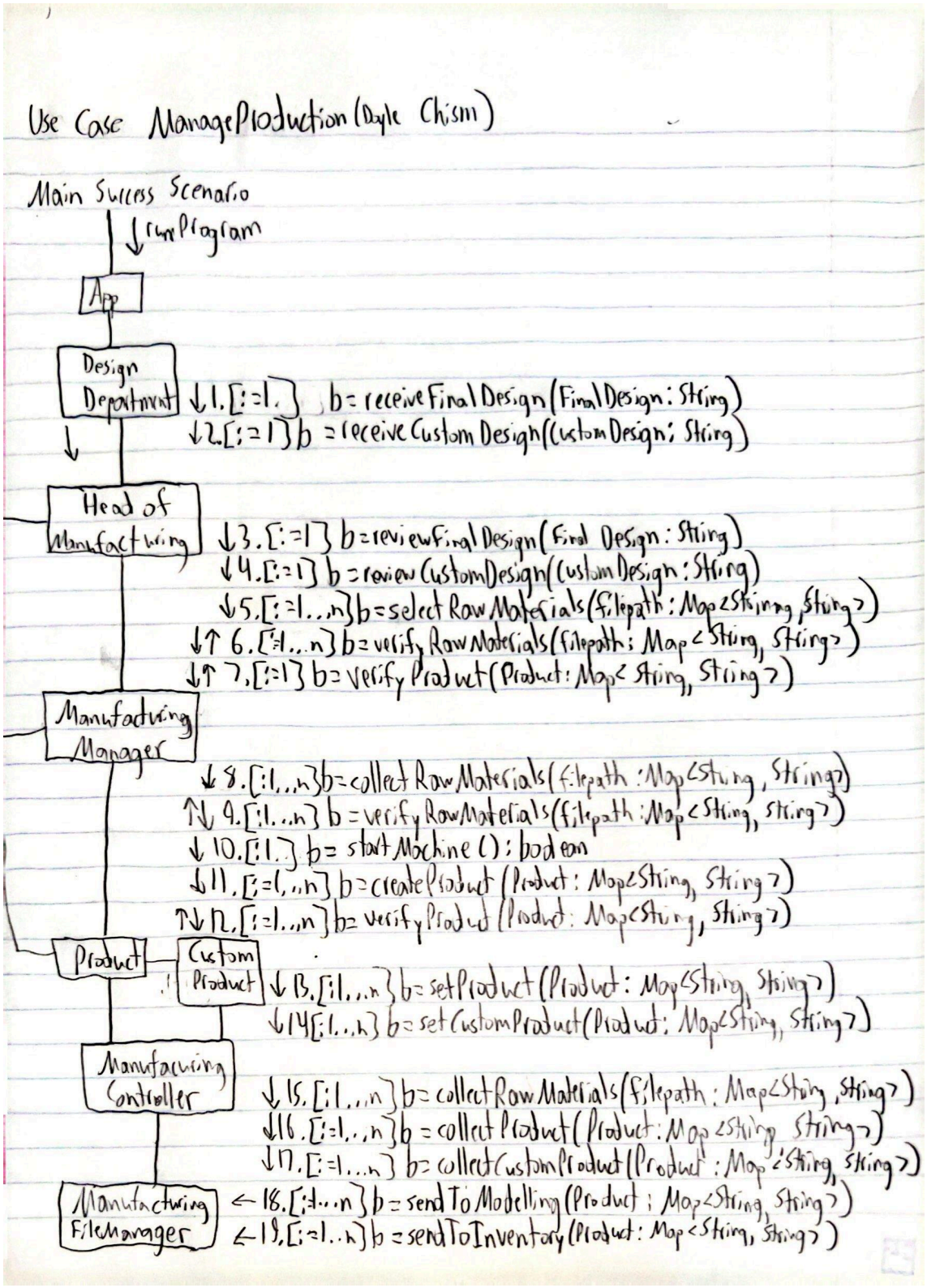
↓1. [:1] b = recieveCustomDesign(CustomDesign: String)

↓2. [:1] b = viewCustomDesign(CustomDesign: String)

↓3. [:1] b = setCustomDesign(CustomDesign: String)

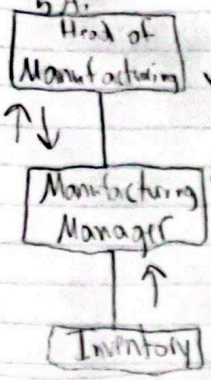
↑4. [:1] b = confirmCustomDesign()

Manufacturing [Doyle Chism]



Alternate Flows

5A:

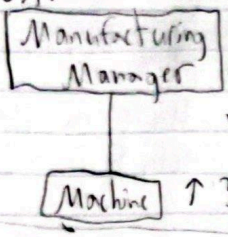


↓2. [i:=1...n] b=verifyRawMaterials(filepath: Map<String, String>)

↓1. [i:=1...n] b=collectRawMaterials(filepath: Map<String, String>)

↑3. [i:=1...n] b=verifyRawMaterials(filepath: Map<String, String>)

6A:

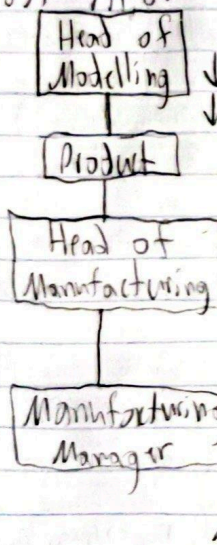


↓1. [i:=1] b=verifyMachineType(Machine: String)

↓↑2. [i:=1] b=startMachine(Machine: boolean)

↑3. [i:=1] b=machineStatus(Machine: boolean)

8A-9AB:



↓↑1. [i:=1] b=verifyCustomProduct(Product: Map<String, String>)

↓2. [i:=1] b=sendProduct(Product: Map<String, String>)

↑↓3. [i:=1...n] b=verifyQuality(Product: Map<String, String>)

↑↓4. [i:=1...n] b=resetProduct(Product: Map<String, String>)

↑↓5. [i:=1...n] b=deliverToManager(filepath: Map<String, String>)

↑6. [i:=1...n] b=collectRawMaterials(filepath: Map<String, String>)

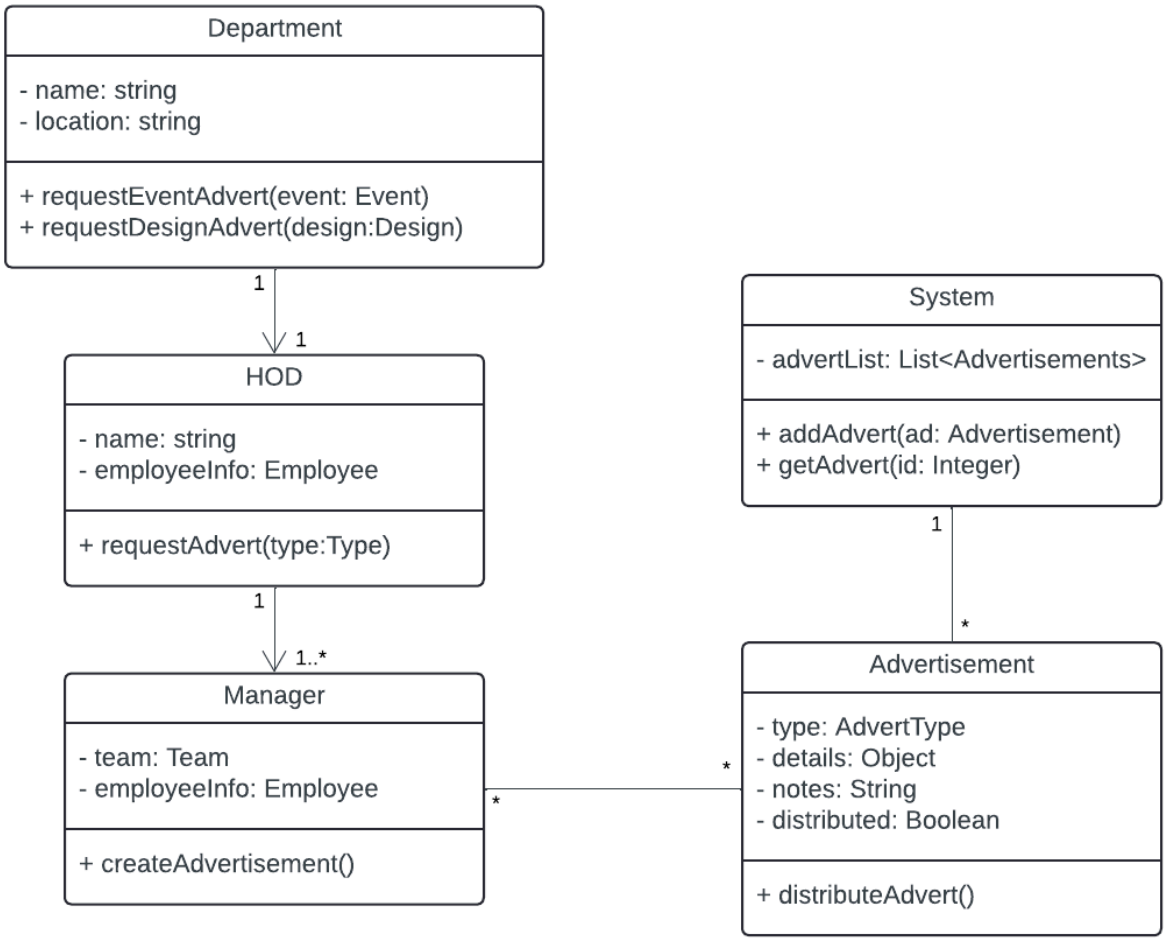
↑↓7. [i:=1...n] b=verifyRawMaterials(filepath: Map<String, String>)

↑8. [i:=1...n] b=createNewProduct(Product: Map<String, String>)

↑9. [i:=1...n] b=deliverNewProduct(Product: Map<String, String>)

Class Diagrams

Marketing:



Manufacturing [Doyle Chism]

Design Department
- FinalDesign finalDesign : Object - CustomDesign customDesign : Object
+ sendFinalDesign(finalDesign) : Object + sendCustomDesign(customDesign) : Object

Head of Manufacturing
- CustomDesign customDesign : Object - FinalDesign finalDesign : Object - Invetory rawMaterials : Map<String, String> - Product product : Object
+ reviewFinalDesign(finalDesign) : Object + reviewCustomDesign(customDesign) : Object + selectRawMaterials(rawMaterials) : Map<String, String> + verifyRawMaterials(rawMaterials) : Map<String, String> + verifyProduct(product) : Object

Product
- FinalDesign product : Object - CustomDesign customProduct : Object
+ setProduct(product) : Object + setCustomProudct(customProduct) : Object

Manufacturing Controller
- Inventory rawMaterials : Map<String, String> - Product product : Object - Product customProduct : Object
+ collectRawMaterials(rawMaterials) : Map<String, String> + collectProduct(product) : Object + collectCustomProduct(customProduct) : Object

Manufacturing File Manager
- Product customProduct : Object - Product product : Object
+ sendToModelling(customProduct) : Map<String, Object> + sendToInvetory(product) : Map<String, Object>

Design [Doyle Chism]

Custom Design
- CustomDesign : designName : String
+ ReceiveCustomDesign(designName) : String + setCustomDesign(designName: String + confirmCustomDesign() : boolean

DesignSketch
- DesignSketch : sketchName : String
+ createSketch(sketchName) : List<String> + selectSketch(sketchName) : String + setFinalSketch(sketchName) : String + viewSketch(sketchName) : String + confirmSketch() : boolean

FinalDesign
- FinalDesign : finalDesignName : String - DesignSketch : sketches : List<String>
+ viewSketches(sketches) : List<String> + selectSketchForFinalDesign(sketches) : List<String> + setFinalDesign(finalDesignName) : String + confirmFinalDesign() : boolean

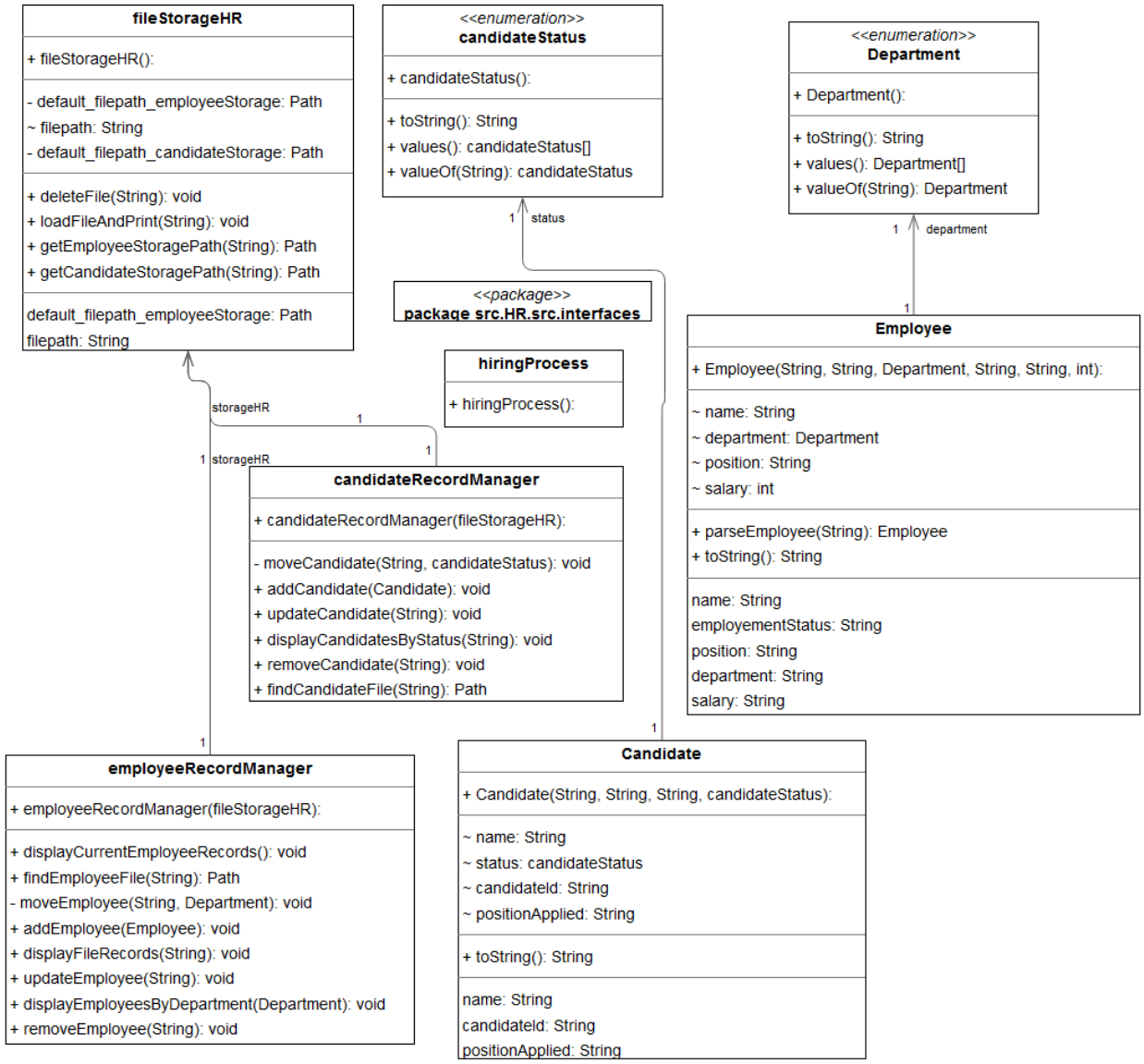
MarketingDesign
- MarketingDesign : marketingDesignName : String - FinalDesign : finalDesignName : String
+ selectFinalDesign(finalDesignName) : String + createMarketingDesignName(marketingDesignName) String + setMarketingDesignName(marketingDesignName) : String + confirmMarketingDesign() : boolean

DesignFileManager
- DesignSketch: sketches : Map<String, Object> - FinalDesign : finalDesign : Map<String, Object> - CustomDesign : customDesign : Map<String, Object> - MarketingDesign : marketingDesign : Map<String, Object>
+ sendDataToRepository() : File + addCustomDesign() : File + addMarketingDesign(): File + addSketches(): File + addFinalDesign(): File

DesignSpecificationsController
- DesignSketch : sketch : Object - FinalDesign : finalDesign : Object - CustomDesign: customDesign : Object - MarketingDesign : marketingDesign : Object
+ setDesignSketch(sketch) : Object + setFinalDesign(finalDesign) : Object + setCustomDesign(customDesign) : Object + setMarketingDesign(marketingDesign) : Object

HeadOfDesignTeam
DesignSketch : sketches : List<String>
+ confirmFinalDesign() : boolean + confirmMarketingDesign() : boolean + confirmSketches() : boolean + viewSketches(sketches) : List<String>

HR [Sam Gumm]


fileStorageHR
+ fileStorageHR():
- default_filepath_employeeStorage: Path
~ filepath: String
- default_filepath_candidateStorage: Path
+ deleteFile(String): void
+ loadFileAndPrint(String): void
+ getEmployeeStoragePath(String): Path
+ getCandidateStoragePath(String): Path
default_filepath_employeeStorage: Path
filepath: String
<<enumeration>>
candidateStatus
+ candidateStatus():
+ toString(): String
+ values(): candidateStatus[]
+ valueOf(String): candidateStatus
<<enumeration>>
Department
+ Department():
+ toString(): String
+ values(): Department[]
+ valueOf(String): Department
1
status
1
department
<<package>>
package src.HR.src.interfaces
hiringProcess
+ hiringProcess():
Employee
+ Employee(String, String, Department, String, String, int):
~ name: String
~ department: Department
~ position: String
~ salary: int
+ parseEmployee(String): Employee
+ toString(): String
name: String
employementStatus: String
position: String
department: String
salary: String
storageHR
1
1
storageHR
candidateRecordManager
+ candidateRecordManager(fileStorageHR):
- moveCandidate(String, candidateStatus): void
+ addCandidate(Candidate): void
+ updateCandidate(String): void
+ displayCandidatesByStatus(String): void
+ removeCandidate(String): void
+ findCandidateFile(String): Path
1
1
employeeRecordManager
+ employeeRecordManager(fileStorageHR):
+ displayCurrentEmployeeRecords(): void
+ findEmployeeFile(String): Path
- moveEmployee(String, Department): void
+ addEmployee(Employee): void
+ displayFileRecords(String): void
+ updateEmployee(String): void
+ displayEmployeesByDepartment(Department): void
+ removeEmployee(String): void
Candidate
+ Candidate(String, String, String, candidateStatus):
~ name: String
~ status: candidateStatus
~ candidateId: String
~ positionApplied: String
+ toString(): String
name: String
candidateId: String
positionApplied: String

Sales


Retailee
name : String
location : String
id : int
Recipt
orderId : int
product :
HashMap<String, Int>
printReceipt (oId: int)
order
product :
HashMap<String, Int>
addProduct (pname; string, quan: int)
SalesManage
order : Map<Int, Map<String, String>>
Retailee : Map<Int, Map<String, String>>
RegisterRetailee (name: String, loc: String)
addOrder (RId: int, item: Map<String, int>)
ReturnOrder (RId: int, item: map<String, int>)
SalesInventory Request
repo : Inventory controller
getAvailableProducts (loc: String)
Change product price (p: int, pname: string)
SalesController
RegisterRetailee (name: String, loc: String)
addOrder (RId: int, item: map<String, int>)
ReturnOrder (RId: int, item: map<String, int>)
checkOrder (Oid: int)
checkRetailee (RId: int)

Security [Kenny]

SecuritySchedulerController
[i = 1 … n] b:= addSecurityPersonnel():boolean [i = 1 … n] b:= deleteSecurityPersonnel(String employeeID):boolean b:= showPendingEvents():boolean b:= showAllEvents():boolean b:= showFreeSecurityEmployees():boolean b:= showAllSecurityEmployees():boolean [i = 1 … n] b:= createSchedule(String requestID):boolean [i = 1 … n] b:= deleteSchedule(String scheduleID):boolean b:= editScheduleAssignments():boolean b:= showAllSchedules():boolean b:= showScheduleByID(String scheduleID):boolean

SecurityScheduler
[i = 1 … n] b:= addSecurityPersonnel():boolean [i = 1 … n] b:= deleteSecurityPersonnel(String employeeID):boolean b:= showPendingEvents():boolean b:= showAllEvents():boolean b:= showFreeSecurityEmployees():boolean b:= showAllSecurityEmployees():boolean [i = 1 … n] b:= createSchedule(String requestID):boolean [i = 1 … n] b:= deleteSchedule(String scheduleID):boolean b:= editScheduleAssignments():boolean b:= showAllSchedules():boolean b:= showScheduleByID(String scheduleID):boolean

AuditSchedulerController
[i = 1 … n] b:= addAuditPersonnel():boolean [i = 1 … n] b:= deleteAuditPersonnel(String employeeID):boolean b:= showOngoingAudits():boolean b:= showPassedAudits():boolean b:= showFailedAudits():boolean b:= showAllAudits():boolean b:= showAuditByID(String auditID):boolean

b:= showFreeAuditEmployees():boolean b:= showAllAuditEmployees():boolean [i = 1 … n] b:= createAudit():boolean [i = 1 … n] b:= deleteAudit(String auditID):boolean [i = 1 … n] b:= editAudit(String auditID):boolean [i = 1 … n] b:= sendFailedAudit(String auditID):boolean

AuditScheduler
[i = 1 … n] b:= addAuditPersonnel():boolean [i = 1 … n] b:= deleteAuditPersonnel(String employeeID):boolean b:= showOngoingAudits():boolean b:= showPassedAudits():boolean b:= showFailedAudits():boolean b:= showAllAudits():boolean b:= showAuditByID(String auditID):boolean b:= showFreeAuditEmployees():boolean b:= showAllAuditEmployees():boolean [i = 1 … n] b:= createAudit():boolean [i = 1 … n] b:= deleteAudit(String auditID):boolean [i = 1 … n] b:= editAudit(String auditID):boolean [i = 1 … n] b:= sendFailedAudit(String auditID):boolean

AuditEmployee/SecurityEmployee
employeeID:String name:String division:String position:String department:String expertise:String rating:String yearsOfExperience:String previousAssignment:String currentAssignment:String

SecurityRequests
requestID:String priorityLevel:String

department:String location:String description:String duration:String tasks:String specialReqs:String dateIssued:String resolved:String fileName:String

SecuritySchedules
scheduleID requestID priorityLevel department location description duration tasks dateScheduled

AuditSchedules
scheduleID priorityLevel department location description duration tasks status dateScheduled

Interface List

Modeling [Mia]: (updated)

```
public interface IHOD extends ITeamMember {
    Event createEvent(Boolean type, String celebrity,
String collab);
    Event createEvent(Boolean type, int modelID, String
collab);
    ArrayList<Event> getEvents();
    ArrayList<Fitting> getFittings();
    void addManager(Manager manager);
    Manager getManager(Team team);
    void requestFitting(Team team, TeamMember model,
String garment, LocalDateTime date);
}
```

```
public interface IEvent {
    int getId();
    String getType();
    String getCelebrity();
    String getCollab();
    String getCompletion();
//    void endEvent();
    Map<String, String> toMap();
}
```

```
public interface ITeamMember {
    public int getId();
    public Map<String, String> toMap();
}
```

```
public interface IManager extends ITeamMember {
    Team getTeam();

    ArrayList<TeamMember> getTeamMembers();
    void addTeamMember(TeamMember member);

    Fitting scheduleFitting(TeamMember model, String
garment, LocalDateTime date);
}
```

```
public interface IFitting {
    int getId();
    TeamMember getModel();
    String getGarment();
    String getCompletion();
    Map<String, String> toMap();
//    void endFitting();
}
```

Marketing [Mia]:

```
public interface IAdvertisement {
    int getId();
    AdvertType getAdvertType();
    Team[] getAssociatedTeams();
    void changeAdvertType(AdvertType type);
    void addPhotoshoot(int modelId);
    Map<String, String> toMap();
}
```

```
public interface IHOD extends ITeamMember{
    ArrayList<EventAdvertisement> getEventAdverts();
    ArrayList<DesignAdvertisement>
getDesignAdverts();
    EventAdvertisement createEventAdvert(Event event,
AdvertType type);
    DesignAdvertisement
createDesignAdvert(AdvertType type, String notes);
    Map<String, String> toMap();
    void addManager(Manager manager);
    Manager getManager(Team team);
    Event requestPhotoshoot(int modelId);
}
```

```
public interface IManager extends ITeamMember{
    Team getTeam();
    ArrayList<TeamMember> getTeamMembers();
    void addTeamMember(TeamMember member);

    Map<String, String> toMap();
}
```

```
public interface ITeamMember {
    int getId();
    Employee getEmployeeInfo();
    Map<String, String> toMap();
}
```

HR [Sam]:

```
package src.HR.src.interfaces;

public interface ICandidate {
        String getName();
        String getCandidateId();
        String getPositionApplied();
        String getStatus();
        void setStatus(String status);

        @Override
        String toString();
}
```

```
package src.HR.src.interfaces;

public interface IEmployee {
        String getName();
        String getEmployeeID();
        String getDepartment();
        String getPosition();
        String getEmployementStatus();
        String getSalary();

        @Override
        String toString();
}
```

Security [Kenny]

```
public interface AuditScheduler {
   boolean addAuditPersonnel();
   boolean deleteAuditPersonnel(String employeeID);
   boolean showOngoingAudits();
   boolean showPassedAudits();
   boolean showFailedAudits();
   boolean showAllAudits();
   boolean showAuditByID(String auditID);
   boolean showFreeAuditEmployees();
   boolean showAllAuditEmployees();
   boolean createAudit();
   boolean deleteAudit(String auditID);
   boolean editAudit(String auditID);
   boolean sendFailedAudit(String auditID);
}

public interface SecurityScheduler {

   boolean addSecurityPersonnel();
   boolean deleteSecurityPersonnel(String
employeeID);
   boolean showPendingEvents();
   boolean showAllEvents();
   boolean showFreeSecurityEmployees();
   boolean showAllSecurityEmployees();
   boolean createSchedule(String requestID);
   boolean deleteSchedule(String scheduleID);
```

```
public interface AuditSchedulerController {
   boolean addAuditPersonnel();
   boolean deleteAuditPersonnel(String employeeID);
   boolean showOngoingAudits();
   boolean showPassedAudits();
   boolean showFailedAudits();
   boolean showAllAudits();
   boolean showAuditByID(String auditID);
   boolean showFreeAuditEmployees();
   boolean showAllAuditEmployees();
   boolean createAudit();
   boolean deleteAudit(String auditID);
   boolean editAudit(String auditID);
   boolean sendFailedAudit(String auditID);
}

public interface SecuritySchedulerController {

   boolean addSecurityPersonnel();
   boolean deleteSecurityPersonnel(String
employeeID);
   boolean showPendingEvents();
   boolean showAllEvents();
   boolean showFreeSecurityEmployees();
   boolean showAllSecurityEmployees();
   boolean createSchedule(String requestID);
   boolean deleteSchedule(String scheduleID);
```

```
    boolean editScheduleAssignments(String
  scheduleID);
    boolean showAllSchedules();
    boolean showScheduleByID(String ScheduleID);
  }
```

```
    boolean editScheduleAssignments(String
scheduleID);
    boolean showAllSchedules();
    boolean showScheduleByID(String ScheduleID);
}
```

Sales [Mani Raj]

```
public interface SalesManagement {

    public void addOrder(int rid, Map<String, Integer> products);

    public void registerRetailer(String name, String retailerLocation);

    public void returnOrder(int rid, int oid, Map<String, Integer> rproducts, int days);

    public void printReceipt(int orderId);

    public void viewRetailers();

}

public interface SalesController {

    public void run();
    public void registerRetailer(String name, String location);
    public void addOrder(int rid, Map<String, Integer> products);
    public void printReceipt(int orderId);
    public void returnOrder(int rid, int oid, Map<String, Integer> rproducts, int days);
    public void viewRetailers();
    public void viewAvaProducts();
    public void viewRegisteredProducts();
    public void changePrice(int price, String pname);
}

public interface Retailer {

            Map<String,String> getRDetails();
            String getName();
            String getLocation();
void print();
}
```

Design [Doyle Chism]

```
    public interface DesignSpecifications {
     void setColor(List<String> colors);
     void setRawMaterials(List<String> rawMaterials);
     void setSizes(List<String> sizes);
     void setQuantities(int quantities);
     void setDesignName(String designName);
     void setDesignImage(String image);
     List<String> getColors();
     List<String> getRawMaterials();
     List<String> getSizes();
     int getQuantities();
     String getDesignName();
     String getDesignImage();
```

```
    public interface HeadOfDesignInterface {
    void viewSketches(List<DesignSketch> sketches);
    void selectSketch(int sketchIndex, List<DesignSketch>
sketchList);
    CustomDesign confirmCustomDesign();
    FinalDesign confirmFinalDesign();
    MarketingDesign confirmMarketingDesign();
}
public interface MarketingDesignSpecifications {
    String getDesignSketchName();
    void setDesignSketchName(String designSketchName);
    void setTargetAudience(String targetAudience);
    String getTargetAudience();
```

```
    String displayAllSpecifications();
    Map<String, Object> mapObjects()
}
```

```
    void setPrice(String price);
    String getPrice();
    void setSeasonType(String seasonType); //fall,spring...
    String getSeasonType();
    void setProductDescription(String productDescription);
    String getProductDescription();
    String displayAllSpecifications();
    Map<String, Object> mapObjects();
}
```

Manufacturing [Doyle Chism]

```
 public interface HeadOfManufacturingInterface {

    void viewRawMaterials(Map<String, Integer>
rawMaterials);
    void selectRawMaterial(int index, String
rawMaterials);
    void receiveFinalDesign(Map<String, String>
finalDesign);
    void receiveCustomDesign(Map<String, String>
customDesign);
    void verifyProduct(Product product);
    void verifyCustomProduct(CustomProduct
customProduct);

}
public interface ProductInterface {

    void setName(String name);
    String getName();

    void setDescription(String description);
    String getDescription();

    void setQuantity(int quantity);
    int getQuantity();

    void setCategory(String category);
    String getCategory();

}
```

```
    public interface MachineOperations {


    void startMachine();
    boolean isRunning();
}
public interface ManagerInterface {

    void collectRawMaterials(Map<String, Integer>
materials);
    Map<String, Integer> getCollectedMaterials();
    boolean createProduct(Map<String, Integer>
materials);
   void deliverProduct(Product product);

}
```

Implementation

Modeling [Mia]: (updated)

```
public class TeamMember implements ITeamMember {
    static int nextid = 0;
    int id;
    Employee employeeInfo;
    Team team;

    TeamMember(int i, Employee employee, Team team) {
        id = i;
        if (id > nextid) nextid = id++;
        employeeInfo = employee;
        this.team = team;
    }

    TeamMember(Employee employeeInfo, Team team) {
        id = nextid;
        nextid++;
        this.employeeInfo = employeeInfo;
        this.team = team;
    }

    public int getId() {
        return id;
    }

    public String toString() {
        String str = team.toString() + ": " + employeeInfo.toString();
        return str;
    }

    public static TeamMember parse(Map<String, String> member) {
        return new TeamMember(
            Integer.parseInt(member.get("id")),
            Employee.parseEmployee(member.get("employeeInfo")),
            Team.parseTeam(member.get("team"))
        );

    }

    public Map<String, String> toMap() {
        Map<String, String> memberDetails = new HashMap<>();
        memberDetails.put("id", Integer.toString(this.id));
        memberDetails.put("employeeInfo", this.employeeInfo.toString());
        memberDetails.put("team", this.team.toString());
        return memberDetails;
    }
}
```

```
public class HOD implements IHOD {

    private final Employee employeeInfo;
    private final ArrayList<Manager> managers;

    public ArrayList<Event> events;
    public ArrayList<Fitting> fittings;

    public HOD(Employee employeeInfo, ArrayList<Manager>
managers) {
        this.employeeInfo = employeeInfo;
        this.managers = managers;

        events = ModelingDepartment.fileManager.getEvents();
        fittings = ModelingDepartment.fileManager.getFittings();
    }

    public HOD() {
        employeeInfo = new Employee("hod", "Head of Modeling",
Department.MODELING, "HOD", "Employeed", 100000);
        managers = new ArrayList<>();
```

```
public class Manager implements IManager {
    static int nextid = 0;
    int id;

    private final Employee employeeInfo;
    private final ArrayList<TeamMember> teamMembers;
    private final Team team;

    public Manager(String name, Team team) {
        id = nextid;
        nextid++;
        employeeInfo =
App.hrDepartment.getEmployee(Department.MODELING, name);
        teamMembers =
ModelingDepartment.fileManager.getTeamMembers(team);
        this.team = team;
    }

    public Manager(int id, Employee employeeInfo, Team team) {
        this.id = id;
        if(id>nextid) nextid = id+1;
        this.employeeInfo = employeeInfo;
        teamMembers =
ModelingDepartment.fileManager.getTeamMembers(team);
        this.team = team;
    }

    public Manager(Team team) {
        id = nextid;
        nextid++;
        employeeInfo = new Employee("manager", team.toString() + "
Manager", Department.MODELING, "Manager", "Employeed", 100000);
        teamMembers = new ArrayList<>();
        for(int i = 0; i <= 3; i++) {
            teamMembers.add(new TeamMember(team));
        }
        this.team = team;

        ModelingDepartment.fileManager.addManager(this);
    }

    @Override
    public int getId() {
        return id;
    }

    @Override
    public Team getTeam() {
        return team;
    }

    @Override
    public ArrayList<TeamMember> getTeamMembers() {
        return teamMembers;
    }

    @Override
    public void addTeamMember(TeamMember member) {
        teamMembers.add(member);
        ModelingDepartment.fileManager.addTeamMember(member);
    }

    @Override
    public Fitting scheduleFitting(TeamMember model, String garment,
LocalDateTime date) {
        return new Fitting(model, garment, date);
    }

    @Override
    public Map<String, String> toMap() {
```

```
        managers.add(new Manager(Team.MODELING));
        managers.add(new Manager(Team.MAKEUP));
        managers.add(new Manager(Team.CLOTHING));

        ModelingDepartment.fileManager.addHOD(this);
    }

    @Override
    public void createEvent(Boolean type, String celebrity, String collab) {
        ArrayList<TeamMember> teamMembers = new ArrayList<>();

        for(Manager manager: managers) {
            ArrayList<TeamMember> team = manager.getTeamMembers();
            team.addAll(manager.getTeamMembers());
        }

        Event event = new Event(teamMembers, type, celebrity, collab);

        events.add(event);
        System.out.println(event);
        ModelingDepartment.fileManager.addEvent(event);
    }

    @Override
    public int getId() {
        return 0;
    }

    @Override
    public Map<String, String> toMap() {
        Map<String, String> memberDetails = new HashMap<>();
        memberDetails.put("employeeInfo", this.employeeInfo.toString());
        Integer[] tmp = new Integer[managers.size()];
        for(int i = 0; i < managers.size(); i++) {
            tmp[i] = managers.get(i).getId();
        }
        memberDetails.put("managers", Arrays.toString(tmp));
        return memberDetails;
    }

    @Override
    public void addManager(Manager manager) {
        managers.add(manager);
        ModelingDepartment.fileManager.addManager(manager);
    }

    @Override
    public Manager getManager(Team team) {
        for (Manager manager: managers) {
            if (manager.getTeam() == team) {
                return manager;
            }
        }
        return null;
    }

    @Override
    public String toString() {
        StringBuilder str = new StringBuilder("\nHOD: ");
        str.append("\nEmployeeInfo:
").append(this.employeeInfo.toString());
        str.append("\nManagers: ");
        for (Manager manager : managers) {
            str.append("\n  ").append(manager.toString());
        }
        return str.toString();
    }

    @Override
    public void requestFitting(Team team, TeamMember model, String
garment, LocalDateTime date) {
        Fitting fitting = getManager(team).scheduleFitting(model, garment,
date);
        fittings.add(fitting);
        System.out.println(fitting.toString());
        ModelingDepartment.fileManager.addFitting(fitting);
    }

    public static HOD parse(Map<String, String> hod) {
        Map<String, String> memberDetails = new HashMap<>();
        memberDetails.put("id", Integer.toString(this.id));
        memberDetails.put("employeeInfo", this.employeeInfo.toString());
        Integer[] tmp = new Integer[teamMembers.size()];
        for(int i = 0; i < teamMembers.size(); i++) {
            tmp[i] = teamMembers.get(i).getId();
        }
        memberDetails.put("teamMembers", Arrays.toString(tmp));
        memberDetails.put("team", this.team.toString());
        return memberDetails;
    }

    @Override
    public String toString() {
        StringBuilder str = new StringBuilder("\n" + this.team.toString() + "
Manager: " + id);
        str.append("\nEmployeeInfo:
").append(this.employeeInfo.toString());
        str.append("\nTeam Members: ");
        for (TeamMember teamMember : teamMembers) {
            str.append("\n").append(teamMember.toString());
        }
        return str.toString();
    }

    public static Manager parse(Map<String, String> manager) {
        return new Manager(
            Integer.parseInt(manager.get("id")),
            Employee.parseEmployee(manager.get("employeeInfo")),
            Team.parseTeam(manager.get("team"))
        );
    }
}
```

```
public class Fitting implements IFitting {
    static int nextid = 0;
    int id;
    TeamMember model;
    String garment;
    LocalDateTime date;
    Boolean completionStatus;

    Fitting (int id, TeamMember model, String garment, LocalDateTime
date, Boolean completionStatus) {
        this.id = id;
        if (id > nextid) nextid = id+1;
        this.model = model;
        this.garment = garment;
        this.date = date;
        this.completionStatus = completionStatus;
    }

    Fitting (TeamMember model, String garment, LocalDateTime date) {
        id = nextid;
        nextid++;
        this.model = model;
        this.garment = garment;
        this.date = date;
        this.completionStatus = false;
    }


    @Override
    public int getId() {
        return id;
    }

    @Override
    public TeamMember getModel() {
        return model;
    }

    @Override
    public String getGarment() {
        return this.garment;
    }

    @Override
```

```
        return new HOD(
                Employee.parseEmployee(hod.get("employeeInfo")),
                ModelingDepartment.fileManager.getManagers()
        );
    }
}
```

```
public class FileManager {
    private Map<String, Map<String, String>> events = new
HashMap<>();
    private Map<String, Map<String, String>> teamMembers = new
HashMap<>();
    private Map<String, Map<String, String>> fittings = new
HashMap<>();

    private final PoorTextEditor editor = new PoorTextEditor();

    private final String repo = "src/Modeling/repository/";

    public FileManager() {
        fillRepos();
    }

    public void fillRepos() {
        File f = new File(repo + "events.txt");
        if(f.exists()) {
            editor.processTextFile(repo + "events.txt");
            events = editor.getRepositoryStringMap();
        }

        f = new File(repo + "fittings.txt");
        if(f.exists()) {
            editor.processTextFile(repo + "fittings.txt");
            fittings = editor.getRepositoryStringMap();
        }

        f = new File(repo + "department.txt");
        if(f.exists()) {
            editor.processTextFile(repo + "department.txt");
            teamMembers = editor.getRepositoryStringMap();
        }
    }

//    Events:
    public void addEvent(Event event) {
        events.put("Event " + event.getId(), event.toMap());

        editor.setRepositoryStrings(events);
        editor.writeToTextFile(repo + "events.txt");
    }

    public ArrayList<Event> getEvents() {
        ArrayList<Event> tmp = new ArrayList<>();
        for(Map<String, String> event: events.values()) {
            tmp.add(Event.parse(event));
        }
        return tmp;
    }

    public ArrayList<Fitting> getFittings() {
        ArrayList<Fitting> tmp = new ArrayList<>();
        for(Map<String, String> event: fittings.values()) {
            tmp.add(Fitting.parse(event));
        }
        return tmp;
    }

//    TeamMembers:
    public void addTeamMember(TeamMember teamMember) {
        teamMembers.put("Team Member "+ teamMember.getId(),
teamMember.toMap());

        editor.setRepositoryStrings(teamMembers);
        editor.writeToTextFile(repo + "department.txt");
    }

    public void addManager(Manager manager) {
```

```
    public String getCompletion() {
        if(completionStatus) return "true";
        return "false";
    }

//    @Override
//    public void endFitting() {
//        this.completionStatus = true;
//        System.out.println(this.toString());
//    }

    @Override
    public String toString() {
        String str = "\nFitting: ";
        str += "\nModel: " + this.model.toString();
        str += "\nGarment: " + this.garment +
                "\nDate: " + this.date +
                "\nCompletion Status: " + this.completionStatus;
        return str;
    }

    @Override
    public Map<String, String> toMap() {
        Map<String, String> fittingDetails = new HashMap<>();
        fittingDetails.put("id", Integer.toString(this.id));
        fittingDetails.put("model", Integer.toString(model.getId()));
        fittingDetails.put("garment", garment);
        fittingDetails.put("date", date.toString());
        fittingDetails.put("completionStatus", this.getCompletion());
        return fittingDetails;
    }

    public static Fitting parse(Map<String, String> fitting) {
        return new Fitting(
                Integer.parseInt(fitting.get("id")),

ModelingDepartment.fileManager.getModel(Integer.parseInt(fitting.get("
model"))),
                fitting.get("garment"),
                LocalDateTime.parse(fitting.get("date")),
                Boolean.parseBoolean(fitting.get("completionStatus"))
        );
    }
}
```

```
public class Event implements IEvent {
    static int nextid = 0;
    int id;
    ArrayList<TeamMember> teamMembers;
    Boolean type; //true for photoshoot, false for fashion show
    String celebrity;
    String collab;
    Boolean completionStatus;

    Event(ArrayList<TeamMember> teamMembers, Boolean type, String
celebrity, String collab) {
        id = nextid;
        nextid++;
        this.teamMembers = teamMembers;
        this.type = type;
        this.celebrity = celebrity;
        this.collab = collab;
        this.completionStatus = false;
    }

    Event(int id, ArrayList<TeamMember> teamMembers, Boolean type,
String celebrity, String collab, Boolean completionStatus) {
        this.id = id;
        if (id > nextid) nextid = id+1;
        this.teamMembers = teamMembers;
        this.type = type;
        this.celebrity = celebrity;
        this.collab = collab;
        this.completionStatus = completionStatus;
    }

    @Override
    public int getId() {
```

```
        teamMembers.put(manager.getTeam().toString() + " Manager",
manager.toMap());

        editor.setRepositoryStrings(teamMembers);
        editor.writeToTextFile(repo + "department.txt");
    }

    public void addHOD(HOD hod) {
        teamMembers.put("HOD", hod.toMap());

        editor.setRepositoryStrings(teamMembers);
        editor.writeToTextFile(repo + "department.txt");
    }

    public ArrayList<Manager> getManagers() {
        ArrayList<Manager> tmp = new ArrayList<>();
        for(Map<String, String> member: teamMembers.values()) {
            if(member.containsKey("teamMembers")) {
                tmp.add(Manager.parse(member));
            }
        }
        return tmp;
    }

    public ArrayList<TeamMember> getTeamMembers(Team team) {
        ArrayList<TeamMember> tmp = new ArrayList<>();
        for(Map<String, String> member: teamMembers.values()) {
            if (member.containsValue(team.toString()) &&
!member.containsKey("teamMembers") &&
!member.containsKey("managers")) {
                tmp.add(TeamMember.parse(member));
            }
        }
        return tmp;
    }

    public ArrayList<TeamMember> getTeamMembers() {
        ArrayList<TeamMember> tmp = new ArrayList<>();
        for(Map<String, String> member: teamMembers.values()) {
            if (!member.containsKey("teamMembers") &&
!member.containsKey("managers")) {
                tmp.add(TeamMember.parse(member));
            }
        }
        return tmp;
    }

    public HOD getHOD() {
        return HOD.parse(teamMembers.get("HOD"));
    }

    public void addFitting(Fitting fitting) {
        fittings.put("Fitting " + fitting.getId(), fitting.toMap());

        editor.setRepositoryStrings(fittings);
        editor.writeToTextFile(repo + "fittings.txt");
    }

    public TeamMember getModel(int model) {
        return TeamMember.parse(teamMembers.get("Team Member "+
model));
    }
}
```

```
        return id;
    }

    @Override
    public String getType() {
        if(this.type) return "Photoshoot";
        return "Fashion Show";
    }

    @Override
    public String getCelebrity() {
        return this.celebrity;
    }

    @Override
    public String getCollab() {
        return this.collab;
    }

    @Override
    public String getCompletion() {
        if(completionStatus) return "true";
        return "false";
    }

//    @Override
//    public void endEvent() {
//        this.completionStatus = true;
//        System.out.println(this.toString());
//    }

    @Override
    public String toString() {
        StringBuilder str = new StringBuilder("\nEvent: " + id);
        if(this.type) {
            str.append("\nPhotoshoot");
        } else {
            str.append("\nFashion show");
        }
        str.append("\nTeam Members: ");

        for (TeamMember teamMember : teamMembers) {
            str.append("\n").append(teamMember.toString());
        }
        str.append("\nCelebrity: ").append(this.celebrity).append("\nCollab:
").append(this.collab).append("\nCompletion Status:
").append(this.completionStatus);
        return str.toString();
    }

    @Override
    public Map<String, String> toMap() {
            Map<String, String> eventDetails = new HashMap<>();
            eventDetails.put("id", Integer.toString(this.id));
            eventDetails.put("type", this.getType());
            Integer[] tmp = new Integer[teamMembers.size()];
            for(int i = 0; i < teamMembers.size(); i++) {
                tmp[i] = teamMembers.get(i).getId();
            }
            eventDetails.put("teamMembers", Arrays.toString(tmp));
            eventDetails.put("celebrity", this.getCelebrity());
            eventDetails.put("collab", this.getCollab());
            eventDetails.put("completionStatus", this.getCompletion());
            return eventDetails;
    }

    public static Event parse(Map<String, String> event) {
        return new Event(
            Integer.parseInt(event.get("id")),
            ModelingDepartment.fileManager.getTeamMembers(),
            event.get("type").equals("Photoshoot"),
            event.get("celebrity"),
            event.get("collab"),
            Boolean.parseBoolean(event.get("completionStatus"))
        );
    }
}
```

Marketing [Mia]:

```
public enum AdvertType {
    BILLBOARD,
    MAGAZINE,
    SOCIALMEDIA,
    COMMERCIAL;

    public String toString() {
        return switch (this) {
            case BILLBOARD -> "Billboard";
            case MAGAZINE -> "Magazine";
            case SOCIALMEDIA -> "Social-Media";
            case COMMERCIAL -> "Commercial";
        };
    }

    public static AdvertType parseType(String team) {
        return switch (team) {
            case "Billboard" -> BILLBOARD;
            case "Magazine" -> MAGAZINE;
            case "Social-Media" -> SOCIALMEDIA;
            case "Commercial" -> COMMERCIAL;
            default -> throw new IllegalArgumentException();
        };
    }
}
```

```
public enum Team {
    EDITING,
    SOCIALMEDIA,
    VIDEO,
    DISTRIBUTION;

    public String toString() {
        return switch (this) {
            case EDITING -> "Editing";
            case SOCIALMEDIA -> "Social-Media";
            case VIDEO -> "Video";
            case DISTRIBUTION -> "Distribution";
        };
    }

    public static Team parseTeam(String team) {
        return switch (team) {
            case "Editing" -> EDITING;
            case "Social-Media" -> SOCIALMEDIA;
            case "Video" -> VIDEO;
            case "Distribution" -> DISTRIBUTION;
            default -> throw new IllegalArgumentException();
        };
    }
}
```

```
public class DesignAdvertisement implements IAdvertisement {
    static int nextid = 0;
    int id;
    FinalDesign design;
    AdvertType advertType;
    Team[] associatedTeams;
    String notes;

    Event photoshoot;

    DesignAdvertisement(AdvertType advertType, String notes) {
        id = nextid;
        nextid++;
        assignTeams(advertType);
        this.advertType = advertType;
        this.design = getDesign();
        this.notes = notes;
    }

    DesignAdvertisement(int i, AdvertType advertType, String notes) {
        id = i;
        if (id >= nextid) nextid = id + 1;
        assignTeams(advertType);
        this.advertType = advertType;
        this.design = getDesign();
        this.notes = notes;
    }

    DesignAdvertisement(int i, AdvertType advertType, Event
photoshoot, String notes) {
        id = i;
        if (id >= nextid) nextid = id + 1;
        assignTeams(advertType);
        this.advertType = advertType;
        this.design = getDesign();
        this.photoshoot = photoshoot;
        this.notes = notes;
    }

    private void assignTeams(AdvertType advertType) {
        switch (advertType) {
            case BILLBOARD, MAGAZINE -> associatedTeams = new
Team[]{Team.EDITING, Team.DISTRIBUTION};
            case SOCIALMEDIA -> associatedTeams = new
```

```
public class EventAdvertisement implements IAdvertisement {
    static int nextid = 0;
    int id;
    Event event;
    AdvertType advertType;
    Team[] associatedTeams;

    Event modelEvent;

    EventAdvertisement(Event event, AdvertType advertType) {
        id = nextid;
        nextid++;
        assignTeams(advertType);
        this.advertType = advertType;
        this.event = event;
    }

    EventAdvertisement(int i, Event event, AdvertType advertType) {
        id = i;
        if (id >= nextid) nextid = id + 1;
        assignTeams(advertType);
        this.advertType = advertType;
        this.event = event;
    }

    EventAdvertisement(int i, Event event, AdvertType advertType, Event
modelEvent) {
        id = i;
        if (id >= nextid) nextid = id + 1;
        assignTeams(advertType);
        this.advertType = advertType;
        this.event = event;
        this.modelEvent = modelEvent;
    }

    private void assignTeams(AdvertType advertType) {
        switch (advertType) {
            case BILLBOARD, MAGAZINE -> associatedTeams = new
Team[]{Team.EDITING, Team.DISTRIBUTION};
            case SOCIALMEDIA -> associatedTeams = new
Team[]{Team.SOCIALMEDIA, Team.EDITING};
            case COMMERCIAL -> associatedTeams = new
Team[]{Team.EDITING, Team.VIDEO};
        }
```

```
Team[]{Team.SOCIALMEDIA, Team.EDITING};
            case COMMERCIAL -> associatedTeams = new
Team[]{Team.EDITING, Team.VIDEO};
        }
    }

    @Override
    public int getId() {
        return id;
    }

    @Override
    public AdvertType getAdvertType() {
        return advertType;
    }

    @Override
    public Team[] getAssociatedTeams() {
        return associatedTeams;
    }

    public String getNotes() {
        return notes;
    }

    @Override
    public void changeAdvertType(AdvertType type) {
        advertType = type;
        assignTeams(type);
    }

    @Override
    public void addPhotoshoot(int modelId) {
        photoshoot =
MarketingDepartment.hod.requestPhotoshoot(modelId);
    }

    @Override
    public String toString() {
        String str =  "\nEvent Advertisement " + id + ":" +
                "\n Design: " + design.getDesignName() +
                "\n AdvertType: " + advertType.toString() +
                "\n Associated Teams: " + associatedTeams[0].toString() + ", "
+ associatedTeams[1].toString() +
                "\n Notes: " + notes;
        if(photoshoot != null) {
            str += "\n Modeling Event: " + photoshoot.getId();
        }

        return str;
    }

    @Override
    public Map<String, String> toMap() {
        Map<String, String> advertDetails = new HashMap<>();
        advertDetails.put("id", Integer.toString(this.id));
        advertDetails.put("design", design.getDesignName());
        advertDetails.put("advertType", this.getAdvertType().toString());
        if(photoshoot != null) {
            advertDetails.put("photoshoot",
Integer.toString(photoshoot.getId()));
        }
        String[] tmp = new String[associatedTeams.length];
        for(int i = 0; i < associatedTeams.length; i++) {
            tmp[i] = associatedTeams[i].toString();
        }
        advertDetails.put("associatedTeams", Arrays.toString(tmp));
        advertDetails.put("notes", notes);
        return advertDetails;
    }

    private FinalDesign getDesign() {
        FinalDesign d = new FinalDesign("Ball Gown");
        ArrayList<String> tmp = new ArrayList<>();
        tmp.add("Red");
        tmp.add("Green");
        d.setColor(tmp);
        tmp.clear();
        tmp.add("Small");
    }

    @Override
    public int getId() {return id;}
    @Override
    public AdvertType getAdvertType() {return advertType;}
    @Override
    public Team[] getAssociatedTeams() {return associatedTeams;}
    @Override
    public void changeAdvertType(AdvertType type) {
        this.advertType = type;
        assignTeams(type);
    }

    @Override
    public void addPhotoshoot(int modelId) {
        modelEvent =
MarketingDepartment.hod.requestPhotoshoot(modelId);
    }

    @Override
    public String toString() {
        String str =  "\nEvent Advertisement " + id + ":" +
                "\n Event: " + event.getId() +
                "\n AdvertType: " + advertType.toString() +
                "\n Associated Teams: " + associatedTeams[0].toString() + ", "
+ associatedTeams[1].toString();
        if(modelEvent != null) {
            str += "\n Modeling Event: " + modelEvent.getId();
        }

        return str;
    }

    @Override
    public Map<String, String> toMap() {
        Map<String, String> advertDetails = new HashMap<>();
        advertDetails.put("id", Integer.toString(this.id));
        advertDetails.put("event", Integer.toString(event.getId()));
        advertDetails.put("advertType", this.getAdvertType().toString());
        if(modelEvent != null) {
            advertDetails.put("modelEvent",
Integer.toString(modelEvent.getId()));
        }
        String[] tmp = new String[associatedTeams.length];
        for(int i = 0; i < associatedTeams.length; i++) {
            tmp[i] = associatedTeams[i].toString();
        }
        advertDetails.put("associatedTeams", Arrays.toString(tmp));
        return advertDetails;
    }

    public static EventAdvertisement parse(Map<String, String> advert) {
        if(advert.containsKey("modelEvent")) {
            return new EventAdvertisement(
                    Integer.parseInt(advert.get("id")),

App.modelingDepartment.getEvent(Integer.parseInt(advert.get("event"))
),
                    AdvertType.parseType(advert.get("advertType")),

App.modelingDepartment.getEvent(Integer.parseInt(advert.get("modelE
vent")))
                );
        }
        return new EventAdvertisement(
                Integer.parseInt(advert.get("id")),

App.modelingDepartment.getEvent(Integer.parseInt(advert.get("event"))
),
                AdvertType.parseType(advert.get("advertType"))
        );
    }
}
```

```
public class FileManager {
    private Map<String, Map<String, String>> teamMembers = new
HashMap<>();
```

```
        tmp.add("Medium");
        d.setSizes(tmp);
        d.setQuantities(5);
        return d;
    }

    public static DesignAdvertisement parse(Map<String, String> advert)
{
        if(advert.containsKey("photoshoot")) {
            return new DesignAdvertisement(
                    Integer.parseInt(advert.get("id")),
                    AdvertType.parseType(advert.get("advertType")),

App.modelingDepartment.getEvent(Integer.parseInt(advert.get("photosh
oot"))),
                    advert.get("notes")
            );
        }
        return new DesignAdvertisement(
                Integer.parseInt(advert.get("id")),
                AdvertType.parseType(advert.get("advertType")),
                advert.get("notes")
        );
    }
}
```

```
public class HOD implements IHOD {

    private final Employee employeeInfo;
    private final ArrayList<Manager> managers;

    public ArrayList<EventAdvertisement> eventAdverts;

    public ArrayList<DesignAdvertisement> designAdverts;


    public HOD(Employee employeeInfo, ArrayList<Manager>
managers) {
        this.employeeInfo = employeeInfo;
        this.managers = managers;

        eventAdverts =
MarketingDepartment.fileManager.getEventAdverts();
        designAdverts =
MarketingDepartment.fileManager.getDesignAdverts();
    }

    public HOD() {
        employeeInfo = new Employee("hod", "Head of Marketing",
Department.MARKETING, "HOD", "Employeed", 100000);
        managers = new ArrayList<>();
        managers.add(new Manager(Team.EDITING));
        managers.add(new Manager(Team.SOCIALMEDIA));
        managers.add(new Manager(Team.VIDEO));
        managers.add(new Manager(Team.DISTRIBUTION));

        MarketingDepartment.fileManager.addHOD(this);
    }

    @Override
    public int getId() {
        return 0;
    }

    @Override
    public Employee getEmployeeInfo() {
        return employeeInfo;
    }

    @Override
    public ArrayList<EventAdvertisement> getEventAdverts() {return
eventAdverts;}

    @Override
    public ArrayList<DesignAdvertisement> getDesignAdverts() {return
designAdverts;}

    @Override
```

```
    private Map<String, Map<String, String>> eventAdverts = new
HashMap<>();
    private Map<String, Map<String, String>> designAdverts = new
HashMap<>();

    private final PoorTextEditor editor = new PoorTextEditor();

    private final String repo = "src/Marketing/repository/";

    public FileManager() {
        fillRepos();
    }

    public void fillRepos() {
        File f = new File(repo + "department.txt");
        if(f.exists()) {
            editor.processTextFile(repo + "department.txt");
            teamMembers = editor.getRepositoryStringMap();
        }

        f = new File(repo + "eventAdvert.txt");
        if(f.exists()) {
            editor.processTextFile(repo + "eventAdvert.txt");
            eventAdverts = editor.getRepositoryStringMap();
        }

        f = new File(repo + "designAdvert.txt");
        if(f.exists()) {
            editor.processTextFile(repo + "designAdvert.txt");
            designAdverts = editor.getRepositoryStringMap();
        }
    }

    //    EventsAdverts:
    public void addEventAdvert(EventAdvertisement advert) {
        eventAdverts.put("Advert " + advert.getId(), advert.toMap());

        editor.setRepositoryStrings(eventAdverts);
        editor.writeToTextFile(repo + "eventAdvert.txt");
    }

    public ArrayList<EventAdvertisement> getEventAdverts() {
        ArrayList<EventAdvertisement> tmp = new ArrayList<>();
        for(Map<String, String> advert: eventAdverts.values()) {
            tmp.add(EventAdvertisement.parse(advert));
        }
        return tmp;
    }



    //    EventsAdverts:
    public void addDesignAdvert(DesignAdvertisement advert) {
        designAdverts.put("Advert " + advert.getId(), advert.toMap());

        editor.setRepositoryStrings(designAdverts);
        editor.writeToTextFile(repo + "designAdvert.txt");
    }

    public ArrayList<DesignAdvertisement> getDesignAdverts() {
        ArrayList<DesignAdvertisement> tmp = new ArrayList<>();
        for(Map<String, String> advert: designAdverts.values()) {
            tmp.add(DesignAdvertisement.parse(advert));
        }
        return tmp;
    }

    //    TeamMembers:
    public void addTeamMember(TeamMember teamMember) {
        teamMembers.put("Team Member "+ teamMember.getId(),
teamMember.toMap());

        editor.setRepositoryStrings(teamMembers);
        editor.writeToTextFile(repo + "department.txt");
    }

    public void addManager(Manager manager) {
        teamMembers.put(manager.getTeam().toString() + " Manager",
manager.toMap());
```

```
    public EventAdvertisement createEventAdvert(Event event,
AdvertType type) {
        EventAdvertisement ad = new EventAdvertisement(event, type);
        eventAdverts.add(ad);
        return ad;
    }

    @Override
    public DesignAdvertisement createDesignAdvert(AdvertType type,
String notes) {
        DesignAdvertisement ad = new DesignAdvertisement(type, notes);
        designAdverts.add(ad);
        return ad;
    }

    @Override
    public Map<String, String> toMap() {
        Map<String, String> memberDetails = new HashMap<>();
        memberDetails.put("employeeInfo", this.employeeInfo.toString());
        Integer[] tmp = new Integer[managers.size()];
        for(int i = 0; i < managers.size(); i++) {
            tmp[i] = managers.get(i).getId();
        }
        memberDetails.put("managers", Arrays.toString(tmp));
        return memberDetails;
    }

    @Override
    public void addManager(Manager manager) {
        managers.add(manager);
        MarketingDepartment.fileManager.addManager(manager);
    }

    @Override
    public Manager getManager(Team team) {
        for (Manager manager: managers) {
            if (manager.getTeam() == team) {
                return manager;
            }
        }
        return null;
    }

    @Override
    public String toString() {
        StringBuilder str = new StringBuilder("\nHOD: ");
        str.append("\nEmployeeInfo:
").append(this.employeeInfo.toString());
        str.append("\nManagers: ");
        for (Manager manager : managers) {
            str.append("\n  ").append(manager.toString());
        }
        return str.toString();
    }

    @Override
    public Event requestPhotoshoot(int modelId) {
        return
App.modelingDepartment.requestPhotoshoot(Integer.toString(modelId),
"");
    }

    public static HOD parse(Map<String, String> hod) {
        return new HOD(
                Employee.parseEmployee(hod.get("employeeInfo")),
                MarketingDepartment.fileManager.getManagers()
        );
    }
}
```

```
public class TeamMember implements ITeamMember {
    static int nextid = 0;
    int id;
    Employee employeeInfo;
    Team team;

    TeamMember(int i, Employee employee, Team team) {
        id = i;
        editor.setRepositoryStrings(teamMembers);
        editor.writeToTextFile(repo + "department.txt");
    }

    public void addHOD(HOD hod) {
        teamMembers.put("HOD", hod.toMap());

        editor.setRepositoryStrings(teamMembers);
        editor.writeToTextFile(repo + "department.txt");
    }

    public ArrayList<Manager> getManagers() {
        ArrayList<Manager> tmp = new ArrayList<>();
        for(Map<String, String> member: teamMembers.values()) {
            if(member.containsKey("teamMembers")) {
                tmp.add(Manager.parse(member));
            }
        }
        return tmp;
    }

    public ArrayList<TeamMember> getTeamMembers(Team team) {
        ArrayList<TeamMember> tmp = new ArrayList<>();
        for(Map<String, String> member: teamMembers.values()) {
            if (member.containsValue(team.toString()) &&
!member.containsKey("teamMembers") &&
!member.containsKey("managers")) {
                tmp.add(TeamMember.parse(member));
            }
        }
        return tmp;
    }

    public ArrayList<TeamMember> getTeamMembers() {
        ArrayList<TeamMember> tmp = new ArrayList<>();
        for(Map<String, String> member: teamMembers.values()) {
            if (!member.containsKey("teamMembers") &&
!member.containsKey("managers")) {
                tmp.add(TeamMember.parse(member));
            }
        }
        return tmp;
    }

    public HOD getHOD() {
        return HOD.parse(teamMembers.get("HOD"));
    }
}
```

```
public class Manager implements IManager {
    static int nextid = 0;
    int id;

    private final Employee employeeInfo;
    private final ArrayList<TeamMember> teamMembers;
    private final Team team;

    public Manager(String name, Team team) {
        id = nextid;
        nextid++;
        employeeInfo =
App.hrDepartment.getEmployee(Department.MODELING, name);
        teamMembers =
MarketingDepartment.fileManager.getTeamMembers(team);
        this.team = team;
    }

    public Manager(int id, Employee employeeInfo, Team team) {
        this.id = id;
        if(id>=nextid) nextid = id+1;
        this.employeeInfo = employeeInfo;
        teamMembers =
MarketingDepartment.fileManager.getTeamMembers(team);
        this.team = team;
    }

    public Manager(Team team) {
```

```
        if (id >= nextid) nextid = id + 1;
        employeeInfo = employee;
        this.team = team;
    }

    TeamMember(Team team) {
        id = nextid;
        nextid++;
        employeeInfo = new Employee("teamMember", team.toString() + "
Member", Department.MODELING, "teamMember", "Employeed",
80000);
        this.team = team;

        MarketingDepartment.fileManager.addTeamMember(this);
    }

    @Override
    public int getId() {
        return id;
    }

    @Override
    public String toString() {
        return this.team.toString() + this.id + ": " + employeeInfo.toString();
    }

    @Override
    public Employee getEmployeeInfo() {
        return employeeInfo;
    }

    public Team getTeam() {
        return team;
    }

    @Override
    public Map<String, String> toMap() {
        Map<String, String> memberDetails = new HashMap<>();
        memberDetails.put("id", Integer.toString(this.id));
        memberDetails.put("employeeInfo", this.employeeInfo.toString());
        memberDetails.put("team", this.team.toString());
        return memberDetails;
    }

    public static TeamMember parse(Map<String, String> member) {
        return new TeamMember(
                Integer.parseInt(member.get("id")),
                Employee.parseEmployee(member.get("employeeInfo")),
                Team.parseTeam(member.get("team"))
        );

    }
}
```

```
        id = nextid;
        nextid++;
        employeeInfo = new Employee("manager", team.toString() + "
Manager", Department.MODELING, "Manager", "Employeed", 100000);
        teamMembers = new ArrayList<>();
        for(int i = 0; i <= 3; i++) {
            teamMembers.add(new TeamMember(team));
        }
        this.team = team;

        MarketingDepartment.fileManager.addManager(this);
    }

    @Override
    public int getId() {
        return id;
    }

    @Override
    public Employee getEmployeeInfo() {
        return employeeInfo;
    }

    @Override
    public Team getTeam() {
        return team;
    }

    @Override
    public ArrayList<TeamMember> getTeamMembers() {
        return teamMembers;
    }

    @Override
    public void addTeamMember(TeamMember member) {
        teamMembers.add(member);
        MarketingDepartment.fileManager.addTeamMember(member);
    }

    @Override
    public Map<String, String> toMap() {
        Map<String, String> memberDetails = new HashMap<>();
        memberDetails.put("id", Integer.toString(this.id));
        memberDetails.put("employeeInfo", this.employeeInfo.toString());
        Integer[] tmp = new Integer[teamMembers.size()];
        for(int i = 0; i < teamMembers.size(); i++) {
            tmp[i] = teamMembers.get(i).getId();
        }
        memberDetails.put("teamMembers", Arrays.toString(tmp));
        memberDetails.put("team", this.team.toString());
        return memberDetails;
    }

    @Override
    public String toString() {
        StringBuilder str = new StringBuilder("\n" + this.team.toString() + "
Manager: " + id);
        str.append("\nEmployeeInfo:
").append(this.employeeInfo.toString());
        str.append("\nTeam Members: ");
        for (TeamMember teamMember : teamMembers) {
            str.append("\n").append(teamMember.toString());
        }
        return str.toString();
    }

    public static Manager parse(Map<String, String> manager) {
        return new Manager(
                Integer.parseInt(manager.get("id")),
                Employee.parseEmployee(manager.get("employeeInfo")),
                Team.parseTeam(manager.get("team"))
        );
    }
}
```

Security [Kenny]:

```
/**
 * @author Kenny
 */

package src.Security.src;

public class AuditEmployee {

    private String employeeID, name, division, position,
            department, expertise, rating, yearsOfExperience,
            previousAssignment, currentAssignment;

    private AuditEmployee(){}

    public AuditEmployee(String employeeID, String name, String
division,
                         String position, String department, String expertise,
                         String rating, String yearsOfExperience, String
previousAssignment,
                         String currentAssignment){
        this.employeeID = employeeID;
        this.name = name;
        this.division = division;
        this.position = position;
        this.department = department;
        this.expertise = expertise;
        this.rating = rating;
        this.yearsOfExperience = yearsOfExperience;
        this.previousAssignment = previousAssignment;
        this.currentAssignment = currentAssignment;
    }

    public String getEmployeeID() {
        return employeeID;
    }

    public String getName() {
        return name;
    }

    public String getDivision() {
        return division;
    }

    public String getPosition() {
        return position;
    }

    public String getDepartment() {
        return department;
    }

    public String getExpertise() {
        return expertise;
    }

    public String getRating() {
        return rating;
    }

    public String getYearsOfExperience() {
        return yearsOfExperience;
    }

    public String getPreviousAssignment() {
        return previousAssignment;
    }

    public String getCurrentAssignment() {
        return currentAssignment;
    }

    public void setEmployeeID(String employeeID) {
        this.employeeID = employeeID;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setDivision(String division) {
        this.division = division;
    }

    public void setPosition(String position) {
        this.position = position;
    }

    public void setDepartment(String department) {
        this.department = department;
    }

    public void setExpertise(String expertise) {
        this.expertise = expertise;
    }

    public void setRating(String rating) {
        this.rating = rating;
    }

    public void setYearsOfExperience(String yearsOfExperience) {
        this.yearsOfExperience = yearsOfExperience;
    }

    public void setPreviousAssignment(String previousAssignment) {
        this.previousAssignment = previousAssignment;
    }

    public void setCurrentAssignment(String currentAssignment) {
        this.currentAssignment = currentAssignment;
    }
}

/**
 * @author Kenny
 */

package src.Security.src;

import jdk.jshell.Snippet;
import src.Security.src.interfaces.AuditSchedulerController;
import src.TextEditor.PoorTextEditor;

import java.io.*;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.*;

public class AuditScheduler implements
src.Security.src.interfaces.AuditScheduler {
```

```
    PriorityQueue<AuditSchedules>
ongoingAuditSchedulesPriorityQueue = new PriorityQueue<>(
            Comparator.comparingInt(request ->
-Integer.parseInt(request.getPriorityLevel()))
    );
    PriorityQueue<AuditSchedules>
passedAuditSchedulesPriorityQueue = new PriorityQueue<>(
            Comparator.comparingInt(request ->
-Integer.parseInt(request.getPriorityLevel()))
    );
    PriorityQueue<AuditSchedules>
failedAuditSchedulesPriorityQueue = new PriorityQueue<>(
            Comparator.comparingInt(request ->
-Integer.parseInt(request.getPriorityLevel()))
    );
    PriorityQueue<AuditSchedules> allAuditSchedulesPriorityQueue =
new PriorityQueue<>(
            Comparator.comparingInt(request ->
-Integer.parseInt(request.getPriorityLevel()))
    );
    /**
     * Repository to store all security request from departments
     */
    Map<String, AuditSchedules> auditSchedulesRepository = new
LinkedHashMap<>();

    // sorts audit employees first by division, then position, then by
rating in descending order
    PriorityQueue<AuditEmployee>
availableAuditEmployeePriorityQueue = new PriorityQueue<>(
            Comparator.comparing(AuditEmployee::getDivision)
                    .thenComparing(AuditEmployee::getPosition)
                    .thenComparing((e1, e2) ->
e2.getRating().compareTo(e1.getRating()))
    );
    PriorityQueue<AuditEmployee> allAuditEmployeePriorityQueue =
new PriorityQueue<>(
            Comparator.comparing(AuditEmployee::getDivision)
                    .thenComparing(AuditEmployee::getPosition)
                    .thenComparing((e1, e2) ->
e2.getRating().compareTo(e1.getRating()))
    );
    /**
     * Repository to store all security employees obtained
     */
    Map<String, AuditEmployee> auditEmployeeRepository = new
LinkedHashMap<>();

    private final String auditSchedulesDir =
"src/Security/repository/auditSchedules/";
    private final String auditEmployeeDir =
"src/Security/repository/auditEmployees/";

    private final String printedAuditSchedulesDir =
"src/Security/repository/printedAuditSchedules/";

    PoorTextEditor editor = new PoorTextEditor();
    private File[] auditSchedulesFiles = null;



    @Override
    public boolean addAuditPersonnel() {
        return false;
    }

    @Override
    public boolean deleteAuditPersonnel(String employeeID) {
        return false;
    }

    @Override
    public boolean showOngoingAudits() {
        if (!retrieveAllAudits()){
            return false;
        }

        for (AuditSchedules schedule :
ongoingAuditSchedulesPriorityQueue){

            auditScheduleToText(schedule);
        }
        return true;
    }

    @Override
    public boolean showPassedAudits() {
        if (!retrieveAllAudits()){
            return false;
        }

        for (AuditSchedules schedules :
passedAuditSchedulesPriorityQueue){

            auditScheduleToText(schedules);
        }
        return true;
    }

    @Override
    public boolean showFailedAudits() {
        if (!retrieveAllAudits()){
            return false;
        }

        for (AuditSchedules schedule :
failedAuditSchedulesPriorityQueue){

            auditScheduleToText(schedule);
        }
        return true;
    }

    @Override
    public boolean showAllAudits() {
        if (!retrieveAllAudits()){
            return false;
        }

        for (AuditSchedules schedule : allAuditSchedulesPriorityQueue){

            auditScheduleToText(schedule);
        }
        return true;
    }

    @Override
    public boolean showAuditByID(String auditID) {
        if (!retrieveAllAudits()){
            return false;
        }

        if (!auditSchedulesRepository.containsKey(auditID)){
            System.out.println("Audit schedule with ID: " + auditID + "not
found");
            return false;
        }

        auditScheduleToText(auditSchedulesRepository.get(auditID));
        return true;
```

```
    }

    @Override
    public boolean showFreeAuditEmployees() {

        if (!retrieveEmployees()){
            return false;
        }

        for (AuditEmployee employee :
availableAuditEmployeePriorityQueue){

            auditEmployeeToText(employee);
        }
        return true;
    }

    @Override
    public boolean showAllAuditEmployees() {

        if (!retrieveEmployees()){
            return false;
        }

        for (AuditEmployee employee : allAuditEmployeePriorityQueue){

            auditEmployeeToText(employee);
        }
        return true;
    }

    @Override
    public boolean createAudit() {

        Scanner scan = new Scanner(System.in);

        String scheduleID = "audit-" + IDGenerator();
        System.out.println("Enter priority level: (0 = Low,  1 = Medium, 2
= High, 3 = Emergency)");
        String priorityLevel = scan.nextLine();

        System.out.println("Enter department to audit: ");
        String department = scan.nextLine();

        System.out.println("Enter audit location: (Country - Specific
Location)");
        String location = scan.nextLine();

        System.out.println("Enter description of audit: ");
        String description = scan.nextLine();

        System.out.println("Enter duration of audit: (MM-DD-YYYY
TT:MM a.m. - TT:MM p.m.)");
        String duration = scan.nextLine();

        System.out.println("Enter tasks of audit: ");
        String tasks = scan.nextLine();

        String status = "ongoing";
        String dateScheduled = dateIssuer();

        AuditSchedules schedule = new AuditSchedules(
                scheduleID,
                priorityLevel,
                department,
                location,
                description,
                duration,
                tasks,
                status,
                dateScheduled);

        // assigning security personnel to schedule
        if (!addPersonnelToSchedule(schedule)){
            System.out.println("Schedule creation for Request ID: " +
scheduleID + " was cancelled");
            return false;
        }

        // writing schedule to repository
        editor.setRepository(auditScheduleToHashMap(schedule));
        editor.writeToTextFile(auditSchedulesDir +
schedule.getScheduleID() + ".txt");

        // writing printed schedules to repository
        printSecuritySchedule(schedule);

        return true;
    }

    @Override
    public boolean deleteAudit(String auditID) {

        if (auditEmployeeRepository.isEmpty()){

            if (!retrieveEmployees()){
                return false;
            }
        }

        if (!retrieveAllAudits()){
            return false;
        }

        if (!auditSchedulesRepository.containsKey(auditID)){

            System.out.println("Audit Schedule ID: " + auditID + " not
found");
            return false;
        }

        AuditSchedules schedule =
auditSchedulesRepository.get(auditID);

        if (!schedule.getAssignedEmployeeIDs().isEmpty()){

            List<String> scheduledEmployeeIDs =
schedule.getAssignedEmployeeIDs();
            editor.processTextFile(auditEmployeeDir +
"auditEmployeeList.txt");

            for (String s : scheduledEmployeeIDs){

                editor.setValue(s,"currentAssignment", "free");
            }

            editor.writeToTextFile(auditEmployeeDir +
"auditEmployeeList.txt");
        }

        File fileToDelete = new File(auditSchedulesDir, auditID + ".txt");

        if (fileToDelete.exists()) {

            if (fileToDelete.delete()) {

                System.out.println("Audit schedule deleted successfully");
            } else {

                System.out.println("Failed to audit schedule");
```

```
            return false;
        }
    } else {
        System.out.println("Audit schedule not found");
        return false;
    }
    return true;
}

@Override
public boolean editAudit(String auditID) {

    if (!retrieveAllAudits()){
        return false;
    }

    if (!auditSchedulesRepository.containsKey(auditID)){

        System.out.println("Audit schedule with ID: " + auditID + "
does not exists");
        return false;
    }

    AuditSchedules schedule =
auditSchedulesRepository.get(auditID);
    Scanner scan = new Scanner(System.in);

    System.out.println("Enter priority level: (0 = Low,  1 = Medium, 2
= High, 3 = Emergency)");
    schedule.setPriorityLevel(scan.nextLine());

    System.out.println("Enter department to audit: ");
    schedule.setDepartment(scan.nextLine());

    System.out.println("Enter audit location: (Country - Specific
Location)");
    schedule.setLocation(scan.nextLine());

    System.out.println("Enter description of audit: ");
    schedule.setDescription(scan.nextLine());

    System.out.println("Enter duration of audit: (MM-DD-YYYY
TT:MM a.m. - TT:MM p.m.)");
    schedule.setDuration(scan.nextLine());

    System.out.println("Enter tasks of audit: ");
    schedule.setTasks(scan.nextLine());

    System.out.println("Enter audit status: (ongoing, pass, fail)");
    schedule.setStatus(scan.nextLine());

    if (!addPersonnelToSchedule(schedule)){
        System.out.println("Updating cancelled");
        return false;
    }

    // writing schedule to repository
    editor.setRepository(auditScheduleToHashMap(schedule));
    editor.writeToTextFile(auditSchedulesDir +
schedule.getScheduleID() + ".txt");

    // writing printed schedules to repository
    printSecuritySchedule(schedule);

    System.out.println("Successfully updated");
    return true;
}

@Override
public boolean sendFailedAudit(String auditID) {
    return false;
}

/**
 * To add audit personnel to a schedule
 * @param schedule given schedule
 */
private boolean addPersonnelToSchedule(AuditSchedules
schedule){

    if (auditEmployeeRepository.isEmpty()){

        if (!retrieveEmployees()){
            return false;
        }
    }

    List<String> selectedEmployeeIDs = new ArrayList<>();
    if (!schedule.getAssignedEmployeeIDs().isEmpty()){
        selectedEmployeeIDs = schedule.getAssignedEmployeeIDs();
    }
    int employeeCount =
schedule.getAssignedEmployeeIDs().size();
    Scanner scan = new Scanner(System.in);

    boolean endProgram = false;
    boolean returnValue = false;

    while (!endProgram){

        if (Integer.parseInt(schedule.getPriorityLevel()) < 3){

System.out.println(minimumEmployeeRequirement(schedule.getPrior
ityLevel()) - employeeCount + " employees left to add\n");
        }
        else {
            System.out.println("<<Overridden>> employees left to
add");
        }
        System.out.println("1. Add Audit Employee by ID");
        System.out.println("2. Remove Audit Employee by ID");
        System.out.println("3. Show Scheduled Employees");
        System.out.println("4. Complete Schedule");
        System.out.println("5. Cancel Schedule");
        System.out.println("0. Exit");

        int choice = scan.nextInt();
        scan.nextLine();

        switch (choice){

            case 1:
                if (employeeCount >=
minimumEmployeeRequirement(schedule.getPriorityLevel())){
                    System.out.println("Exceeded limit");
                    break;
                }

                System.out.println("Enter Employee ID: ");
                String employeeID = scan.nextLine();

                if (!auditEmployeeRepository.containsKey(employeeID)){

                    System.out.println("Employee with ID: " + employeeID
+ " does not exists");
                    break;
                }

                AuditEmployee employee =
auditEmployeeRepository.get(employeeID);
```

```
                if (!employee.getCurrentAssignment().equals("free")){

                    System.out.println("This employee has already been
assigned to :" + employee.getCurrentAssignment() + "\n" +
                            "Overwrite? (y/n)\n" +
                            "!!!WARNING!!! This process is irreversible!");

                    String yesNo = scan.nextLine();

                    if (yesNo.equals("y")){

employee.setPreviousAssignment(employee.getCurrentAssignment()
);

employee.setCurrentAssignment(schedule.getScheduleID());

selectedEmployeeIDs.add(employee.getEmployeeID());

                        // updating employee assignments
                        editor.processTextFile(auditEmployeeDir +
"auditEmployeeList.txt");
                        editor.setValue(employee.getEmployeeID(),
"previousAssignment", employee.getPreviousAssignment());
                        editor.setValue(employee.getEmployeeID(),
"currentAssignment", employee.getCurrentAssignment());
                        editor.writeToTextFile(auditEmployeeDir +
"auditEmployeeList.txt");

                        employeeCount++;
                    }
                }
                else {


availableAuditEmployeePriorityQueue.remove(employee);

employee.setCurrentAssignment(schedule.getScheduleID());

selectedEmployeeIDs.add(employee.getEmployeeID());

                    // updating employee assignments
                    editor.processTextFile(auditEmployeeDir +
"auditEmployeeList.txt");
                    editor.setValue(employee.getEmployeeID(),
"currentAssignment", employee.getCurrentAssignment());
                    editor.writeToTextFile(auditEmployeeDir +
"auditEmployeeList.txt");

                    employeeCount++;
                }
                break;
            case 2:
                if (selectedEmployeeIDs.isEmpty()){

                    System.out.println("No employees to remove from this
schedule");
                    break;
                }

                System.out.println("Enter Employee ID to remove: ");
                String removingID = scan.nextLine();

                if (!selectedEmployeeIDs.contains(removingID)){
                    System.out.println("Employee ID: " + removingID + "
not assigned to this schedule");
                    break;
                }

                selectedEmployeeIDs.remove(removingID);

                AuditEmployee employee2 =
auditEmployeeRepository.get(removingID);
                employee2.setCurrentAssignment("free");
                availableAuditEmployeePriorityQueue.add(employee2);

                // updating employee assignments
                editor.processTextFile(auditEmployeeDir +
"auditEmployeeList.txt");
                editor.setValue(employee2.getEmployeeID(),
"currentAssignment", employee2.getCurrentAssignment());
                editor.writeToTextFile(auditEmployeeDir +
"auditEmployeeList.txt");

                employeeCount--;
                break;
            case 3:
                if (selectedEmployeeIDs.isEmpty()){
                    System.out.println("No employees to show from this
schedule");
                    break;
                }

                System.out.println("Selected Employees by ID: ");
                for (String s : selectedEmployeeIDs){
                    System.out.println(s);
                }
                System.out.println();
                break;
            case 4:
                if (employeeCount != 0){

schedule.setAssignedEmployeeIDs(selectedEmployeeIDs);
                    System.out.println("Assignment created");
                    endProgram = true;
                    returnValue = true;
                }
                else {
                    System.out.println("Have at least one assigned
employee");
                }
                break;
            case 5:
                for (String s : selectedEmployeeIDs){

                    AuditEmployee resetEmployee =
auditEmployeeRepository.get(s);
                    resetEmployee.setCurrentAssignment("free");

availableAuditEmployeePriorityQueue.add(resetEmployee);

                    // updating employee assignments
                    editor.processTextFile(auditEmployeeDir +
"auditEmployeeList.txt");
                    editor.setValue(resetEmployee.getEmployeeID(),
"currentAssignment", resetEmployee.getCurrentAssignment());
                    editor.writeToTextFile(auditEmployeeDir +
"auditEmployeeList.txt");
                }
                return returnValue;
            case 0:
                if (selectedEmployeeIDs.isEmpty()){
                    endProgram = true;
                }
                else {
                    System.out.println("Remove all assigned employees
or cancel before exiting");
                }
                break;
            default:
                System.out.println("Invalid choice. Try again");
```

```
            }
        }
        return returnValue;
    }

    /**
     * Returns current time in MM-dd-yyyy-HH-mm-ss format when
called
     * @return date in MM-dd-yyyy-HH-mm-ss format
     */
    private String dateIssuer(){

        LocalDateTime timeNow = LocalDateTime.now();
        DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("MM-dd-yyyy-HH-mm-ss");

        return timeNow.format(formatter);
    }

    /**
     * Generates IDs for security schedules
     * @return generated security schedule ID
     */
    private String IDGenerator(){

        LocalDateTime timeNow = LocalDateTime.now();
        DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("MMddyyyyHHmmss");

        return timeNow.format(formatter);
    }

    /**
     * Retrieve all audit schedules
     * @return true if successful retrieval, otherwise false
     */
    private boolean retrieveAllAudits(){

        if (!retrieveAuditFiles()){
            return false;
        }

        // clearing everything for new retrieval
        auditSchedulesRepository.clear();
        allAuditSchedulesPriorityQueue.clear();
        ongoingAuditSchedulesPriorityQueue.clear();
        failedAuditSchedulesPriorityQueue.clear();
        passedAuditSchedulesPriorityQueue.clear();

        // saving file requests to repository
        for (File f : auditSchedulesFiles) {

            editor.processTextFile(auditSchedulesDir + f.getName());
            String[] audits = editor.getArrayNames();
            String id = audits[0];

            for (String s : audits) {

                AuditSchedules audit = new AuditSchedules(s,
                        editor.retrieveValue(s, "priorityLevel"),
                        editor.retrieveValue(s, "department"),
                        editor.retrieveValue(s, "location"),
                        editor.retrieveValue(s, "description"),
                        editor.retrieveValue(s, "duration"),
                        editor.retrieveValue(s, "tasks"),
                        editor.retrieveValue(s, "status"),
                        editor.retrieveValue(s, "dateScheduled"));

                List<String> employeeIDs = new ArrayList<>();
                int num = 0;
                String employeeNum = "employee" + num;
                while ((editor.retrieveValue(id, employeeNum) != null)){

                    employeeIDs.add(editor.retrieveValue(id, "employee" +
num));
                    num++;
                    employeeNum = "employee" + num;
                }
                audit.setAssignedEmployeeIDs(employeeIDs);

                if (audit.getPriorityLevel() == null || audit.getDepartment()
== null ||
                        audit.getLocation() == null || audit.getDescription() ==
null ||
                        audit.getDuration() == null || audit.getTasks() == null ||
                        audit.getStatus() == null || audit.getDateScheduled()
== null) {

                    System.out.println("Invalid security requests detected");
                    return false;
                }

                // adding audit schedules to repository and priority queue
                auditSchedulesRepository.put(s, audit);
                allAuditSchedulesPriorityQueue.add(audit);

                if (audit.getStatus().equals("ongoing")){
                    ongoingAuditSchedulesPriorityQueue.add(audit);
                }
                if (audit.getStatus().equals("pass")){
                    passedAuditSchedulesPriorityQueue.add(audit);
                }
                if (audit.getStatus().equals("fail")){
                    failedAuditSchedulesPriorityQueue.add(audit);
                }
            }
        }
        return true;
    }

    /**
     * Retrieve files containing audits
     * @return files containing audits
     */
    private boolean retrieveAuditFiles(){

        File directory = new File(auditSchedulesDir);
        File[] textFiles = null;

        if (directory.exists() && directory.isDirectory()){

            //grab list of text files
            FilenameFilter textFileFilter = (((dir, name) ->
name.toLowerCase().endsWith(".txt")));
            textFiles = directory.listFiles(textFileFilter);
        }
        else {
            System.out.println("Repository not found");
            return false;
        }

        if (textFiles != null){

            if (textFiles.length == 0){
                System.out.println("No audit schedules found");
                return false;
            }
            auditSchedulesFiles = textFiles;
        }
        else {
```

```
            System.out.println("No audit schedules found");
            return false;
        }
        return true;
    }

    /**
     * Retrieve all audit employees
     * @return all audit employees
     */
    private boolean retrieveEmployees(){

        availableAuditEmployeePriorityQueue.clear();
        allAuditEmployeePriorityQueue.clear();
        auditEmployeeRepository.clear();

        File directory = new File(auditEmployeeDir);
        File[] textFiles = null;

        if (directory.exists() && directory.isDirectory()){

            //grab list of text files
            FilenameFilter textFileFilter = ((dir, name) ->
name.toLowerCase().endsWith(".txt"));
            textFiles = directory.listFiles(textFileFilter);
        }
        else {
            System.out.println("Repository not found");
            return false;
        }

        if (textFiles != null){

            if (textFiles.length == 0){
                System.out.println("No audit employees found");
                return false;
            }
        }
        else {
            System.out.println("No audit employees found");
            return false;
        }

        // saving employees to repository
        for (File f : textFiles) {

            editor.processTextFile(auditEmployeeDir + f.getName());
            String[] employees = editor.getArrayNames();

            for (String s : employees) {

                AuditEmployee employee = new AuditEmployee(s,
                        editor.retrieveValue(s, "name"),
                        editor.retrieveValue(s, "division"),
                        editor.retrieveValue(s, "position"),
                        editor.retrieveValue(s, "department"),
                        editor.retrieveValue(s, "expertise"),
                        editor.retrieveValue(s, "rating"),
                        editor.retrieveValue(s, "yearsOfExperience"),
                        editor.retrieveValue(s, "previousAssignment"),
                        editor.retrieveValue(s, "currentAssignment"));

                if (employee.getEmployeeID() == null ||
employee.getName() == null ||
                        employee.getDivision() == null ||
employee.getPosition() == null ||
                        employee.getDepartment() == null ||
employee.getExpertise() == null ||
                        employee.getRating() == null ||
employee.getYearsOfExperience() == null ||
                        employee.getPreviousAssignment() == null ||
employee.getCurrentAssignment() == null) {

                    System.out.println("Invalid employee data detected");
                    return false;
                }

                // adding available employees to priority queue
                if (employee.getCurrentAssignment().equals("free")){
                    availableAuditEmployeePriorityQueue.add(employee);
                }
                // adding security request to priority queue
                allAuditEmployeePriorityQueue.add(employee);
                auditEmployeeRepository.put(s, employee);
            }
        }
        return true;
    }

    /**
     * Converts SecuritySchedules object into nested HashMap
     * @param schedule specified SecuritySchedule
     * @return HashMap of SecuritySchedule
     */
    private Map<String, Object>
auditScheduleToHashMap(AuditSchedules schedule){

        Map<String, Object> innerMap = new LinkedHashMap<>();
        innerMap.put("priorityLevel", schedule.getPriorityLevel());
        innerMap.put("department", schedule.getDepartment());
        innerMap.put("location", schedule.getLocation());
        innerMap.put("description", schedule.getDescription());
        innerMap.put("duration", schedule.getDuration());
        innerMap.put("tasks", schedule.getTasks());
        innerMap.put("status",schedule.getStatus());
        innerMap.put("dateScheduled", schedule.getDateScheduled());

        // debug
        //System.out.println("Assigned Employee IDs hashing part: " +
schedule.getAssignedEmployeeIDs());

        int num = 0;
        for (String s : schedule.getAssignedEmployeeIDs()){
            String employee = "employee" + num;
            innerMap.put(employee, s);
            num++;
        }

        Map<String, Object> outerMap = new LinkedHashMap<>();
        outerMap.put(schedule.getScheduleID(), innerMap);

        return outerMap;
    }

    /**
     * Pretty print audit schedules
     * @param schedule specific audit schedule
     */
    private void auditScheduleToText(AuditSchedules schedule){


System.out.println("=====================================
==================");
        System.out.println("Audit Schedule ID: " +
schedule.getScheduleID());
        System.out.println("Priority Level: " +
schedule.getPriorityLevel());
        System.out.println("Department: " + schedule.getDepartment());
        System.out.println("-----------------------------------------------------");
        System.out.println("Location: " + schedule.getLocation());
```

```
        System.out.println("Description: " + schedule.getDescription());
        System.out.println("Duration: " + schedule.getDuration());
        System.out.println("Tasks: " + schedule.getTasks());
        System.out.println("-------------------------------------------------------");
        System.out.println("Date Issued: " +
schedule.getDateScheduled());
        System.out.println("Audit Status: " + schedule.getStatus());
        System.out.println("-------------------------------------------------------");
        System.out.println("Assigned Security Members:");
        for (String s : schedule.getAssignedEmployeeIDs()){
            System.out.println(s);
        }

System.out.println("=====================================
==================\n");
    }

    /**
     * Pretty print security employee
     * @param employee specific audit employee
     */
    private void auditEmployeeToText(AuditEmployee employee){


System.out.println(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>");
        System.out.println("Employee ID: " +
employee.getEmployeeID());
        System.out.println("Name: " + employee.getName());
        System.out.println("Division: " + employee.getDivision());
        System.out.println("Position: " + employee.getPosition());
        System.out.println("Department: " + employee.getDepartment());
        System.out.println("-------------------------------------------------------");
        System.out.println("Expertise: " + employee.getExpertise());
        System.out.println("Rating: " + employee.getRating());
        System.out.println("Years of Experience: " +
employee.getYearsOfExperience());
        System.out.println("Previous Assignment: " +
employee.getPreviousAssignment());
        System.out.println("Current Assignment: " +
employee.getCurrentAssignment());

System.out.println("<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
<<<<<<<<<<<<<<<<\n");
    }

    /**
     * To created printable audit schedules
     * @param schedule specific audit schedule
     */
    private void printSecuritySchedule(AuditSchedules schedule){

        BufferedWriter writer = null;
        try{
            writer = new BufferedWriter(new
FileWriter(printedAuditSchedulesDir + schedule.getScheduleID() +
".txt"));

writer.write("=============================================
==========================\n");
            writer.write("AUDIT SCHEDULE\n");
            writer.write("Schedule ID: " + schedule.getScheduleID() +
"\n");
            writer.write("Priority Level: " + schedule.getPriorityLevel() + "
(3 = Emergency, 2 = High, 1 = Medium, 0 = Low)\n");
            writer.write("Department: " + schedule.getDepartment() +
"\n");

writer.write("-------------------------------------------------------------------\n");
            writer.write("Location: " + schedule.getLocation() + "\n");
            writer.write("Description: " + schedule.getDescription() + "\n");
            writer.write("Duration :" + schedule.getDuration() + "\n");
            writer.write("Tasks: " + schedule.getTasks() + "\n");

writer.write("-------------------------------------------------------------------\n");
            writer.write("Date Issued: " + schedule.getDateScheduled() +
"\n");
            writer.write("Audit status: " + schedule.getStatus() + "\n");

writer.write("-------------------------------------------------------------------\n");
            writer.write("Assigned Personnel by IDs:\n");
            for (String s : schedule.getAssignedEmployeeIDs()){
                writer.write(s + "\n");
            }

writer.write("=============================================
==========================\n");

        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (writer != null) {
                    writer.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    /**
     * Returns minimum amount of security personnel required for a
given
     * priority level
     * @param priorityLevel given priority level
     * @return minimum number of employees required
     */
    private int minimumEmployeeRequirement(String priorityLevel){

        return switch (priorityLevel) {
            case "3" -> Integer.MAX_VALUE;
            case "2" -> 10;
            case "1" -> 7;
            case "0" -> 3;
            default -> -1;
        };
    }
}

/**
 * @author Kenny
 */

package src.Security.src;

import src.App;

import java.util.Scanner;

public class AuditSchedulerController implements
src.Security.src.interfaces.AuditSchedulerController {

    private AuditSchedulerController controller;

    private AuditScheduler scheduler = null;

    public AuditSchedulerController (AuditSchedulerController
controller){
        this.controller = controller;
```

```
    }

    public AuditSchedulerController(){}

    /**
     * Run the audit program
     */
    public void run() throws Exception {

        System.out.println("Welcome to the Audit Security Scheduler
System");

        boolean exit = false;

        while (!exit) {

            Scanner scan = new Scanner(System.in);

            System.out.println("1. Add New Audit Personnel");
            System.out.println("2. Delete Audit Personnel by ID");
            System.out.println("3. Show Ongoing Audits");
            System.out.println("4. Show Passed Audits");
            System.out.println("5. Show Failed Audits");
            System.out.println("6. Show All Audits");
            System.out.println("7. Show Audit by ID");
            System.out.println("8. Show Available Audit Employees");
            System.out.println("9. Show All Audit Employees");
            System.out.println("10. Create New Audit");
            System.out.println("11. Delete Audit by ID");
            System.out.println("12. Edit Audit by ID");
            System.out.println("13. Send Failed Audit by ID");
            System.out.println("0. Exit program");

            int choice = scan.nextInt();
            scan.nextLine();

            switch (choice) {

                case 1:
                    addAuditPersonnel();
                    System.out.println();
                    break;
                case 2:
                    System.out.println("Enter Audit Employee ID to delete: ");
                    String deleteID = scan.nextLine();
                    deleteAuditPersonnel(deleteID);
                    System.out.println();
                    break;
                case 3:
                    showOngoingAudits();
                    System.out.println();
                    break;
                case 4:
                    showPassedAudits();
                    System.out.println();
                    break;
                case 5:
                    showFailedAudits();
                    System.out.println();
                    break;
                case 6:
                    showAllAudits();
                    System.out.println();
                    break;
                case 7:
                    System.out.println("Enter Audit ID: ");
                    String findAuditID = scan.next();
                    showAuditByID(findAuditID);
                    System.out.println();
                    break;
                case 8:
                    showFreeAuditEmployees();
                    System.out.println();
                    break;
                case 9:
                    showAllAuditEmployees();
                    System.out.println();
                    break;
                case 10:
                    createAudit();
                    System.out.println();
                    break;
                case 11:
                    System.out.println("Enter Audit Schedule ID to delete : ");
                    String deleteAuditID = scan.next();
                    deleteAudit(deleteAuditID);
                    System.out.println();
                    break;
                case 12:
                    System.out.println("Enter Audit Schedule ID to edit : ");
                    String editAuditID = scan.next();
                    editAudit(editAuditID);
                    System.out.println();
                    break;
                case 13:
                    System.out.println("Enter Failed Audit Schedule ID to
send : ");
                    String sendFailedAuditID = scan.next();
                    sendFailedAudit(sendFailedAuditID);
                    System.out.println();
                    break;
                case 0:
                    System.out.println("Closing program...");
                    exit = true;
                    App.prompt();
                default:
                    System.out.println("Incorrect choice. Try again");
            }
        }
    }

    /**
     * To define one instance of the class SecurityScheduler to
     * call SecurityScheduler methods
     * @return SecurityScheduler instance
     */
    private AuditScheduler getSchedulerInstance(){

        if (scheduler == null){
            scheduler = new AuditScheduler();
        }
        return scheduler;
    }

    @Override
    public boolean addAuditPersonnel() {
        return this.getSchedulerInstance().addAuditPersonnel();
    }

    @Override
    public boolean deleteAuditPersonnel(String employeeID) {
        return
this.getSchedulerInstance().deleteAuditPersonnel(employeeID);
    }

    @Override
    public boolean showOngoingAudits() {
        return this.getSchedulerInstance().showOngoingAudits();
    }
```

```
    @Override
    public boolean showPassedAudits() {
        return this.getSchedulerInstance().showPassedAudits();
    }

    @Override
    public boolean showFailedAudits() {
        return this.getSchedulerInstance().showFailedAudits();
    }

    @Override
    public boolean showAllAudits() {
        return this.getSchedulerInstance().showAllAudits();
    }

    @Override
    public boolean showAuditByID(String auditID) {
        return this.getSchedulerInstance().showAuditByID(auditID);
    }

    @Override
    public boolean showFreeAuditEmployees() {
        return this.getSchedulerInstance().showFreeAuditEmployees();
    }

    @Override
    public boolean showAllAuditEmployees() {
        return this.getSchedulerInstance().showAllAuditEmployees();
    }

    @Override
    public boolean createAudit() {
        return this.getSchedulerInstance().createAudit();
    }

    @Override
    public boolean deleteAudit(String auditID) {
        return this.getSchedulerInstance().deleteAudit(auditID);
    }

    @Override
    public boolean editAudit(String auditID) {
        return this.getSchedulerInstance().editAudit(auditID);
    }

    @Override
    public boolean sendFailedAudit(String auditID) {
        return this.getSchedulerInstance().sendFailedAudit(auditID);
    }
}
/**
 * @author Kenny
 */

package src.Security.src;

import java.util.ArrayList;
import java.util.List;

public class AuditSchedules {

    private String scheduleID, priorityLevel, department, location,
            description, duration, tasks, status, dateScheduled;
    private List<String> assignedEmployeeIDs = new ArrayList<>();

    public AuditSchedules(){};

    public AuditSchedules (String scheduleID, String priorityLevel,
String department,
                           String location, String description, String duration,
                           String tasks, String status, String dateScheduled) {
        this.scheduleID = scheduleID;
        this.priorityLevel = priorityLevel;
        this.department = department;
        this.location = location;
        this.description = description;
        this.duration = duration;
        this.tasks = tasks;
        this.status = status;
        this.dateScheduled = dateScheduled;
    }

    public String getScheduleID(){
        return scheduleID;
    }

    public String getPriorityLevel(){
        return priorityLevel;
    }

    public String getDepartment() {
        return department;
    }

    public String getLocation(){
        return location;
    }

    public String getDescription(){
        return description;
    }

    public String getDuration(){
        return duration;
    }

    public String getTasks(){
        return tasks;
    }

    public String getStatus(){
        return status;
    }

    public String getDateScheduled(){
        return dateScheduled;
    }

    public List<String> getAssignedEmployeeIDs(){
        return assignedEmployeeIDs;
    }

    public void setScheduleID(String scheduleID){
        this.scheduleID = scheduleID;
    }

    public void setPriorityLevel(String priorityLevel){
        this.priorityLevel = priorityLevel;
    }

    public void setDepartment(String department){
        this.department = department;
    }

    public void setLocation(String location){
        this.location = location;
    }

    public void setDescription(String description){
        this.description = description;
```

```
    }

    public void setDuration(String duration){
        this.duration = duration;
    }

    public void setTasks(String tasks){
        this.tasks = tasks;
    }

    public void setStatus(String status){
        this.tasks = status;
    }

    public void setDateScheduled(String dateScheduled){
        this.dateScheduled = dateScheduled;
    }

    public void setAssignedEmployeeIDs(List<String>
assignedEmployeeIDs){
        this.assignedEmployeeIDs = assignedEmployeeIDs;
    }
}

/**
 * @author Kenny
 */

package src.Security.src;

public class SecurityEmployee {

    private String employeeID, name, division, position,
            department, expertise, rating, yearsOfExperience,
            previousAssignment, currentAssignment;

    public SecurityEmployee(){}

    public SecurityEmployee(String employeeID, String name, String
division,
                            String position, String department, String expertise,
                            String rating, String yearsOfExperience, String
previousAssignment,
                            String currentAssignment){
        this.employeeID = employeeID;
        this.name = name;
        this.division = division;
        this.position = position;
        this.department = department;
        this.expertise = expertise;
        this.rating = rating;
        this.yearsOfExperience = yearsOfExperience;
        this.previousAssignment = previousAssignment;
        this.currentAssignment = currentAssignment;
    };

    public String getEmployeeID() {
        return employeeID;
    }

    public String getName() {
        return name;
    }

    public String getDivision() {
        return division;
    }

    public String getPosition() {
        return position;
    }

    public String getDepartment() {
        return department;
    }

    public String getExpertise() {
        return expertise;
    }

    public String getRating() {
        return rating;
    }

    public String getYearsOfExperience() {
        return yearsOfExperience;
    }

    public String getPreviousAssignment() {
        return previousAssignment;
    }

    public String getCurrentAssignment() {
        return currentAssignment;
    }

    public void setEmployeeID(String employeeID) {
        this.employeeID = employeeID;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setDivision(String division) {
        this.division = division;
    }

    public void setPosition(String position) {
        this.position = position;
    }

    public void setDepartment(String department) {
        this.department = department;
    }

    public void setExpertise(String expertise) {
        this.expertise = expertise;
    }

    public void setRating(String rating) {
        this.rating = rating;
    }

    public void setYearsOfExperience(String yearsOfExperience) {
        this.yearsOfExperience = yearsOfExperience;
    }

    public void setPreviousAssignment(String previousAssignment) {
        this.previousAssignment = previousAssignment;
    }

    public void setCurrentAssignment(String currentAssignment) {
        this.currentAssignment = currentAssignment;
    }
}

/**
 * @author Kenny
 */
```

```
package src.Security.src;

public class SecurityRequests {

    private String requestID, priorityLevel, department, location,
            description, duration, tasks, specialReqs, dateIssued,
            resolved, fileName;

    public SecurityRequests(){}

    public SecurityRequests (String requestID, String priorityLevel,
String department,
                             String location, String description, String duration,
                             String tasks, String specialReqs, String dateIssued,
                             String resolved){
        this.requestID = requestID;
        this.priorityLevel = priorityLevel;
        this.department = department;
        this.location = location;
        this.description = description;
        this.duration = duration;
        this.tasks = tasks;
        this.specialReqs = specialReqs;
        this.dateIssued = dateIssued;
        this.resolved = resolved;
    }

    public String getRequestID() {
        return requestID;
    }

    public String getPriorityLevel() {
        return priorityLevel;
    }

    public String getDepartment() {
        return department;
    }

    public String getLocation() {
        return location;
    }

    public String getDescription() {
        return description;
    }

    public String getDuration() {
        return duration;
    }

    public String getTasks() {
        return tasks;
    }

    public String getSpecialReqs() {
        return specialReqs;
    }

    public String getDateIssued() {
        return dateIssued;
    }

    public String getResolved() {
        return resolved;
    }

    public String getFileName(){
        return fileName;
    }

    public void setRequestID(String requestID) {
        this.requestID = requestID;
    }

    public void setPriorityLevel(String priorityLevel) {
        this.priorityLevel = priorityLevel;
    }

    public void setDepartment(String department) {
        this.department = department;
    }

    public void setLocation(String location) {
        this.location = location;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public void setDuration(String duration) {
        this.duration = duration;
    }

    public void setTasks(String tasks) {
        this.tasks = tasks;
    }

    public void setSpecialReqs(String specialReqs) {
        this.specialReqs = specialReqs;
    }

    public void setDateIssued(String dateIssued) {
        this.dateIssued = dateIssued;
    }

    public void setResolved(String resolved) {
        this.resolved = resolved;
    }

    public void setFileName(String fileName){
        this.fileName = fileName;
    }
}

/**
 * @author Kenny
 */

package src.Security.src;

import src.TextEditor.PoorTextEditor;

import java.io.*;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.*;

import src.Security.src.SecuritySchedulerController;

public class SecurityScheduler implements
src.Security.src.interfaces.SecurityScheduler {

    PriorityQueue<SecurityRequests> securityRequestsPriorityQueue
= new PriorityQueue<>(
            Comparator.comparingInt(request ->
-Integer.parseInt(request.getPriorityLevel()))
    );
```

```
    PriorityQueue<SecurityRequests>
allSecurityRequestPriorityQueue = new PriorityQueue<>(
            Comparator.comparingInt(request ->
-Integer.parseInt(request.getPriorityLevel()))
    );
    /**
     * Repository to store all security request from departments
     */
    Map<String, SecurityRequests> securityRequestsRepository =
new LinkedHashMap<>();

    // sorts security employees first by division, then position, then by
rating in descending order
    PriorityQueue<SecurityEmployee>
availableSecurityEmployeePriorityQueue = new PriorityQueue<>(
            Comparator.comparing(SecurityEmployee::getDivision)
                    .thenComparing(SecurityEmployee::getPosition)
                    .thenComparing((e1, e2) ->
e2.getRating().compareTo(e1.getRating()))
    );
    PriorityQueue<SecurityEmployee>
allSecurityEmployeePriorityQueue = new PriorityQueue<>(
            Comparator.comparing(SecurityEmployee::getDivision)
                    .thenComparing(SecurityEmployee::getPosition)
                    .thenComparing((e1, e2) ->
e2.getRating().compareTo(e1.getRating()))
    );
    /**
     * Repository to store all security employees obtained
     */
    Map<String, SecurityEmployee> securityEmployeeRepository =
new LinkedHashMap<>();

    // sorts security schedules first by issue date, then by priority level
    PriorityQueue<SecuritySchedules>
securitySchedulesPriorityQueue = new PriorityQueue<>(
            Comparator.comparing((SecuritySchedules schedule) -> {

                DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("MM-dd-yyyy-HH-mm-ss");
                return LocalDateTime.parse(schedule.getDateScheduled(),
formatter);
            }).reversed().thenComparing(schedule ->
-Integer.parseInt(schedule.getPriorityLevel()))
    );
    /**
     * Repository to store all security schedules obtained
     */
    Map<String, SecuritySchedules> securitySchedulesRepository =
new LinkedHashMap<>();

    private final String departmentRequestsDir =
"src/Security/repository/departmentRequests/";
    private final String securitySchedulesDir =
"src/Security/repository/securitySchedules/";
    private final String printedSecuritySchedulesDir =
"src/Security/repository/printedSecuritySchedules/";
    private File[] securityRequestFiles = null;

    private final String securityEmployeeDir =
"src/Security/repository/securityEmployees/";

    private PoorTextEditor editor = new PoorTextEditor();

    @Override
    public boolean addSecurityPersonnel() {
        return false;
    }

    @Override
    public boolean deleteSecurityPersonnel(String employeeID) {
        return false;
    }

    @Override
    public boolean showPendingEvents() {

        if (!retrievePendingRequests()) {
            return false;
        }

        for (SecurityRequests request : securityRequestsPriorityQueue){

            securityRequestToText(request);
        }
        return true;
    }

    @Override
    public boolean showAllEvents() {

        if (!retrieveAllRequests()){
            return false;
        }

        for (SecurityRequests request :
allSecurityRequestPriorityQueue){

            securityRequestToText(request);
        }
        return true;
    }

    @Override
    public boolean showFreeSecurityEmployees() {

        if (!retrieveEmployees()){
            return false;
        }

        for (SecurityEmployee employee :
availableSecurityEmployeePriorityQueue){

            securityEmployeeToText(employee);
        }
        return true;
    }

    @Override
    public boolean showAllSecurityEmployees() {

        if (!retrieveEmployees()){
            return false;
        }

        for (SecurityEmployee employee :
allSecurityEmployeePriorityQueue){

            securityEmployeeToText(employee);
        }
        return true;
    }

    @Override
    public boolean createSchedule(String requestID) {

        if (securityRequestsRepository.isEmpty()){

            if (!retrieveAllRequests()){
                return false;
```

```
            }
        }

        // check if security request ID exists
        if (!securityRequestsRepository.containsKey(requestID)){

            System.out.println("Request ID: " + requestID + " not found");
            return false;
        }

        // check if security request already solved
        if (checkIfRequestResolved(requestID)){

            System.out.println("This security request has already been
resolved. Edit it instead.");
            return false;
        }

        SecurityRequests request =
securityRequestsRepository.get(requestID);
        securityRequestToText(request);

        // creating new security schedule filled with security request
details
        SecuritySchedules schedule = new SecuritySchedules(
                IDGenerator() + "-" + request.getRequestID(),
                request.getRequestID(),
                request.getPriorityLevel(),
                request.getDepartment(),
                request.getLocation(),
                request.getDescription(),
                request.getDuration(),
                request.getTasks(),
                dateIssuer());

        // assigning security personnel to schedule
        if (!addPersonnelToSchedule(schedule)){
            System.out.println("Schedule creation for Request ID: " +
requestID + " was cancelled");
            return false;
        }

        // writing schedule to repository
        editor.setRepository(securityScheduleToHashmap(schedule));
        editor.writeToTextFile(securitySchedulesDir +
schedule.getScheduleID() + ".txt");

        // writing printed schedules to repository
        printSecuritySchedule(schedule);

        // removing specific request from pending request queue
        securityRequestsPriorityQueue.remove(request);
        // removing specific request in all queue
        allSecurityRequestPriorityQueue.remove(request);
        // setting resolve status of security request to true
        request.setResolved("true");
        // adding back changed specific request
        allSecurityRequestPriorityQueue.add(request);

        // editing security resolved status in text file repository
        editor.processTextFile(departmentRequestsDir +
request.getFileName());
        editor.setValue(request.getRequestID(), "resolved", "true");
        editor.writeToTextFile(departmentRequestsDir +
request.getFileName());

        return true;
    }

    @Override
    public boolean deleteSchedule(String scheduleID) {

        if (securityEmployeeRepository.isEmpty()){

            if (!retrieveEmployees()){
                return false;
            }
        }

        if (!retrieveSchedules()){
            return false;
        }

        if (!retrieveAllRequests()){
            return false;
        }

        if (!securitySchedulesRepository.containsKey(scheduleID)){

            System.out.println("Schedule ID: " + scheduleID + " not
found");
            return false;
        }

        SecuritySchedules schedule =
securitySchedulesRepository.get(scheduleID);

        if (!schedule.getAssignedEmployeeIDs().isEmpty()){

            List<String> scheduledEmployeeIDs =
schedule.getAssignedEmployeeIDs();
            editor.processTextFile(securityEmployeeDir +
"securityEmployeeList.txt");

            for (String s : scheduledEmployeeIDs){

                editor.setValue(s, "currentAssignment", "free");
            }

            editor.writeToTextFile(securityEmployeeDir +
"securityEmployeeList.txt");
        }

        // reverting resolved status
        SecurityRequests request =
securityRequestsRepository.get(schedule.getRequestID());
        editor.processTextFile(departmentRequestsDir +
request.getFileName());
        editor.setValue(schedule.getRequestID(), "resolved", "false");
        editor.writeToTextFile(departmentRequestsDir +
request.getFileName());

        File fileToDelete = new File(securitySchedulesDir, scheduleID +
".txt");

        if (fileToDelete.exists()) {

            if (fileToDelete.delete()) {

                System.out.println("Security schedule deleted
successfully");
            } else {

                System.out.println("Failed to security schedule");
                return false;
            }
        } else {
            System.out.println("Security schedule not found");
            return false;
        }
```

```
        return true;
    }

    @Override
    public boolean editScheduleAssignments(String scheduleID) {

        if (!retrieveSchedules()){
            return false;
        }

        if (!securitySchedulesRepository.containsKey(scheduleID)){

            System.out.println("Security schedule with ID: " + scheduleID
+ " does not exists");
            return false;
        }

        SecuritySchedules schedule =
securitySchedulesRepository.get(scheduleID);
        if (!addPersonnelToSchedule(schedule)){
            System.out.println("Updating cancelled");
            return false;
        }

        // writing schedule to repository
        editor.setRepository(securityScheduleToHashmap(schedule));
        editor.writeToTextFile(securitySchedulesDir +
schedule.getScheduleID() + ".txt");

        // writing printed schedules to repository
        printSecuritySchedule(schedule);

        System.out.println("Successfully updated");
        return true;
    }

    @Override
    public boolean showAllSchedules() {

        if (!retrieveSchedules()){
            return false;
        }

        for (SecuritySchedules schedule :
securitySchedulesPriorityQueue){

            securityScheduleToText(schedule);
        }
        return true;
    }

    @Override
    public boolean showScheduleByID(String ScheduleID) {

        if (!retrieveSchedules()){
            return false;
        }

        if (!securitySchedulesRepository.containsKey(ScheduleID)){
            System.out.println("Security schedule with ID: " + ScheduleID
+ "not found");
            return false;
        }


securityScheduleToText(securitySchedulesRepository.get(ScheduleID
));
        return true;
    }

    /**
     * To add security personnel to a schedule
     * @param schedule given schedule
     */
    private boolean addPersonnelToSchedule(SecuritySchedules
schedule){

        if (securityEmployeeRepository.isEmpty()){

            if (!retrieveEmployees()){
                return false;
            }
        }

        List<String> selectedEmployeeIDs = new ArrayList<>();
        if (!schedule.getAssignedEmployeeIDs().isEmpty()){
            selectedEmployeeIDs = schedule.getAssignedEmployeeIDs();
        }
        int employeeCount =
schedule.getAssignedEmployeeIDs().size();
        Scanner scan = new Scanner(System.in);

        boolean endProgram = false;
        boolean returnValue = false;

        while (!endProgram){

            if (Integer.parseInt(schedule.getPriorityLevel()) < 3){

System.out.println(minimumEmployeeRequirement(schedule.getPrior
ityLevel()) - employeeCount + " employees left to add\n");
            }
            else {
                System.out.println("<<Overridden>> employees left to
add");
            }
            System.out.println("1. Add Security Employee by ID");
            System.out.println("2. Remove Security Employee by ID");
            System.out.println("3. Show Scheduled Employees");
            System.out.println("4. Complete Schedule");
            System.out.println("5. Cancel Schedule");
            System.out.println("0. Exit");

            int choice = scan.nextInt();
            scan.nextLine();

            switch (choice){

                case 1:
                    if (employeeCount >=
minimumEmployeeRequirement(schedule.getPriorityLevel())){
                        System.out.println("Exceeded limit");
                        break;
                    }

                    System.out.println("Enter Employee ID: ");
                    String employeeID = scan.nextLine();

                    if
(!securityEmployeeRepository.containsKey(employeeID)){

                        System.out.println("Employee with ID: " + employeeID
+ " does not exists");
                        break;
                    }

                    SecurityEmployee employee =
securityEmployeeRepository.get(employeeID);

                    if (!employee.getCurrentAssignment().equals("free")){
```

```
                    System.out.println("This employee has already been
assigned to :" + employee.getCurrentAssignment() + "\n" +
                            "Overwrite? (y/n)\n" +
                            "!!!WARNING!!! This process is irreversible!");

                    String yesNo = scan.nextLine();

                    if (yesNo.equals("y")){

employee.setPreviousAssignment(employee.getCurrentAssignment()
);

employee.setCurrentAssignment(schedule.getScheduleID());

selectedEmployeeIDs.add(employee.getEmployeeID());

                        // updating employee assignments
                        editor.processTextFile(securityEmployeeDir +
"securityEmployeeList.txt");
                        editor.setValue(employee.getEmployeeID(),
"previousAssignment", employee.getPreviousAssignment());
                        editor.setValue(employee.getEmployeeID(),
"currentAssignment", employee.getCurrentAssignment());
                        editor.writeToTextFile(securityEmployeeDir +
"securityEmployeeList.txt");

                        employeeCount++;
                    }
                }
                else {


availableSecurityEmployeePriorityQueue.remove(employee);

employee.setCurrentAssignment(schedule.getScheduleID());

selectedEmployeeIDs.add(employee.getEmployeeID());

                    // updating employee assignments
                    editor.processTextFile(securityEmployeeDir +
"securityEmployeeList.txt");
                    editor.setValue(employee.getEmployeeID(),
"currentAssignment", employee.getCurrentAssignment());
                    editor.writeToTextFile(securityEmployeeDir +
"securityEmployeeList.txt");

                    employeeCount++;
                }
                break;
            case 2:
                if (selectedEmployeeIDs.isEmpty()){

                    System.out.println("No employees to remove from this
schedule");
                    break;
                }

                System.out.println("Enter Employee ID to remove: ");
                String removingID = scan.nextLine();

                if (!selectedEmployeeIDs.contains(removingID)){
                    System.out.println("Employee ID: " + removingID + "
not assigned to this schedule");
                    break;
                }

                selectedEmployeeIDs.remove(removingID);
                SecurityEmployee employee2 =
securityEmployeeRepository.get(removingID);
                employee2.setCurrentAssignment("free");

availableSecurityEmployeePriorityQueue.add(employee2);

                // updating employee assignments
                editor.processTextFile(securityEmployeeDir +
"securityEmployeeList.txt");
                editor.setValue(employee2.getEmployeeID(),
"currentAssignment", employee2.getCurrentAssignment());
                editor.writeToTextFile(securityEmployeeDir +
"securityEmployeeList.txt");

                employeeCount--;
                break;
            case 3:
                if (selectedEmployeeIDs.isEmpty()){
                    System.out.println("No employees to show from this
schedule");
                    break;
                }

                System.out.println("Selected Employees by ID: ");
                for (String s : selectedEmployeeIDs){
                    System.out.println(s);
                }
                System.out.println();
                break;
            case 4:
                if (employeeCount != 0){

schedule.setAssignedEmployeeIDs(selectedEmployeeIDs);
                    System.out.println("Assignment created");
                    endProgram = true;
                    returnValue = true;
                }
                else {
                    System.out.println("Have at least one assigned
employee");
                }
                break;
            case 5:
                for (String s : selectedEmployeeIDs){

                    SecurityEmployee resetEmployee =
securityEmployeeRepository.get(s);
                    resetEmployee.setCurrentAssignment("free");

availableSecurityEmployeePriorityQueue.add(resetEmployee);

                    // updating employee assignments
                    editor.processTextFile(securityEmployeeDir +
"securityEmployeeList.txt");
                    editor.setValue(resetEmployee.getEmployeeID(),
"currentAssignment", resetEmployee.getCurrentAssignment());
                    editor.writeToTextFile(securityEmployeeDir +
"securityEmployeeList.txt");
                }
                return returnValue;
            case 0:
                if (selectedEmployeeIDs.isEmpty()){
                    endProgram = true;
                }
                else {
                    System.out.println("Remove all assigned employees
or cancel before exiting");
                }
                break;
            default:
                System.out.println("Invalid choice. Try again");
        }
```

```
        }
        return returnValue;
    }

    /**
     * Retrieve files containing security requests
     * @return files containing security requests
     */
    private boolean retrieveRequestFiles(){

        File directory = new File(departmentRequestsDir);
        File[] textFiles = null;

        if (directory.exists() && directory.isDirectory()){

            //grab list of text files
            FilenameFilter textFileFilter = ((dir, name) ->
name.toLowerCase().endsWith(".txt"));
            textFiles = directory.listFiles(textFileFilter);
        }
        else {
            System.out.println("Repository not found");
            return false;
        }

        if (textFiles != null){

            if (textFiles.length == 0){
                System.out.println("No security requests found");
                return false;
            }
            securityRequestFiles = textFiles;
        }
        else {
            System.out.println("No security requests found");
            return false;
        }
        return true;
    }

    /**
     * Retrieve all pending security requests
     * @return true if successful retrieval, otherwise false
     */
    public boolean retrievePendingRequests() {

        if (!retrieveRequestFiles()){
            return false;
        }

        // saving file requests to repository
        for (File f : securityRequestFiles) {

            editor.processTextFile(departmentRequestsDir +
f.getName());
            String[] requests = editor.getArrayNames();

            for (String s : requests) {

                if (!Boolean.parseBoolean(editor.retrieveValue(s,
"resolved"))) {

                    SecurityRequests request = new SecurityRequests(s,
                            editor.retrieveValue(s, "priorityLevel"),
                            editor.retrieveValue(s, "department"),
                            editor.retrieveValue(s, "location"),
                            editor.retrieveValue(s, "description"),
                            editor.retrieveValue(s, "duration"),
                            editor.retrieveValue(s, "tasks"),
                            editor.retrieveValue(s, "specialReqs"),
                            editor.retrieveValue(s, "dateIssued"),
                            editor.retrieveValue(s, "resolved"));

                    request.setFileName(f.getName());

                    if (request.getPriorityLevel() == null ||
request.getDepartment() == null ||
                            request.getLocation() == null ||
request.getDescription() == null ||
                            request.getDuration() == null || request.getTasks()
== null ||
                            request.getSpecialReqs() == null ||
request.getDateIssued() == null) {

                        System.out.println("Invalid security requests
detected");
                        return false;
                    }

                    // adding security request to priority queue
                    securityRequestsPriorityQueue.add(request);
                }
            }
        }
        return true;
    }

    /**
     * Retrieve all security requests
     * @return true if successful retrieval, otherwise false
     */
    private boolean retrieveAllRequests() {

        if (!retrieveRequestFiles()) {
            return false;
        }

        // saving file requests to repository
        for (File f : securityRequestFiles) {

            editor.processTextFile(departmentRequestsDir +
f.getName());
            String[] requests = editor.getArrayNames();

            for (String s : requests) {

                SecurityRequests request = new SecurityRequests(s,
                        editor.retrieveValue(s, "priorityLevel"),
                        editor.retrieveValue(s, "department"),
                        editor.retrieveValue(s, "location"),
                        editor.retrieveValue(s, "description"),
                        editor.retrieveValue(s, "duration"),
                        editor.retrieveValue(s, "tasks"),
                        editor.retrieveValue(s, "specialReqs"),
                        editor.retrieveValue(s, "dateIssued"),
                        editor.retrieveValue(s, "resolved"));

                request.setFileName(f.getName());

                if (request.getPriorityLevel() == null ||
request.getDepartment() == null ||
                        request.getLocation() == null ||
request.getDescription() == null ||
                        request.getDuration() == null || request.getTasks() ==
null ||
                        request.getSpecialReqs() == null ||
request.getDateIssued() == null) {

                    System.out.println("Invalid security requests detected");
                    return false;
```

```
            }

            // adding security request to repository and priority queue
            securityRequestsRepository.put(s, request);
            allSecurityRequestPriorityQueue.add(request);
          }
       }
       return true;
    }

    /**
     * Retrieve all security employees
     * @return all security employees
     */
    private boolean retrieveEmployees(){

       availableSecurityEmployeePriorityQueue.clear();
       allSecurityEmployeePriorityQueue.clear();
       securityEmployeeRepository.clear();

       File directory = new File(securityEmployeeDir);
       File[] textFiles = null;

       if (directory.exists() && directory.isDirectory()){

          //grab list of text files
          FilenameFilter textFileFilter = ((dir, name) ->
name.toLowerCase().endsWith(".txt"));
          textFiles = directory.listFiles(textFileFilter);
       }
       else {
          System.out.println("Repository not found");
          return false;
       }

       if (textFiles != null){

          if (textFiles.length == 0){
             System.out.println("No security employees found");
             return false;
          }
       }
       else {
          System.out.println("No security employees found");
          return false;
       }

       // saving employees to repository
       for (File f : textFiles) {

          editor.processTextFile(securityEmployeeDir + f.getName());
          String[] employees = editor.getArrayNames();

          for (String s : employees) {

             SecurityEmployee employee = new SecurityEmployee(s,
                   editor.retrieveValue(s, "name"),
                   editor.retrieveValue(s, "division"),
                   editor.retrieveValue(s, "position"),
                   editor.retrieveValue(s, "department"),
                   editor.retrieveValue(s, "expertise"),
                   editor.retrieveValue(s, "rating"),
                   editor.retrieveValue(s, "yearsOfExperience"),
                   editor.retrieveValue(s, "previousAssignment"),
                   editor.retrieveValue(s, "currentAssignment"));

             if (employee.getEmployeeID() == null ||
employee.getName() == null ||
                   employee.getDivision() == null ||
employee.getPosition() == null ||
                   employee.getDepartment() == null ||
employee.getExpertise() == null ||
                   employee.getRating() == null ||
employee.getYearsOfExperience() == null ||
                   employee.getPreviousAssignment() == null ||
employee.getCurrentAssignment() == null) {

                System.out.println("Invalid employee data detected");
                return false;
             }

             // adding available employees to priority queue
             if (employee.getCurrentAssignment().equals("free")){

availableSecurityEmployeePriorityQueue.add(employee);
             }
             // adding security request to priority queue
             allSecurityEmployeePriorityQueue.add(employee);
             securityEmployeeRepository.put(s, employee);
          }
       }
       return true;
    }

    /**
     * Retrieves all security schedules
     * @return all security schedules
     */
    private boolean retrieveSchedules(){

       securitySchedulesPriorityQueue.clear();
       securitySchedulesRepository.clear();

       File directory = new File(securitySchedulesDir);
       File[] textFiles = null;

       if (directory.exists() && directory.isDirectory()){

          // grab list of text files
          FilenameFilter textFileFilter = (dir, name) ->
name.toLowerCase().endsWith(".txt");
          textFiles = directory.listFiles(textFileFilter);
       }
       else {
          System.out.println("Repository not found");
          return false;
       }

       if (textFiles != null){

          if (textFiles.length == 0){
             System.out.println("No security schedules found");
             return false;
          }
       }
       else {
          System.out.println("No security schedules found");
          return false;
       }

       for (File f : textFiles){

          editor.processTextFile(securitySchedulesDir + f.getName());
          String[] scheduleIDs = editor.getArrayNames();
          String id = scheduleIDs[0];

          SecuritySchedules schedule = new SecuritySchedules(id,
                editor.retrieveValue(id, "requestID"),
                editor.retrieveValue(id, "priorityLevel"),
                editor.retrieveValue(id, "department"),
```

```java
                editor.retrieveValue(id, "location"),
                editor.retrieveValue(id, "description"),
                editor.retrieveValue(id, "duration"),
                editor.retrieveValue(id, "tasks"),
                editor.retrieveValue(id, "dateScheduled"));

            List<String> employeeIDs = new ArrayList<>();
            int num = 0;
            String employeeNum = "employee" + num;
            while ((editor.retrieveValue(id, employeeNum) != null)){

                employeeIDs.add(editor.retrieveValue(id, "employee" +
num));
                num++;
                employeeNum = "employee" + num;
            }
            schedule.setAssignedEmployeeIDs(employeeIDs);

            if (schedule.getScheduleID() == null ||
schedule.getRequestID() == null ||
                schedule.getPriorityLevel() == null ||
schedule.getDepartment() == null ||
                schedule.getLocation() == null || schedule.getDescription()
== null ||
                schedule.getDuration() == null || schedule.getTasks() ==
null ||
                schedule.getDateScheduled() == null ||
employeeIDs.isEmpty()){

                System.out.println("Invalid schedule data detected");
                return false;
            }

            securitySchedulesPriorityQueue.add(schedule);
            securitySchedulesRepository.put(id, schedule);
        }
        return true;
    }

    /**
     * Pretty print security requests
     * @param request specific security request
     */
    private void securityRequestToText(SecurityRequests request){


System.out.println("=====================================
=================");
        System.out.println("Security Request ID: " +
request.getRequestID());
        System.out.println("Priority Level: " + request.getPriorityLevel());
        System.out.println("Department: " + request.getDepartment());
        System.out.println();
        System.out.println("Location: " + request.getLocation());
        System.out.println("Description: " + request.getDescription());
        System.out.println("Duration: " + request.getDuration());
        System.out.println("Tasks: " + request.getTasks());
        System.out.println("Special Requests: " +
request.getSpecialReqs());
        System.out.println();
        System.out.println("Date Issued: " + request.getDateIssued());
        System.out.println("Resolve Status: " + request.getResolved());

System.out.println("=====================================
=================\n");
    }

    /**
     * Pretty print security employee
     * @param employee specific security employee
     */
    private void securityEmployeeToText(SecurityEmployee
employee){


System.out.println(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>");
        System.out.println("Employee ID: " +
employee.getEmployeeID());
        System.out.println("Name: " + employee.getName());
        System.out.println("Division: " + employee.getDivision());
        System.out.println("Position: " + employee.getPosition());
        System.out.println("Department: " + employee.getDepartment());
        System.out.println("-----------------------------------------------------");
        System.out.println("Expertise: " + employee.getExpertise());
        System.out.println("Rating: " + employee.getRating());
        System.out.println("Years of Experience: " +
employee.getYearsOfExperience());
        System.out.println("Previous Assignment: " +
employee.getPreviousAssignment());
        System.out.println("Current Assignment: " +
employee.getCurrentAssignment());

System.out.println("<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
<<<<<<<<<<<<<<<<<<\n");
    }

    /**
     * Pretty print security schedule
     * @param schedule specific security schedule
     */
    private void securityScheduleToText(SecuritySchedules schedule){


System.out.println("=====================================
==================");
        System.out.println("Schedule ID: " + schedule.getScheduleID());
        System.out.println("is associated with");
        System.out.println("Request ID: " + schedule.getRequestID());
        System.out.println("-----------------------------------------------------");
        System.out.println("Priority Level: " +
schedule.getPriorityLevel());
        System.out.println("Department: " + schedule.getDepartment());
        System.out.println("Location: " + schedule.getLocation());
        System.out.println("Description: " + schedule.getDescription());
        System.out.println("Duration: " + schedule.getDuration());
        System.out.println("Tasks: " + schedule.getTasks());
        System.out.println("Date Scheduled: " +
schedule.getDateScheduled());
        System.out.println("-----------------------------------------------------");
        System.out.println("Assigned Security Members:");
        for (String s : schedule.getAssignedEmployeeIDs()){
            System.out.println(s);
        }

System.out.println("=====================================
==================\n");
    }

    /**
     * Check if a security request has already been resolved
     * @param requestID ID of security request
     * @return true if already resolved, otherwise false
     */
    private boolean checkIfRequestResolved(String requestID){

        return
Boolean.parseBoolean(securityRequestsRepository.get(requestID).g
etResolved());
    }
```

```
    /**
     * Returns current time in MM-dd-yyyy-HH-mm-ss format when
called
     * @return date in MM-dd-yyyy-HH-mm-ss format
     */
    private String dateIssuer(){

        LocalDateTime timeNow = LocalDateTime.now();
        DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("MM-dd-yyyy-HH-mm-ss");

        return timeNow.format(formatter);
    }

    /**
     * Generates IDs for security schedules
     * @return generated security schedule ID
     */
    private String IDGenerator(){

        LocalDateTime timeNow = LocalDateTime.now();
        DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("MMddyyyyHHmmss");

        return timeNow.format(formatter);
    }

    /**
     * Converts SecuritySchedules object into nested HashMap
     * @param schedule specified SecuritySchedule
     * @return HashMap of SecuritySchedule
     */
    private Map<String, Object>
securityScheduleToHashmap(SecuritySchedules schedule){

        Map<String, Object> innerMap = new LinkedHashMap<>();
        innerMap.put("requestID", schedule.getRequestID());
        innerMap.put("priorityLevel", schedule.getPriorityLevel());
        innerMap.put("department", schedule.getDepartment());
        innerMap.put("location", schedule.getLocation());
        innerMap.put("description", schedule.getDescription());
        innerMap.put("duration", schedule.getDuration());
        innerMap.put("tasks", schedule.getTasks());
        innerMap.put("dateScheduled", schedule.getDateScheduled());

        // debug
        //System.out.println("Assigned Employee IDs hashing part: " +
schedule.getAssignedEmployeeIDs());

        int num = 0;
        for (String s : schedule.getAssignedEmployeeIDs()){
            String employee = "employee" + num;
            innerMap.put(employee, s);
            num++;
        }

        Map<String, Object> outerMap = new LinkedHashMap<>();
        outerMap.put(schedule.getScheduleID(), innerMap);

        return outerMap;
    }

    /**
     * To created printable security schedules
     * @param schedule specific security schedule
     */
    private void printSecuritySchedule(SecuritySchedules schedule){

        BufferedWriter writer = null;

        try{
            writer = new BufferedWriter(new
FileWriter(printedSecuritySchedulesDir + schedule.getScheduleID() +
".txt"));

writer.write("==========================================
==========================\n");
            writer.write("SECURITY SCHEDULE\n");
            writer.write("Schedule ID: " + schedule.getScheduleID() + " is
associated with Request ID: " + schedule.getRequestID() + "\n");
            writer.write("Priority Level: " + schedule.getPriorityLevel() + "
(3 = Emergency, 2 = High, 1 = Medium, 0 = Low)\n");
            writer.write("Department: " + schedule.getDepartment() +
"\n");

writer.write("----------------------------------------------------------------------\n");
            writer.write("Location: " + schedule.getLocation() + "\n");
            writer.write("Description: " + schedule.getDescription() + "\n");
            writer.write("Duration :" + schedule.getDuration() + "\n");
            writer.write("Tasks: " + schedule.getTasks() + "\n");
            writer.write("Date Scheduled: " +
schedule.getDateScheduled() + "\n");

writer.write("----------------------------------------------------------------------\n");
            writer.write("Assigned Personnel by IDs:\n");
            for (String s : schedule.getAssignedEmployeeIDs()){
                writer.write(s + "\n");
            }

writer.write("==========================================
==========================\n");
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (writer != null) {
                    writer.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    /**
     * Returns minimum amount of security personnel required for a
given
     * priority level
     * @param priorityLevel given priority level
     * @return minimum number of employees required
     */
    private int minimumEmployeeRequirement(String priorityLevel){

        return switch (priorityLevel) {
            case "3" -> Integer.MAX_VALUE;
            case "2" -> 10;
            case "1" -> 7;
            case "0" -> 3;
            default -> -1;
        };
    }
}

/**
 * @author Kenny
 */

package src.Security.src;

import src.App;
```

```
import java.util.Scanner;

public class SecuritySchedulerController implements
src.Security.src.interfaces.SecuritySchedulerController {

    private SecuritySchedulerController controller;

    private SecurityScheduler scheduler = null;

    public SecuritySchedulerController (SecuritySchedulerController
controller){
        this.controller = controller;
    }

    public SecuritySchedulerController(){}

    /**
     * Run the security program
     */
    public void run() throws Exception {

        System.out.println("Welcome to the On-site Security Scheduler
System");

        boolean exit = false;

        while (!exit) {

            Scanner scan = new Scanner(System.in);

            System.out.println("1. Add New Security Personnel");
            System.out.println("2. Delete Security Personnel by ID");
            System.out.println("3. Show Pending Security Events");
            System.out.println("4. Show All Security Events");
            System.out.println("5. Show Available Security Employees");
            System.out.println("6. Show All Security Employees");
            System.out.println("7. Create Security Schedule");
            System.out.println("8. Delete Security Schedule");
            System.out.println("9. Edit Existing Security Schedule Team
Assignment");
            System.out.println("10. Show All Security Schedules");
            System.out.println("11. Show Security Schedule by ID");
            System.out.println("0. Exit program");

            int choice = scan.nextInt();
            scan.nextLine();

            switch (choice) {

                case 1:
                    addSecurityPersonnel();
                    System.out.println();
                    break;
                case 2:
                    System.out.println("Enter Security Employee ID to
delete: ");
                    String deleteID = scan.nextLine();
                    deleteSecurityPersonnel(deleteID);
                    System.out.println();
                    break;
                case 3:
                    showPendingEvents();
                    System.out.println();
                    break;
                case 4:
                    showAllEvents();
                    System.out.println();
                    break;
                case 5:
                    showFreeSecurityEmployees();
                    System.out.println();
                    break;
                case 6:
                    showAllSecurityEmployees();
                    System.out.println();
                    break;
                case 7:
                    System.out.println("Enter Security Request/Event ID to
create a schedule for : ");
                    String requestID = scan.next();
                    createSchedule(requestID);
                    System.out.println();
                    break;
                case 8:
                    System.out.println("Enter Security Schedule ID to delete
: ");
                    String deleteScheduleID = scan.next();
                    deleteSchedule(deleteScheduleID);
                    System.out.println();
                    break;
                case 9:
                    System.out.println("Enter Security Schedule ID to edit :
");
                    String editScheduleID = scan.next();
                    editScheduleAssignments(editScheduleID);
                    System.out.println();
                    break;
                case 10:
                    showAllSchedules();
                    System.out.println();
                    break;
                case 11:
                    System.out.println("Enter Security Schedule ID to view :
");
                    String viewScheduleID = scan.next();
                    showScheduleByID(viewScheduleID);
                    System.out.println();
                    break;
                case 0:
                    System.out.println("Closing program...");
                    exit = true;
                    App.prompt();
                default:
                    System.out.println("Incorrect choice. Try again");
            }
        }
    }

    @Override
    public boolean addSecurityPersonnel() {
        return this.getSchedulerInstance().addSecurityPersonnel();
    }

    @Override
    public boolean deleteSecurityPersonnel(String employeeID) {
        return
this.getSchedulerInstance().deleteSecurityPersonnel(employeeID);
    }

    @Override
    public boolean showPendingEvents() {
        return this.getSchedulerInstance().showPendingEvents();
    }

    @Override
    public boolean showAllEvents() {
        return this.getSchedulerInstance().showAllEvents();
    }
```

```
    @Override
    public boolean showFreeSecurityEmployees() {
        return
this.getSchedulerInstance().showFreeSecurityEmployees();
    }

    @Override
    public boolean showAllSecurityEmployees() {
        return this.getSchedulerInstance().showAllSecurityEmployees();
    }

    @Override
    public boolean createSchedule(String requestID) {
        return this.getSchedulerInstance().createSchedule(requestID);
    }

    @Override
    public boolean deleteSchedule(String scheduleID) {
        return this.getSchedulerInstance().deleteSchedule(scheduleID);
    }

    @Override
    public boolean editScheduleAssignments(String scheduleID) {
        return
this.getSchedulerInstance().editScheduleAssignments(scheduleID);
    }

    @Override
    public boolean showAllSchedules() {
        return this.getSchedulerInstance().showAllSchedules();
    }

    @Override
    public boolean showScheduleByID(String ScheduleID) {
        return
this.getSchedulerInstance().showScheduleByID(ScheduleID);
    }

    /**
     * To define one instance of the class SecurityScheduler to
     * call SecurityScheduler methods
     * @return SecurityScheduler instance
     */
    private SecurityScheduler getSchedulerInstance(){

        if (scheduler == null){
            scheduler = new SecurityScheduler();
        }
        return scheduler;
    }
}

/**
 * @author Kenny
 */

package src.Security.src;

import java.util.ArrayList;
import java.util.List;

public class SecuritySchedules {

    private String scheduleID, requestID, priorityLevel, department,
            location, description, duration, tasks, dateScheduled;
    private List<String> assignedEmployeeIDs = new ArrayList<>();

    public SecuritySchedules(){}

    public SecuritySchedules (String scheduleID, String requestID,
String priorityLevel,
                              String department, String location, String
description,
                              String duration, String tasks, String
dateScheduled){
        this.scheduleID = scheduleID;
        this.requestID = requestID;
        this.priorityLevel = priorityLevel;
        this.department = department;
        this.location = location;
        this.description = description;
        this.duration = duration;
        this.tasks = tasks;
        this.dateScheduled = dateScheduled;
    }

    public String getScheduleID(){
        return scheduleID;
    }

    public String getRequestID(){
        return requestID;
    }

    public String getPriorityLevel(){
        return priorityLevel;
    }

    public String getDepartment() {
        return department;
    }

    public String getLocation(){
        return location;
    }

    public String getDescription(){
        return description;
    }

    public String getDuration(){
        return duration;
    }

    public String getTasks(){
        return tasks;
    }

    public String getDateScheduled(){
        return dateScheduled;
    }

    public List<String> getAssignedEmployeeIDs(){
        return assignedEmployeeIDs;
    }

    public void setScheduleID(String scheduleID){
        this.scheduleID = scheduleID;
    }

    public void setRequestID(String requestID){
        this.requestID = requestID;
    }

    public void setPriorityLevel(String priorityLevel){
        this.priorityLevel = priorityLevel;
    }

    public void setDepartment(String department){
```

```
        this.department = department;
    }

    public void setLocation(String location){
        this.location = location;
    }

    public void setDescription(String description){
        this.description = description;
    }

    public void setDuration(String duration){
        this.duration = duration;
    }

    public void setTasks(String tasks){
        this.tasks = tasks;
    }

    public void setDateScheduled(String dateScheduled){
        this.dateScheduled = dateScheduled;
    }

    public void setAssignedEmployeeIDs(List<String>
assignedEmployeeIDs){
        this.assignedEmployeeIDs = assignedEmployeeIDs;
    }
}
```

Sales [Mani Raj]

```
public class BasicRetailer implements Retailer {
        private String name;
        private String location;

        private Map<String,String> rDetails=new HashMap<>();

        public BasicRetailer(String name, String location) {
                this.name = name;
                this.location = location;
                rDetails.put("name",name);
                rDetails.put("retailerLocation", location);
        }

        public Map<String,String> getRDetails()
        {
                return  rDetails;
        }

        public String getName() {
                return name;
        }

        public void setName(String name) {
                rDetails.put("name",name);
        }
        public String getLocation() {
                return location;
        }

        public void setLocation() {
                rDetails.put("retailerLocation", location);
        }

        public void print() {

System.out.println("-------------------------------------------------------");
                System.out.println("Retailer Information");
                for (Map.Entry<String, String> rd :
rDetails.entrySet()) {

                        System.out.printf("%-30s %s%n",
rd.getKey(), rd.getValue());

                }

System.out.println("-------------------------------------------------------");
        }
}

package src.Sales.src;
```

```
import src.Inventory.src.BasicStorageManage;
import src.Inventory.src.interfaces.StorageManagement;
import src.Sales.SalesCommand;
import src.Sales.src.interfaces.SalesController;
import src.Sales.src.interfaces.SalesManagement;

import java.util.Map;
import java.util.Scanner;

public class BasicSalesController implements SalesController {
    SalesManagement sale;
    SalesInventoryRequest sir=new SalesInventoryRequest();

    public void run()
    {
        SalesCommand scd=new SalesCommand();

        System.out.println("Welcome to the Sales Management
System");

        System.out.println("Enter 'help' to see available commands");
        System.out.println("Enter 'exit' to leave the sales Management
System");
        System.out.println("Enter the command for your task");

        Scanner sc =new Scanner(System.in);
        System.out.println("Enter command: ");
        String command=sc.nextLine();
        scd.perform(command);

    }

    public void changePrice(int price, String pname)
    {
        sir.changePrice(price,pname);
    }

    public void registerRetailer(String name, String location) {
        getSalesInstance().registerRetailer(name, location);
    }


    public void addOrder(int rid, Map<String, Integer> products) {
        getSalesInstance().addOrder(rid, products);
    }

    public void printReceipt(int orderId)
    {
        getSalesInstance().printReceipt(orderId);
```

```
    }
    public void returnOrder(int rid, int oid, Map<String, Integer>
rproducts, int days)
    {
        getSalesInstance().returnOrder(rid,oid,rproducts,days);
    }

    public void viewRetailers()
    {
        sale.viewRetailers();
    }

    public void viewAvaProducts()
    {
        sir.viewAvaProducts();
    }

    public void viewRegisteredProducts()
    {
        sir.viewProducts();
    }


    private SalesManagement getSalesInstance() {
        if (sale == null) {
            sale = new BasicSalesManage();
        }
        return sale;
    }

}

public class BasicSalesManage implements SalesManagement {

    // id to details
    private Map<String, Map<String, String>> retailers;

   // id to details
    private Map<String, Map<String, String>> orders;

    private SalesInventoryRequest sir;
    private String location;
    private String repoPath = "src/Sales/repository/";

    PoorTextEditor textEditor;

    public BasicSalesManage() {
        this.retailers = new HashMap<>();
        this.orders = new HashMap<>();
        location = "Main";

        set();
    }

    private void set() {
        String rPath = repoPath + "Retailers.txt";
        String oPath= repoPath+ "Orders.txt";

        try {
            File rFile = new File(rPath);
            if (rFile.exists()) {
                textEditor = new PoorTextEditor();
                textEditor.processTextFile(rPath);
                retailers = textEditor.getRepositoryString();
            }
        } catch (Exception e) {
            System.out.println("Error processing Retailers.txt");

        }
        try {
            File oFile = new File(rPath);
            if (oFile.exists()) {
                textEditor = new PoorTextEditor();
                textEditor.processTextFile(oPath);
                orders = textEditor.getRepositoryString();
            }
        } catch (Exception e) {
            System.out.println("Error processing Retailers.txt");
        }
    }

    public void addOrder(int rid, Map<String, Integer> products) {

        int size= orders.size();
        sir =new SalesInventoryRequest();
        String id = String.valueOf(rid);
        if (retailers.containsKey(id)) {

          Map<String,String> odetail= sir.addPToOrder(products);
          odetail.put("retailer id", String.valueOf(rid));
          orders.put(String.valueOf(size+1),odetail);

           System.out.println("order placed by id: " + size+1);

          System.out.println("use command to print receipt if needed
using the order id");
        } else {
           System.out.println("Retailer id does not exist. Register the
retailer!");
        }


    }

    public void returnOrder(int rid, int oid, Map<String, Integer>
rproducts, int days) {
        String rId = String.valueOf(rid);
        String oId = String.valueOf(oid);
        if (retailers.containsKey(rId)) {

            if (orders.containsKey(oId)) {
                if(checkReturnTime(days,oid))
                {
                 sir.removePToOrder(rproducts);
                 System.out.println("return successful and products sent to
inventory");
                }else{
                    System.out.println("Order passes return period co cannot
be returned");
                }

            } else {
                System.out.println("Order id does not exist!");
            }
        } else {
            System.out.println("Retailer id does not exist!");
        }
    }

    public void registerRetailer(String name, String retailerLocation) {

        Retailer rd = new BasicRetailer(name, retailerLocation);
        String size = String.valueOf(retailers.size() + 1);

        retailers.put(size, rd.getRDetails());

        textEditor = new PoorTextEditor();
        textEditor.setRepositoryStrings(retailers);
        textEditor.writeToTextFile(repoPath + "Retailers.txt");
        rd.print();
```

```
        System.out.println("Retailer registered with id:" + size);
    }

    public boolean checkReturnTime(int days, int orderId) {
        return days <= 15;
    }


    public void printReceipt(int orderId)
    {
        Map<String, String> odetails = new
HashMap<>(orders.get(String.valueOf(orderId)));
        sir.receiptPrint(odetails);
    }


    public void viewRetailers()
    {
        print(retailers, "Registered Retailers");
    }

    public void print(Map<String, Map<String, String>> items, String
whichItem) {
        System.out.println("-------------------------------------------------------");
        System.out.println(whichItem);
        for (Map.Entry<String, Map<String, String>> itms :
items.entrySet()) {
            System.out.println(itms.getKey() + ":");
            for (Map.Entry<String, String> sitm :
itms.getValue().entrySet()) {
                System.out.printf("%-30s %s%n", sitm.getKey(),
sitm.getValue());
            }
        }
        System.out.println("-------------------------------------------------------");
    }
}

public class Order {
                private int id;
                private Map<String, String> productList; // productId
mapped to quantity

                private String retailerName;

                private Map<String, Object> orderDetails=new
HashMap<>();

                public Order(int id,String retailerName) {
                                this.id = id;
                                this.retailerName=retailerName;
                                this.productList = new HashMap<>();
                }

                public void setOrderDetails()
                {
                                orderDetails.put(retailerName,productList);
                }

                public void addProduct(String pName, int quantity) {
                                productList.put(pName,
String.valueOf(quantity));
                }

                public void deleteProduct(String pName) {
                                productList.remove(pName);
                }

                public  Map<String, Object> getOrderDetails()
                {
                                return orderDetails;
                }

                public Map<String, String> getProductList() {
                                return productList;
                }

                public int getId() {

                                return id;
                }

}

public class SalesInventoryRequest {

    Map<String, Map<String, String>> products;
    private Map<String, Map<String, String>> availableProducts;
    BasicStorageManage sm = new BasicStorageManage();

    public SalesInventoryRequest() {

        products = sm.getProducts();
        availableProducts = sm.getAvailableProducts();
    }

    public void removePQ(String pname, int quantityRemoved) {
        sm.removeProductCount(pname, quantityRemoved);
    }

    public void addPQ(String pname, int quantityRemoved) {
        sm.addProductCount(pname, quantityRemoved);
    }

    public void changePrice(int price, String pname) {
        sm.changePrice(price, pname);
    }

    public int getPrice(String pname) {
        return
Integer.parseInt(availableProducts.get(pname).get("price"));
    }

    public boolean checkAvailability(Map<String, Integer>
orderProducts) {

        for (Map.Entry<String, Integer> op : orderProducts.entrySet()) {
            String pname = op.getKey();
            if (availableProducts.containsKey(pname)) {
                int avaCount =
Integer.parseInt(availableProducts.get(op.getKey()).get("count"));
                if (avaCount < op.getValue()) {
                    System.out.println("Insufficient quantity, available
quantity is " + avaCount + " for " + pname);
                }
            } else {
                System.out.println(pname + " not available currently");
            }
        }
        return true;
    }


    public Map<String, String> addPToOrder(Map<String, Integer>
products) {
        Map<String, String> cOrder = new HashMap<>();

        if (checkAvailability(products)) {
            String pname;
```

```
            int quan;
            for (Map.Entry<String, Integer> p : products.entrySet()) {
                pname = p.getKey();
                quan = p.getValue();
                cOrder.put(pname, String.valueOf(quan));
                removePQ(pname, quan);
            }
        }
        return cOrder;
    }

    public void removePToOrder(Map<String, Integer> products) {
        String pname;
        int quan;
        for (Map.Entry<String, Integer> p : products.entrySet()) {
            pname = p.getKey();
            quan = p.getValue();
            addPQ(pname, quan);
        }
    }

    public void receiptPrint(Map<String, String> odetails) {
        String rid = odetails.remove("retailer id");
        int total = 0;

        String pname;
        String quan;
        int price;
        int ptotal;
        System.out.println("-----------------------------------------------------");
        System.out.printf("%60s %n", "Receipt");
        System.out.println("-----------------------------------------------------");
        System.out.println("Retailer id :" + rid);
        System.out.printf("%-30s %-30s %-30s %s%n", "product name",
"quantity", "per quantity", "total");
        for (Map.Entry<String, String> itms : odetails.entrySet()) {
            pname = itms.getKey();
            quan = itms.getValue();
            price = getPrice(pname);
            ptotal = price * Integer.parseInt(quan);
            total += ptotal;
            System.out.printf("%-30s %-30s %-30d %d%n", pname, quan,
price, ptotal);
        }

        System.out.println("Total Amount :" + total);
        System.out.println("-----------------------------------------------------");
    }


    public void viewAvaProducts() {
        print(availableProducts, "Available products at Storage");
    }

    public void viewProducts() {
        print(products, "registered products at Storage");
    }

    public void print(Map<String, Map<String, String>> items, String
whichItem) {
        System.out.println("-----------------------------------------------------");
        System.out.println(whichItem);
        for (Map.Entry<String, Map<String, String>> itms :
items.entrySet()) {
            System.out.println(itms.getKey() + ":");
            for (Map.Entry<String, String> sitm :
itms.getValue().entrySet()) {
                System.out.printf("%-30s %s%n", sitm.getKey(),
sitm.getValue());
            }
        }
        System.out.println("-----------------------------------------------------");
    }

}

public class SalesCommand {
    Map<String, Runnable> commands = new HashMap<>();
    SalesController sc = new BasicSalesController();
    private void setCommands() {
        Scanner scan = new Scanner(System.in);

        commands.put("Register Retailer", () -> {
            System.out.println("Enter Retailer Name:");
            String retailerName = scan.nextLine();
            System.out.println("Enter Retailer Location:");
            String retailerLocation = scan.nextLine();
            sc.registerRetailer(retailerName, retailerLocation);
        });
        commands.put("Add Order", () -> {
            System.out.println("Enter Retailer id:");
            int rid = scan.nextInt();
            scan.nextLine();
            Map<String, Integer> products = new HashMap<>();
            boolean flag = true;
            while (flag) {
                System.out.println("Enter product name:");
                String pname = scan.nextLine();
                System.out.println("Enter quantity");
                int quan = scan.nextInt();
                scan.nextLine();
                products.put(pname, quan);

                System.out.println("Do you want to still add (yes/no):");
                String response = scan.nextLine();

                if (response.equals("no")) {
                    flag = false;
                }
            }
            sc.addOrder(rid, products);
        });
        commands.put("Print Receipt", () -> {
            System.out.println("Enter Order id:");
            int oid = scan.nextInt();
            scan.nextLine();
            sc.printReceipt(oid);
        });
        commands.put("Change Price", () -> {
            System.out.println("Enter Product Name:");
            String pname = scan.nextLine();
            System.out.println("Enter new price");
            int price = scan.nextInt();
            scan.nextLine();

            sc.changePrice(price, pname);
        });
        commands.put("View Retailers", () -> {
            sc.viewRetailers();
        });
        commands.put("View Registered Products", () -> {
            sc.viewRegisteredProducts();
        });
        commands.put("View Available Products", () -> {
            sc.viewAvaProducts();
        });
        commands.put("Return Products", () -> {
            System.out.println("Enter Retailer id:");
            int rid = scan.nextInt();
            scan.nextLine();
```

```
            System.out.println("Enter order id:");
            int oid = scan.nextInt();
            scan.nextLine();
            System.out.println("How many days has it been since the
purchase? :");
            int days = scan.nextInt();
            scan.nextLine();
            Map<String, Integer> products = new HashMap<>();
            boolean flag = true;
            while (flag) {
                System.out.println("Enter product name:");
                String pname = scan.nextLine();
                System.out.println("Enter quantity");
                int quan = scan.nextInt();
                scan.nextLine();
                products.put(pname, quan);

                System.out.println("Do you want to still add (yes/no):");
                String response = scan.nextLine();

                if (response.equals("no")) {
                    flag = false;
                }
            }
            sc.returnOrder(rid, oid, products, days);
        });


        commands.put("help", () -> {
            System.out.println(" 'Register Retailer', 'Add Order', 'Print
Receipt', 'Change Price', " + "'View Retailers', " +
                    "'View Registered Products','View Available Products',
'Return Products' ");
        });

        commands.put("exit", () -> {
            try {
                App.prompt();
            } catch (Exception e) {

            }
        });
    }
    public void perform(String command) {
        Runnable action = commands.get(command.trim());
        if (action != null) {
            action.run();
        } else {
            System.out.println("Unknown command: " + command);
        }
    }
}
```

Design [Doyle Chism]

```
package src.Design.src;

import src.Design.src.interfaces.DesignSpecifications;
import src.Design.src.interfaces.HeadOfDesignInterface;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class CustomDesign implements DesignSpecifications {

    private String designName;
    private String designImage;
    private List<String> colors;
    private List<String> rawMaterials;
    private List<String> sizes;
    private int quantity;

    public CustomDesign(String designName) {
        this.designName = designName;
    }

    @Override
    public void setColor(List<String> colors) {
        this.colors = colors;
    }

    @Override
    public void setRawMaterials(List<String> rawMaterials) {
        this.rawMaterials = rawMaterials;
    }

    @Override
    public void setSizes(List<String> sizes) {
        this.sizes = sizes;
    }

    @Override
    public void setQuantities(int quantities) {
        this.quantity = quantities;
    }

    @Override
    public void setDesignName(String designName) {
        this.designName = designName;
    }

    @Override
    public void setDesignImage(String image) {
        this.designImage = image;
    }

    @Override
    public List<String> getColors() {
        return colors;
    }

    @Override
    public List<String> getRawMaterials() {
        return rawMaterials;
    }

    @Override
    public List<String> getSizes() {
        return sizes;
    }

    @Override
    public int getQuantities() {
        return quantity;
    }

    @Override
    public String getDesignName() {
        return designName;
    }

    @Override
    public String getDesignImage() {
        return designImage;
    }

    @Override
    public String displayAllSpecifications() {
        return
                "Display all design specifications\n" +
                        "Design Name: " + designName + "\n" +
                        "Design Image: " + designImage + "\n" +
                        "Design Colors: " + colors + "\n" +
                        "Design Raw Materials: " + rawMaterials + "\n" +
                        "Design Sizes: " + sizes + "\n" +
                        "Design Quantities: " + quantity;

    }

    @Override
    public Map<String, Object> mapObjects() {
        Map<String, Object> map = new HashMap<>();
        map.put("DesignName", designName);
        map.put("DesignImage", designImage);
        map.put("DesignColors", String.join(",", colors));
        map.put("DesignRawMaterials", String.join(",", rawMaterials));
        map.put("DesignSizes", String.join(",", sizes));
        map.put("DesignQuantities", quantity);

        return map;
    }
}


package src.Design.src;

import src.TextEditor.PoorTextEditor;

import java.io.File;
import java.util.HashMap;
import java.util.Map;

public class DesignFileManager {

    private Map<String, Object> sketches = new HashMap<>();
    private Map<String, Object> finalDesign = new HashMap<>();
    private Map<String, Object> customDesign = new HashMap<>();
    private Map<String, Object> marketingDesign = new
HashMap<>();


    private final PoorTextEditor editor = new PoorTextEditor();

    private final String repo = "src/Design/repository/";

    public void sendDataToRepo() {
        File f = new File(repo + "sketches.txt");
        if (f.exists()) {
            editor.processTextFile(repo + "sketches.txt");
            sketches = editor.getRepository();
```

```
        }
        f = new File(repo + "finalDesign.txt");
        if (f.exists()) {
            editor.processTextFile(repo + "finalDesign.txt");
            finalDesign = editor.getRepository();
        }
        f = new File(repo + "customDesign.txt");
        if (f.exists()) {
            editor.processTextFile(repo + "customDesign.txt");
            customDesign = editor.getRepository();
        }
        f = new File(repo + "marketingDesign.txt");
        if (f.exists()) {
            editor.processTextFile(repo + "marketingDesign.txt");
            marketingDesign = editor.getRepository();

        }
    }

    public void addSketch(DesignSketch sketch) {
        sketches.put("Sketches " + sketch.getDesignName(),
sketch.mapObjects());

        editor.setRepository(sketches);
        editor.writeToTextFile(repo + "sketches.txt");
    }

    /*
    public Map<String, Map<String, String>> getSketches() {
        // Return the sketches map directly
        return new HashMap<>(sketches);
    }
     */
    public Map<String, Object> getSketches() {

        Map<String, Object> newSketch = new HashMap<>();
        for (Map.Entry<String, Object> entry : sketches.entrySet()) {
            newSketch.put(entry.getKey(), entry.getValue());
        }
        return newSketch;

    }

    public void addFinalDesign(FinalDesign design) {
        finalDesign.put("Final Design " + design.getDesignName(),
design.mapObjects());

        editor.setRepository(finalDesign);
        editor.writeToTextFile(repo + "finalDesign.txt");
    }

    public Map<String, Object> getFinalDesign() {

        Map<String, Object> newDesign = new HashMap<>();
        for (Map.Entry<String, Object> entry : finalDesign.entrySet()) {
            newDesign.put(entry.getKey(), entry.getValue());
        }
        return newDesign;
    }

    public void addCustomDesign(CustomDesign design) {
        customDesign.put("Custom Design " + design.getDesignName(),
design.mapObjects());

        editor.setRepository(customDesign);
        editor.writeToTextFile(repo + "customDesign.txt");
    }

    public Map<String, Object> getCustomDesign() {
        Map<String, Object> newDesign = new HashMap<>();
        for (Map.Entry<String, Object> entry : customDesign.entrySet())
{
            newDesign.put(entry.getKey(), entry.getValue());
        }
        return newDesign;
    }

    public void addMarketingDesign(MarketingDesign design) {
        customDesign.put("Marketing Design " +
design.getDesignSketchName(), design.mapObjects());
        editor.setRepository(customDesign);
        editor.writeToTextFile(repo + "marketingDesign.txt");
    }

    public Map<String, Object> getMarketingDesign() {
        Map<String, Object> newDesign = new HashMap<>();
        for (Map.Entry<String, Object> entry : customDesign.entrySet())
{
            newDesign.put(entry.getKey(), entry.getValue());
        }
        return newDesign;
    }

}
```

```
package src.Design.src;

import src.Design.src.interfaces.DesignSpecifications;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class DesignSketch implements DesignSpecifications {

    private String designName;
    private String designImage;
    private List<String> colors;
    private List<String> rawMaterials;
    private List<String> sizes;
    private int quantity;

    public DesignSketch(String designName) {
        this.designName = designName;
    }

    @Override
    public void setColor(List<String> colors) {

        this.colors = colors;
    }

    @Override
    public void setRawMaterials(List<String> rawMaterials) {
        this.rawMaterials = rawMaterials;
    }

    @Override
    public void setSizes(List<String> sizes) {
        this.sizes = sizes;
    }

    @Override
    public void setQuantities(int quantities) {
        this.quantity = quantities;
    }

    @Override
    public void setDesignName(String designName) {
        this.designName = designName;
```

```
    }

    @Override
    public void setDesignImage(String image) {
        this.designImage = image;
    }

    @Override
    public List<String> getColors() {
        return colors;
    }

    @Override
    public List<String> getRawMaterials() {
        return rawMaterials;
    }

    @Override
    public List<String> getSizes() {
        return sizes;
    }

    @Override
    public int getQuantities() {
        return quantity;
    }

    @Override
    public String getDesignName() {
        return designName;
    }

    @Override
    public String getDesignImage() {
        return designImage;
    }

    @Override
    public String displayAllSpecifications() {
        return
                "Display all design specifications\n" +
                        "Design Name: " + designName + "\n" +
                        "Design Image: " + designImage + "\n" +
                        "Design Colors: " + colors + "\n" +
                        "Design Raw Materials: " + rawMaterials + "\n" +
                        "Design Sizes: " + sizes + "\n" +
                        "Design Quantities: " + quantity;
    }

    @Override
    public Map<String, Object> mapObjects() {
        Map<String, Object> map = new HashMap<>();
        map.put("DesignName", designName);
        map.put("DesignImage", designImage);
        map.put("DesignColors", String.join(",", colors));
        map.put("DesignRawMaterials", String.join(",", rawMaterials));
        map.put("DesignSizes", String.join(",", sizes));
        map.put("DesignQuantities", quantity);

        return map;
    }
}
```

```
package src.Design.src;


//import src.App;

import src.Design.src.interfaces.DesignSpecifications;
import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.Scanner;

public class DesignSpecificationsController {


    private HeadOfDesignTeam headOfDesignTeam;
    private List<DesignSketch> sketches;
    private FinalDesign finalDesign;
    private DesignSketch sketch;
    private CustomDesign customDesign;
    private MarketingDesign marketingDesign;
    private final DesignFileManager designFileManager = new
DesignFileManager();

    public DesignSpecificationsController() {

        sketches = new ArrayList<>();
        headOfDesignTeam = new HeadOfDesignTeam(sketches); //
they ask for sketches

    }


    public void run() throws Exception {


        Scanner scan = new Scanner(System.in);
        boolean exit = false;
        System.out.println("Welcome to the Design Management
System.");

        while (!exit) {
            System.out.println("1. Create a Design Sketch");
            System.out.println("2. View and Select a Design Sketch");
            System.out.println("3. Set a Final Design for Manufacturing");
            System.out.println("4. Set a Final Design for Marketing");
            System.out.println("5. Set a custom Final Design for
Modelling");
            System.out.println("6. Create a Marketing Design based on
Final Design");
            System.out.println("7. Send Designs to Repository.");
            System.out.println("8. Exit Program");

            int option = scan.nextInt();
            scan.nextLine();

            switch (option) {
                case 1:
                    System.out.println("Begin Listing the Specifications for
the design sketch.");
                    System.out.println("If entering multiple entries input as
such S,M,L with no spaces");
                    System.out.println("Enter Design Name: ");
                    String designName = scan.nextLine();
                    System.out.println("Enter raw materials needed: ");
                    String rawMaterialsNeeded = scan.nextLine();
                    System.out.println("Enter the colors you want for design:
");
                    String colorsNeeded = scan.nextLine();
                    System.out.println("Enter sizes for design: ");
                    String sizesNeeded = scan.nextLine();
                    System.out.println("Enter design  quantity: ");
                    int designQuantity = scan.nextInt();
                    scan.nextLine();
```

```
                System.out.println("Enter image of the design: ");
                String imageOfDesign = scan.nextLine();
                //call to helper to set all specifications
                createDesignSketch(designName, rawMaterialsNeeded,
colorsNeeded, sizesNeeded, designQuantity, imageOfDesign);

                break;
            case 2:
                headOfDesignTeam.viewSketches(sketches);
                Map<String, Object> storedSketches =
designFileManager.getSketches();
                if (sketches.isEmpty() || storedSketches.isEmpty()) {
                    System.out.println("No sketches found.");
                    break;
                }
                System.out.println("Available Design Sketches: ");
                int i = 0;
                for (String sketchName : storedSketches.keySet()) {
                    System.out.println((i + 1) + "." + sketchName);
                }
                //ask for input
                System.out.println("Select the Design Sketch to verify");
                int sketchNumber = scan.nextInt() - 1;
                scan.nextLine();

                if (sketchNumber < 0 || sketchNumber >=
sketches.size()) {
                    System.out.println("Invalid sketch number");
                    break;
                }

                String selectedSketchName = new
ArrayList<>(storedSketches.keySet()).get(sketchNumber);
                Map<String, Object> selectedSketchDetails =
(Map<String, Object>) storedSketches.get(selectedSketchName);

                selectedSketchDetails.forEach((key, value) -> {
                    System.out.println(key + ": " + value);
                });

                headOfDesignTeam.selectSketch(sketchNumber,
sketches);
                System.out.println("(Y/N) Do you want to verify the
design sketch?");
                String verifyDesign = scan.nextLine();
                if (verifyDesign.equals("Y")) {
                    //call to begin Final Design case 3

                    FinalDesign selectedSketch =
headOfDesignTeam.confirmFinalDesign();
                    if (selectedSketch != null) {
                        System.out.println("Sketch Verfied\nDisplay
specifications:\n" + selectedSketch.displayAllSpecifications());
                        finalDesign = selectedSketch;
                    } else {
                        System.out.println("No sketch has been selected.
Try again.");
                    }
                } else if (verifyDesign.equals("N")) {
                    System.out.println("Design was not verified");
                } else {
                    System.out.println("Invalid Input. Select Y or N");
                }
                break;
            case 3:

                if (finalDesign == null) {
                    System.out.println("No Final Design has been set.
Verify a design first");
                    break;
                } else {
                    System.out.println("Design was verified by the Head of
Design");
                }
                System.out.println("Current Final Design Specifications:
\n" + finalDesign.displayAllSpecifications());
                System.out.println("Do you want to modify the current
final design?");
                System.out.println("(Y/N)");
                String modifyDesign = scan.nextLine();
                if (modifyDesign.equals("Y")) {
                    System.out.println("Set a Final Design and set
specifications of the final design");
                    System.out.println("Enter Design Name: ");
                    String finalDesignName = scan.nextLine();
                    System.out.println("Enter raw materials needed: ");
                    String finalRawMaterialsNeeded = scan.nextLine();
                    System.out.println("Enter the colors you want for
design: ");
                    String finalColorsNeeded = scan.nextLine();
                    System.out.println("Enter sizes for design: ");
                    String finalSizesNeeded = scan.nextLine();
                    System.out.println("Enter design  quantity: ");
                    int finalDesignQuantity = scan.nextInt();
                    scan.nextLine();
                    System.out.println("Enter image of the design: ");
                    String finalImageOfDesign = scan.nextLine();
                    setFinalDesign(finalDesignName,
finalRawMaterialsNeeded, finalColorsNeeded, finalSizesNeeded,
finalDesignQuantity, finalImageOfDesign);
                } else if (modifyDesign.equals("N")) {
                    System.out.println("No changes are needed in the
Final Design");
                    break;
                } else {
                    System.out.println("Invalid Input. Select Y or N");
                }
            case 4:
                if (finalDesign == null) {
                    System.out.println("No Final Design has been set.
Verify a design first");
                    break;
                } else {
                    System.out.println("Design was verified by the Head of
Design");
                }
                System.out.println("Current Final Design Specifications:
\n" + finalDesign.displayAllSpecifications());
                System.out.println("Do you want to modify the current
final design?");
                System.out.println("(Y/N)");
                String modifyDesign2 = scan.nextLine();
                if (modifyDesign2.equals("Y")) {
                    System.out.println("Set a Final Design and set
specifications of the final design");
                    System.out.println("Enter Design Name: ");
                    String finalDesignName = scan.nextLine();
                    System.out.println("Enter raw materials needed: ");
                    String finalRawMaterialsNeeded = scan.nextLine();
                    System.out.println("Enter the colors you want for
design: ");
                    String finalColorsNeeded = scan.nextLine();
                    System.out.println("Enter sizes for design: ");
                    String finalSizesNeeded = scan.nextLine();
                    System.out.println("Enter design  quantity: ");
                    int finalDesignQuantity = scan.nextInt();
                    scan.nextLine();
                    System.out.println("Enter image of the design: ");
                    String finalImageOfDesign = scan.nextLine();
```

```
                    setFinalDesign(finalDesignName,
finalRawMaterialsNeeded, finalColorsNeeded, finalSizesNeeded,
finalDesignQuantity, finalImageOfDesign);
                } else if (modifyDesign2.equals("N")) {
                    System.out.println("No changes are needed in the
Final Design");
                    break;
                } else {
                    System.out.println("Invalid Input. Select Y or N");
                }
            case 5:
                if (customDesign == null) {
                    System.out.println("No Final Design has been set.
Verify a design first");
                    break;
                } else {
                    System.out.println("Design was verified by the Head of
Design");
                }
                System.out.println("Current Final Design Specifications:
\n" + customDesign.displayAllSpecifications());
                System.out.println("Do you want to modify the current
final design?");
                System.out.println("(Y/N)");
                String response = scan.nextLine();
                if (response.equals("Y")) {
                    System.out.println("Set a Final Design and set
specifications of the final design");
                    System.out.println("Enter Design Name: ");
                    String finalDesignName = scan.nextLine();
                    System.out.println("Enter raw materials needed: ");
                    String finalRawMaterialsNeeded = scan.nextLine();
                    System.out.println("Enter the colors you want for
design: ");
                    String finalColorsNeeded = scan.nextLine();
                    System.out.println("Enter sizes for design: ");
                    String finalSizesNeeded = scan.nextLine();
                    System.out.println("Enter design  quantity: ");
                    int finalDesignQuantity = scan.nextInt();
                    scan.nextLine();
                    System.out.println("Enter image of the design: ");
                    String finalImageOfDesign = scan.nextLine();
                    //have to change to set custom design
                    setCustomDesign(finalDesignName,
finalRawMaterialsNeeded, finalColorsNeeded, finalSizesNeeded,
finalDesignQuantity, finalImageOfDesign);
                } else if (response.equals("N")) {
                    System.out.println("No changes are needed in the
Final Design");
                    break;
                } else {
                    System.out.println("Invalid Input. Select Y or N");
                }

            case 6:
                if (marketingDesign == null) {
                    System.out.println("No Design has been set. Verify a
design first");
                    break;
                } else {
                    System.out.println("Design was verified by the Head of
Design");
                }
                System.out.println("Current Final Design Specifications:
\n" + finalDesign.displayAllSpecifications());
                System.out.println("Begin setting marketing Design
based on the current final Design.");
                MarketingDesign currentMarketingDesign = new
MarketingDesign(finalDesign);
                System.out.println("Final Design Name is: " +
currentMarketingDesign.getDesignSketchName());
                String marketingName = scan.nextLine();
                System.out.println("Enter design price: ");
                String price = scan.nextLine();
                System.out.println("Enter target Audience: ");
                String targetAudience = scan.nextLine();
                System.out.println("Enter design description: ");
                String description = scan.nextLine();
                System.out.println("Enter season Type: ");
                String seasonType = scan.nextLine();

                setMarketingDesign(marketingName, targetAudience,
price, description, seasonType);
                System.out.println("Marketing Design has been set");

            case 7:
                //sending to manufacturing
//                designFileManager.addFinalDesign(finalDesign);
//                sendToFinalDesign();
                designFileManager.sendDataToRepo();

            case 8:
                System.out.println("Exit Program");
                exit = true;
//                App.prompt();
                break;

            default:
                System.out.println("Invalid option");
        }
    }
    scan.close();
  }

  //lists all the things needed to create a new sketch which will then
be reviewed
  public void createDesignSketch(String designName, String
rawMaterialsNeeded, String colorsNeeded, String sizesNeeded, int
designQuantity, String imageOfDesign) {

    List<String> rawMaterials =
List.of(rawMaterialsNeeded.split(","));
    List<String> colors = List.of(colorsNeeded.split(","));
    List<String> sizes = List.of(sizesNeeded.split(","));
    //make sure I am calling this correctly
    sketch = new DesignSketch(designName);
    sketch.setDesignName(designName);
    sketch.setRawMaterials(rawMaterials);
    sketch.setColor(colors);
    sketch.setSizes(sizes);
    sketch.setQuantities(designQuantity);
    sketch.setDesignImage(imageOfDesign);
    sketches.add(sketch);
    sketch.displayAllSpecifications();
    designFileManager.addSketch(sketch);
    System.out.println("Sketch created and added to a list of
sketches waiting to be approved");

  }

  public void setFinalDesign(String finalDesignName, String
finalRawMaterialsNeeded, String finalColorsNeeded, String
finalSizesNeeded, int finalDesignQuantity, String finalImageOfDesign)
{

    List<String> rawMaterials =
List.of(finalRawMaterialsNeeded.split(","));
    List<String> colors = List.of(finalColorsNeeded.split(","));
    List<String> sizes = List.of(finalSizesNeeded.split(","));
```

```
        finalDesign = new FinalDesign(finalDesignName);

        finalDesign.setDesignName(finalDesignName);
        finalDesign.setRawMaterials(rawMaterials);
        finalDesign.setColor(colors);
        finalDesign.setSizes(sizes);
        finalDesign.setQuantities(finalDesignQuantity);
        finalDesign.setDesignImage(finalImageOfDesign);
        designFileManager.addFinalDesign(finalDesign);
        System.out.println("Final Design has been set with all
specifications");
        finalDesign.displayAllSpecifications();

    }

    public void setCustomDesign(String customDesignName, String
rawMaterialsNeeded, String colorsNeeded, String sizeNeeded, int
designQuantity, String imageOfDesign) {

        List<String> rawMaterials =
List.of(rawMaterialsNeeded.split(","));
        List<String> colors = List.of(colorsNeeded.split(","));
        List<String> sizes = List.of(sizeNeeded.split(","));

        customDesign = new CustomDesign(customDesignName);
        customDesign.setRawMaterials(rawMaterials);
        customDesign.setColor(colors);
        customDesign.setSizes(sizes);
        customDesign.setQuantities(designQuantity);
        customDesign.setDesignImage(imageOfDesign);
        customDesign.displayAllSpecifications();
        designFileManager.addCustomDesign(customDesign);
        System.out.println("Custom Design has been set with all
specifications");
        customDesign.displayAllSpecifications();

    }

    public void setMarketingDesign(String sketch, String
targetAudience, String price, String description, String seasonType) {

        marketingDesign = new MarketingDesign(finalDesign);
        marketingDesign.setDesignSketchName(sketch);
        marketingDesign.setPrice(price);
        marketingDesign.setProductDescription(description);
        marketingDesign.setSeasonType(seasonType);
        marketingDesign.setTargetAudience(targetAudience);
        designFileManager.addMarketingDesign(marketingDesign);
        System.out.println("MarketingDesign has been set with all
specifications");
        marketingDesign.displayAllSpecifications();

    }


}
```

```
package src.Design.src;

import src.Design.src.interfaces.DesignSpecifications;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class FinalDesign implements DesignSpecifications {

    private String designName;
    private String designImage;
    private List<String> colors;
    private List<String> rawMaterials;
    private List<String> sizes;
    private int quantity;


    public FinalDesign(String designName) {
        this.designName = designName;
    }


    @Override
    public void setColor(List<String> colors) {
        this.colors = colors;
    }

    @Override
    public void setRawMaterials(List<String> rawMaterials) {
        this.rawMaterials = rawMaterials;
    }

    @Override
    public void setSizes(List<String> sizes) {
        this.sizes = sizes;
    }

    @Override
    public void setQuantities(int quantities) {
        this.quantity = quantities;
    }

    @Override
    public void setDesignName(String designName) {
        this.designName = designName;
    }

    @Override
    public void setDesignImage(String image) {
        this.designImage = image;
    }

    @Override
    public List<String> getColors() {
        return colors;
    }

    @Override
    public List<String> getRawMaterials() {
        return rawMaterials;
    }

    @Override
    public List<String> getSizes() {
        return sizes;
    }

    @Override
    public int getQuantities() {
        return quantity;
    }

    @Override
    public String getDesignName() {
        return designName;
    }

    @Override
    public String getDesignImage() {
        return designImage;
    }
```

```
    @Override
    public String displayAllSpecifications() {

        return
                "Design Name: " + designName + "\n" +
                        "Design Image: " + designImage + "\n" +
                        "Design Colors: " + colors + "\n" +
                        "Design Raw Materials: " + rawMaterials + "\n" +
                        "Design Sizes: " + sizes + "\n" +
                        "Design Quantities: " + quantity;
    }

    @Override
    public Map<String, Object> mapObjects() {
        Map<String, Object> map = new HashMap<>();
        map.put("DesignName", designName);
        map.put("DesignImage", designImage);
        map.put("DesignColors", String.join(",", colors));
        map.put("DesignRawMaterials", String.join(", ", rawMaterials));
        map.put("DesignSizes", String.join(", ", sizes));
        map.put("DesignQuantities", quantity);
        return map;
    }
}


package src.Design.src;

import src.Design.src.interfaces.*;

import java.util.List;

public class HeadOfDesignTeam implements HeadOfDesignInterface
{


    private List<DesignSketch> allSketches;
    private DesignSketch selectedSketch;
    private CustomDesign customDesign;
    private MarketingDesign marketingDesign;
    private FinalDesign finalDesign;


    public HeadOfDesignTeam(List<DesignSketch> sketches) {
        this.allSketches = sketches;
    }


    @Override
    public void viewSketches(List<DesignSketch> sketches) {

        if (sketches.isEmpty()) {
            System.out.println("No sketch selected");
            return;
        }
        System.out.println("Select a sketch");
        for (int i = 0; i < sketches.size(); i++) {
            System.out.println((i + 1) + ". " +
sketches.get(i).getDesignName());
        }
    }

    @Override
    public void selectSketch(int sketchIndex, List<DesignSketch>
sketches) {

        if (sketchIndex >= 0 && sketchIndex < sketches.size()) {
            selectedSketch = sketches.get(sketchIndex);
            System.out.println("Selcted sketch was: " +
selectedSketch.getDesignName());
        } else {
            System.out.println("Incorrect Sketch selected, Try Again");
        }
    }

    @Override
    public FinalDesign confirmFinalDesign() {

        if (selectedSketch == null) {
            System.out.println("No sketch selected");
            return null;
        }
        FinalDesign finalDesign = new
FinalDesign(selectedSketch.getDesignName());

        finalDesign.setDesignName(selectedSketch.getDesignName());
        finalDesign.setColor(selectedSketch.getColors());
        finalDesign.setDesignImage(selectedSketch.getDesignImage());
        finalDesign.setSizes(selectedSketch.getSizes());
        finalDesign.setQuantities(selectedSketch.getQuantities());

finalDesign.setRawMaterials(selectedSketch.getRawMaterials());
        finalDesign.displayAllSpecifications();

        return finalDesign;
    }

    @Override
    public MarketingDesign confirmMarketingDesign() {

        if (marketingDesign == null) {
            System.out.println("No marketing design selected");
            return null;
        }
        MarketingDesign market = new MarketingDesign(finalDesign);

        market.setDesignSketchName(finalDesign.getDesignName());
        market.setPrice(marketingDesign.getPrice());

market.setProductDescription(marketingDesign.getProductDescriptio
n());
        market.setSeasonType(marketingDesign.getSeasonType());

market.setTargetAudience(marketingDesign.getTargetAudience());
        market.displayAllSpecifications();

        return market;
    }

    @Override
    public CustomDesign confirmCustomDesign() {

        if (customDesign == null) {
            System.out.println("No custom design selected");
            return null;
        }
        CustomDesign custom = new
CustomDesign(customDesign.getDesignName());

        custom.setDesignName(customDesign.getDesignName());
        custom.setColor(customDesign.getColors());
        custom.setDesignImage(customDesign.getDesignImage());
        custom.setSizes(customDesign.getSizes());
        custom.setQuantities(customDesign.getQuantities());
        custom.setRawMaterials(customDesign.getRawMaterials());
        custom.displayAllSpecifications();

        return custom;
```

```
    }

}


package src.Design.src;

import src.Design.src.interfaces.MarketingDesignSpecifications;

import java.util.HashMap;
import java.util.Map;

public class MarketingDesign implements
MarketingDesignSpecifications {

    private FinalDesign finalDesign;
    private String targetAudience;
    private String price;
    private String description;
    private String seasonType;

    public MarketingDesign(FinalDesign finalDesign) {
        this.finalDesign = finalDesign;

    }

    @Override
    public String getDesignSketchName() {
        return finalDesign.getDesignName();
    }

    @Override
    public void setDesignSketchName(String designSketchName) {
        this.finalDesign.setDesignName(designSketchName);
    }


    @Override
    public void setTargetAudience(String targetAudience) {
        this.targetAudience = targetAudience;

    }

    @Override
    public String getTargetAudience() {
        return targetAudience;
    }

    @Override
    public void setPrice(String price) {
        this.price = price;

    }

    @Override
    public String getPrice() {
        return price;
    }

    @Override
    public void setSeasonType(String seasonType) {
        this.seasonType = seasonType;
    }

    @Override
    public String getSeasonType() {
        return seasonType;
    }

    @Override
    public void setProductDescription(String productDescription) {
        this.description = productDescription;
    }

    @Override
    public String getProductDescription() {
        return description;
    }

    @Override
    public String displayAllSpecifications() {
        return
                "Display all specifications \n" +
                        "Design sketch " + finalDesign + "\n" +
                        "Target Audience " + targetAudience + "\n"
                        + "Price " + price + "\n"
                        + "Season type " + seasonType + "\n"
                        + "Product description " + description + "\n";
    }

    @Override
    public Map<String, Object> mapObjects() {
        Map<String, Object> map = new HashMap<>();
        map.put("sketch", finalDesign);
        map.put("targetAudience", targetAudience);
        map.put("price", price);
        map.put("seasonType", seasonType);

        return map;
    }

}
```

HR [Sam]

```
package src.HR.src;

import src.HR.src.interfaces.ICandidate;

/**
 * @author Sam Gumm
 */

public class Candidate implements ICandidate {
                String candidateId;
                String name;
                String positionApplied;
                candidateStatus status;

                /**
                * @param candidateId the Candidate's identifying number
                * @param name the Candidates name
                * @param positionApplied the position the Candidate has
applied to
                * @param status  the Status of the Candidate in the
Hiring process
                */
                public Candidate(String candidateId, String name, String
positionApplied, candidateStatus status) {
                this.candidateId = candidateId;
                this.name = name;
                this.positionApplied = positionApplied;
                this.status = status;
                }
```

```
	/**
	*
	* @return name
	*/
	@Override
	public String getName() {
	return name;
	}

	/**
	*
	* @return candidateId
	*/
	@Override
	public String getCandidateId() {
	return candidateId;
	}

	/**
	*
	* @return positionApplied
	*/
	@Override
	public String getPositionApplied() {
	return positionApplied;
	}

	/**
	* @return status
	*/
	@Override
	public candidateStatus getStatus() {
	return status;
	}

	/**
	*
	* @param status the Status can be 5 things:
	*		Applied --> they have submitted an
application, but not had an interview
	*		Pending --> they are waiting on an interview
	*		Approved --> they have had an interview and
are approved for next stage
	*		Rejected --> they have been ultimately
rejected at any stage
	*		Hired --> they are set to be turned into
employees
	*/
	@Override
	public void setStatus(String status) {

	if(status == null) {
	throw new NullPointerException("Status cannot be null");
	}

	//TODO: use enumeration here (maybe?)
	else if (status.equalsIgnoreCase("APPLIED") ||
status.equalsIgnoreCase("Pending") ||
status.equalsIgnoreCase("Approved") ||
status.equalsIgnoreCase("Rejected") ||
status.equalsIgnoreCase("Hired")) {
	this.status = candidateStatus.valueOf(status);
	}
	else {
	throw new IllegalArgumentException("Status must be:
Applied, Pending, Approved, Rejected, Hired");
	}
	}
	/**
	* @return String
	*/
	@Override
	public String toString() {
	return String.format("ID: %s, Name: %s, Position Applied:
%s, Status: %s",
	candidateId, name, positionApplied, status);
	}
}

package src.HR.src;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.nio.file.DirectoryStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.StandardCopyOption;
import java.util.*;

/**
 * Manages Candidate records, allowing for adding, removing,
updating, moving and displaying Candidate.
 * @author Sam Gumm
 */
public class candidateRecordManager {
	Map<String, Map<String, String>> data = new
LinkedHashMap<>();
	private final fileStorageHR storageHR;

	public candidateRecordManager(fileStorageHR
storageHR) {
	this.storageHR = storageHR;
	}


	/**
	* Add a new candidate into a txt file in format "candID".txt
and stores it to candidateStorage
	* @param candidate the Candidate object to be stored to
file
	*/
	public void addCandidate(Candidate candidate) throws
IOException {
	Map<String, String> candidateObject = new
LinkedHashMap<>();
	candidateObject.put("candidateID",
candidate.getCandidateId());
	candidateObject.put("name", candidate.getName());
	candidateObject.put("positionApplied",
candidate.getPositionApplied());
	candidateObject.put("status",
candidate.getStatus().toString());
	data.put(candidate.getCandidateId(), candidateObject);
	storageHR.poorJarser.setRepositoryStrings(data);
	//TODO: Print out all potential statuses here
	String filepathToCandidateStorage =
String.valueOf(storageHR.getCandidateStoragePath(String.valueOf(c
andidate.getStatus())));

storageHR.poorJarser.writeToTextFile(filepathToCandidateStorage +
"/" + candidate.getCandidateId() + ".txt");
	data.clear();
	}

	/**
	* Remove a candidate by ID
	* @param candidateID ID of the Candidate file to remove
	*
```

```
		*/
		public void removeCandidate(String candidateID) throws
Exception {
		if(data.containsKey(candidateID)) {
		try {
		data.remove(candidateID);
		storageHR.poorJarser.setRepositoryStrings(data);
		} catch (Exception e) {
		throw new Exception("Error in removeCandidate when
removing found ID from LinkedHashMap<>(): \n" + e.getMessage());
		}
		}

		String filePath =
String.valueOf(findCandidateFile(candidateID));
		if(filePath == null) {
		throw new NullPointerException("filepath is null");
		}

		try {
		storageHR.deleteFile(filePath);
		} catch (Exception e) {
		throw new Exception("Error in removeCandidate when
deleting file from repository with file path: "
		+ filePath + "\n" + e.getMessage());
		}
		}

		public Path findCandidateFile(String candidateID) throws
IOException {
		Path base =
storageHR.getDefault_filepath_candidateStorage();
		String candidateFileName = candidateID + ".txt"; // The
candidate file name format

		try (DirectoryStream<Path> directoryStream =
Files.newDirectoryStream(base)) {
		for (Path statusDir : directoryStream) {
		if (Files.isDirectory(statusDir)) { // Ensure it's a directory
		Path candidateFile =
statusDir.resolve(candidateFileName);
		if (Files.exists(candidateFile)) {
				return candidateFile;
		}
		}
		}
		}
		return null; // Return null if the candidate file is not found
		}



		private void moveCandidate(String candidateID,
candidateStatus status) throws IOException {
		// Find the current candidate file path
		Path currentCandidateFile =
findCandidateFile(candidateID);
		if (currentCandidateFile == null ||
!Files.exists(currentCandidateFile)) {
		throw new FileNotFoundException("Candidate file not
found: " + candidateID + ".txt");
		}

		// Get the path to the new status folder
		Path newStatusFolder =
storageHR.getCandidateStoragePath(status.toString());
		if (!Files.exists(newStatusFolder)) {
		Files.createDirectories(newStatusFolder);
		}

		// Construct the destination path for the candidate file
		Path destinationFile =
newStatusFolder.resolve(candidateID + ".txt");

		// Move the candidate file to the new status folder
		Files.move(currentCandidateFile, destinationFile,
StandardCopyOption.REPLACE_EXISTING);

		System.out.println("Candidate was successfully moved to
" + destinationFile);
		}

		/**
		* Takes the CandidateID of the Candidate object to be
changed, then checks candidateStorage
		* for the ID. If found, the filepath is returned, and the
candidateObject is parsed, altered, and
		* moved to the new Status folder if necessary.
		* @param candidateId ID of the candidate to be updated
		*/
		public void updateCandidate(String candidateId) throws
Exception {
		String filepath;

		//As findCandidateFolder may return null, we need to
make sure that doesn't happen
		filepath =
Objects.requireNonNull(findCandidateFile(candidateId)).toString();
		System.out.println("updateCandidate filepath from finding:
" + filepath);

		//read from Candidate file
		storageHR.poorJarser.processTextFile(filepath);

		//initialize data from current repo
		Map<String, Map<String, String>> data =
storageHR.poorJarser.getRepositoryStringMap();

		//make sure data is not null
		//TODO: is this necessary?
		if(data == null) {
		throw new Exception("data was not initialized properly:
\n");
		}

		//initialize candidateObject to modify
		Map<String, String> candidateObject =
data.get(candidateId);
		if(candidateObject == null) {
		candidateObject = new LinkedHashMap<>();
		}
		Scanner scanner = new Scanner(System.in);

		System.out.println("Enter candidate name: ");
		String name = scanner.next();
		System.out.println("Enter position applied: ");
		String positionApplied = scanner.next();
		System.out.println("Enter in new candidate hiring status:
\nApplied\nPending\nApproved\nRejected\nHiring\n Enter Here: ");
		candidateStatus candidateStatus =
src.HR.src.candidateStatus.valueOf(scanner.next().toUpperCase());
		candidateObject.put("candidateName", name);
		candidateObject.put("positionApplied", positionApplied);
		candidateObject.put("candidateStatus",
String.valueOf(candidateStatus));
		data.put(candidateId, candidateObject);
		storageHR.poorJarser.setRepositoryStrings(data);
		moveCandidate(candidateId, candidateStatus);
		storageHR.poorJarser.writeToTextFile(filepath);
		storageHR.deleteFile(filepath);
		}
```

```
	public void displayCandidatesByStatus(String canStatus)
throws Exception {
	try {
	String statusDirName = canStatus.toUpperCase();

	Path statusDir =
storageHR.getCandidateStoragePath(statusDirName);

	if(statusDir == null) {
	throw new NullPointerException("statusDir is null");
	}

	if(!Files.exists(statusDir) || !Files.isDirectory(statusDir)) {
	System.out.println("No candidates found with status: " +
canStatus);
	return; // Add return to exit the method if directory doesn't
exist
	}

	try (DirectoryStream<Path> directoryStream =
Files.newDirectoryStream(statusDir, "*.txt")) {
	boolean candidatesFound = false;
	for (Path path : directoryStream) {
	candidatesFound = true;
	storageHR.loadFileAndPrint(path.toString()); // Pass full
path as string
	}
	if(!candidatesFound) {
	System.out.println("No candidates found with status: " +
canStatus);
	}
	}
	} catch (Exception e) {
	throw new Exception("displayCandidatesByStatus failed:
\n" + e.getMessage(), e);
	}
	}



}

package src.HR.src;

public enum candidateStatus {
	APPLIED,
	APPROVED,
	HIRING,
	PENDING,
	REJECTED;

	@Override
	public String toString() {
	return switch (this) {
	case APPLIED -> "Applied";
	case APPROVED -> "Approved";
	case HIRING -> "Hiring";
	case PENDING -> "Pending";
	case REJECTED -> "Rejected";
	};
	}
}

package src.HR.src;

/**
 * @author Sam Gumm
 */
public enum Department {
	ENGINEERING,
	MARKETING,
	HUMAN_RESOURCES,
	FINANCE,
	DESIGN,
	MODELING,
	MANUFACTURING;

	@Override
	public String toString() {
	return switch (this) {
	case ENGINEERING -> "Engineering";
	case MARKETING -> "Marketing";
	case HUMAN_RESOURCES -> "Human_Resources";
	case FINANCE -> "Finance";
	case DESIGN -> "Design";
	case MODELING -> "Modeling";
	case MANUFACTURING -> "Manufacturing";
	};
	}
}

package src.HR.src;

import src.HR.src.interfaces.IEmployee;

/**
 * @author Sam Gumm
 */
public class Employee implements IEmployee {
	String employeeId;
	String name;
	Department department;
	String position;
	String employmentStatus;
	int salary;

	/**
	*
	* @param employeeId String, the Employee's identifying
number
	* @param name String the Employee's name
	* @param department Department, the Employee's
assigned department
	* @param position String, the Employee's assigned
position
	* @param employmentStatus String, the Employee's
current employment status:
	*			Onboarding --> Just hired from
Candidate
	*			Active --> Finished Onboarding
tasks, full employee
	*			Offboarding --> Pending being
deleted from system (Employee has been fired)
	* @param salary int, the Employee's recorded salary
	*/
	public Employee(String employeeId, String name,
Department department, String position, String employmentStatus, int
salary) {
	this.employeeId = employeeId;
	this.name = name;
	this.department = department;
	this.position = position;
	this.employmentStatus = employmentStatus;
	this.salary = salary;
	}

	/**
	*
```

```
	* @return name
	*/
	@Override
	public String getName() {
	return name;
	}

	/**
	*
	* @return employeeId
	*/
	@Override
	public String getEmployeeID() {
	return employeeId;
	}

	/**
	* @return department
	*/
	@Override
	public String getDepartment() {
	return String.valueOf(department);
	}

	/**
	*
	* @return position
	*/
	@Override
	public String getPosition() {
	return position;
	}

	/**
	*
	* @return employmentStatus
	*/
	@Override
	public String getEmployementStatus() {
	return employmentStatus;
	}

	/**
	* @return salary
	*/
	@Override
	public String getSalary() {
	return String.valueOf(salary);
	}

	/**
	* @return String
	*/
	@Override
	public String toString() {
	return String.format("ID: %s, Name: %s, Department: %s,
Position: %s, Status: %s, Salary: %d",
	employeeId, name, department, position,
employmentStatus, salary);
	}

	public static Employee parseEmployee(String
employeeString) {
	// Remove unnecessary spaces and split the string by
commas
	String[] parts = employeeString.split(", ");

	// Extract the individual fields using the known format
	String employeeId = parts[0].split(": ")[1];
	String name = parts[1].split(": ")[1];
	Department department =
Department.valueOf(parts[2].split(": ")[1].toUpperCase()); // Assuming
Department is an enum
	String position = parts[3].split(": ")[1];
	String employmentStatus = parts[4].split(": ")[1];
	int salary = Integer.parseInt(parts[5].split(": ")[1]);

	// Create and return the Employee object
	return new Employee(employeeId, name, department,
position, employmentStatus, salary);
	}
}
```

```
package src.HR.src;


import java.io.*;
import java.nio.file.DirectoryStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.StandardCopyOption;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.Objects;
import java.util.Scanner;

/**
 * @author Sam Gumm
 */
public class employeeRecordManager {
	Map<String, Map<String, String>> data = new
LinkedHashMap<>();
	private final fileStorageHR storageHR;

	/*
	TODO:
	- make methods to transfer employees to new folders
	- implement this in updateEmployee
	*/

	public employeeRecordManager(fileStorageHR
storageHR) {
	this.storageHR = storageHR;
	}

	/**
	* Adds an Employee to record, then persistently stores it
in "empID".txt
	*
	* @param employee the Employee object to be added
	*/
	// Add a new employee
	public void addEmployee(Employee employee) throws
IOException {
	//TODO: have counter that increments for employeeID
	Map<String, String> employeeObject = new
LinkedHashMap<>();
	employeeObject.put("name", employee.getName());
	employeeObject.put("employeeID",
employee.getEmployeeID());
	employeeObject.put("employeeDepartment",
employee.getDepartment());
	employeeObject.put("employeePosition",
employee.getPosition());
	employeeObject.put("employeeStatus",
employee.getEmployementStatus());
	employeeObject.put("employeeSalary",
employee.getSalary());
	data.put(employee.getEmployeeID(), employeeObject);
	storageHR.poorJarser.setRepositoryStrings(data);
```

```
            String filepathToEmployeeStorage =
String.valueOf(storageHR.getEmployeeStoragePath(String.valueOf(e
mployee.getDepartment())));

storageHR.poorJarser.writeToTextFile(filepathToEmployeeStorage +
"/" + employee.getEmployeeID() + ".txt");
            }

            /**
            * Removes employee FILE from employeeStorage repo,
uses fileStorageHR filepath to find file.
            * Also attempts to remove Employee from
LinkedHashMap record if it exists there as well.
            *
            * @param employeeID the EmployeeID to be removed
            *
            */
            public void removeEmployee(String employeeID) throws
Exception {
            //remove employee from data LinkedHashMap if it
contains the id
            if(data.containsKey(employeeID)) {
            try {
            data.remove(employeeID);
            storageHR.poorJarser.setRepositoryStrings(data);
            } catch (Exception e) {
            throw new Exception("Error in removeEmployee when
removing found ID from LinkedHashMap<>(): \n" + e.getMessage());
            }
            }

            String filepath =
String.valueOf(findEmployeeFile(employeeID));
            if(filepath == null) {
            throw new NullPointerException("filepath from
removeEmployee is null");
            }

            try {
            storageHR.deleteFile(filepath);

            } catch (Exception e) {
            throw new Exception("Error in removeEmployee when
deleting file from repository with file path: "
            + filepath + "\n" + e.getMessage());
            }
            }


            private void moveEmployee(String employeeID,
Department department) throws IOException {
            // Find the current employee file path
            Path currentEmployeeFile =
findEmployeeFile(employeeID);
            if (currentEmployeeFile == null ||
!Files.exists(currentEmployeeFile)) {
            throw new FileNotFoundException("Employee file not
found: " + employeeID + ".txt");
            }

            // Get the path to the new Department folder
            Path newDepartmentFolder =
storageHR.getEmployeeStoragePath(department.toString().toUpperC
ase());
            if (!Files.exists(newDepartmentFolder)) {
            throw new FileNotFoundException("Department folder not
found: " + department.toString().toUpperCase());
            }

            // Construct the destination path for the candidate file
            Path destinationFile =
newDepartmentFolder.resolve(employeeID + ".txt");

            // Move the candidate file to the new status folder
            Files.move(currentEmployeeFile, destinationFile,
StandardCopyOption.REPLACE_EXISTING);

            System.out.println("Employee was successfully moved to
" + destinationFile);
            }
            /**
            * Takes the EmployeeID of the Employee object to be
changed, then checks the LinkedHashMap
            * for the ID; if it finds the ID, the associated Employee is
altered and re-uploaded to file,
            * otherwise, an error is thrown.
            *
            * @param employeeID the associated ID of the Employee
            */
            public void updateEmployee(String employeeID) throws
Exception {
            String filepath;

            filepath =
Objects.requireNonNull(findEmployeeFile(employeeID)).toString();

            //read from Employee file
            storageHR.poorJarser.processTextFile(filepath);

            //initialize data from current repo
            Map<String, Map<String, String>> data =
storageHR.poorJarser.getRepositoryStringMap();

            //make sure data is not null
            //TODO: is this necessary?
            if(data == null) {
            throw new Exception("data was not initialized properly:
\n");
            }

            //initialize candidateObject to modify
            Map<String, String> employeeObject =
data.get(employeeID);
            if(employeeObject == null) {
            employeeObject = new LinkedHashMap<>();
            }
            removeEmployee(employeeID);
            Scanner scanner = new Scanner(System.in);

            for(int i = 0; i < Department.values().length; i++) {
            System.out.println(Department.values()[i].name());
            }
            System.out.println("Enter new employee Department: ");
            Department department =
src.HR.src.Department.valueOf(scanner.next().toUpperCase());

            System.out.println("Enter in new employee position: ");
            String position = scanner.next();

            System.out.println("Enter in new employee Status: ");
            String status = scanner.next();

            System.out.println("Enter in new employee salary: ");
            String salary = scanner.next();

            employeeObject.put("employeeID", employeeID);
            employeeObject.put("name",
employeeObject.get("name"));
            employeeObject.put("department", department.toString());
            employeeObject.put("position", position);
```

```
        employeeObject.put("status", status);
        employeeObject.put("salary", salary);

        data.put(employeeID, employeeObject);


        storageHR.poorJarser.setRepositoryStrings(data);
        storageHR.poorJarser.writeToTextFile(filepath);
        moveEmployee(employeeID, department);
        }

        /**
        * Display all records currently in the LinkedHashMap
"data"
        */
        public void displayCurrentEmployeeRecords() {
        data.values().forEach(System.out::println);
        }

        //TODO: move this to fileStorageHR
        public void displayFileRecords(String folderPath) throws
Exception {
        File folder = new File(folderPath);
        //TODO: add direct path to employeeStorage
        if(folder.isDirectory()) {
        File[] files = folder.listFiles();
        int i = 0;
        do {
        assert files != null;
        File file = files[i];
        if(file.isFile() && file.getName().endsWith(".txt")) {
        try (BufferedReader br = new BufferedReader(new
FileReader(file))) {
                String line;
                while ((line = br.readLine()) != null) {
                System.out.println(line);
                }
        } catch (IOException e) {
                throw new RuntimeException(e);
        }
        }
        i++;
        } while (i < files.length);
        }
        else {
        throw new Exception("File is not a directory");
        }
        }



        /**
        * @param departmentName the Department to iterate
through.
        */
        public void displayEmployeesByDepartment(Department
departmentName) throws Exception {
        try {
        String statusDirName =
String.valueOf(departmentName).toUpperCase();

        Path statusDir =
storageHR.getEmployeeStoragePath(statusDirName);

        if(statusDir == null) {
        throw new NullPointerException("statusDir is null");
        }

        if(!Files.exists(statusDir) || !Files.isDirectory(statusDir)) {
                System.out.println("No candidates found with status: " +
departmentName);
                return;
                }

                try (DirectoryStream<Path> directoryStream =
Files.newDirectoryStream(statusDir, "*.txt")) {
                boolean employeesFound = false;
                for (Path path : directoryStream) {
                employeesFound = true;
                storageHR.loadFileAndPrint(path.toString()); // Pass full
path as string
                }
                if(!employeesFound) {
                System.out.println("No candidates found with status: " +
departmentName);
                }
                }
                } catch (Exception e) {
                throw new Exception("displayEmployeesByDepartment
failed: \n" + e.getMessage(), e);
                }
                }

                public Path findEmployeeFile(String employeeID) throws
IOException {
                Path base =
storageHR.getDefault_filepath_employeeStorage();
                String employeeFileName = employeeID + ".txt";

                try (DirectoryStream<Path> directoryStream =
Files.newDirectoryStream(base)) {
                for (Path statusDir : directoryStream) {
                if (Files.isDirectory(statusDir)) { // Ensure it's a directory
                Path employeeFile =
statusDir.resolve(employeeFileName);
                if (Files.exists(employeeFile)) {
                        return employeeFile;
                }
                }
                }
                }
                return null; // Return null if the candidate file is not found
                }


//              public void retrieveEmployeeByEmployeeID(String
employeeID) {
//              try {
//              //TODO: implement file storage into the loadFileAndPrint
path
//              storageHR.loadFileAndPrint(storageHR.getFilepath() +
"\\" + employeeID + ".txt");
//              } catch (Exception e) {
//              throw new RuntimeException("Error in
retrieveEmployeeByEmployeeID: \n" + e);
//              }
//              }
}

package src.HR.src;
import src.TextEditor.*;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.Scanner;
```

```
public class fileStorageHR {

	private final Path default_filepath_employeeStorage =
Paths.get("src", "HR", "repository", "employeeStorage");
	private final Path default_filepath_candidateStorage =
Paths.get("src", "HR", "repository", "candidateStorage");
	String filepath = "";
	public PoorTextEditor poorJarser = new PoorTextEditor();
//haha haha

	public fileStorageHR() {}

	public String getFilepath() {
	return filepath;
	}

	public Path getDefault_filepath_employeeStorage() {
	return default_filepath_employeeStorage;
	}

	public Path getDefault_filepath_candidateStorage() {
return default_filepath_candidateStorage; }

	public void setFilepath(String filepath) {
	this.filepath = filepath;
	}

	/**
	* Deletes a given file, expects to be given the filepath
however
	* @param filename name of File to be deleted, must
contain file extension
	*/
	public void deleteFile(String filename) throws Exception {
	try {
	Path filePath = Paths.get(getFilepath(), filename);
	File file = filePath.toFile();
	System.out.println("Attempting to delete: " +
file.getAbsolutePath());

	if (file.exists()) {
	if (file.delete()) {
	System.out.println(filename + " deleted successfully");
	} else {
	throw new Exception("Failed to delete " + filename);
	}
	} else {
	throw new FileNotFoundException("File not found: " +
file.getAbsolutePath());
	}
	} catch (Exception e) {
	throw new Exception(e.getMessage());
	}
	}



	/**
	* @param folderName the name of the Department folder
to be accessed
	* @return returns a String representation of the folderPath
to the Department Folder
	* @throws IOException error checking
	*/
	public Path getEmployeeStoragePath(String folderName)
throws IOException {
	Path folderPath = Paths.get("src", "HR", "repository",
"employeeStorage", folderName.toUpperCase());
	if (!Files.exists(folderPath)) {
	throw new FileNotFoundException("Folder not found: " +
folderName.toUpperCase());
	}

	return folderPath.toAbsolutePath();
	}

	/**
	*
	* @param folderName the specific Status folder to set the
path to
	* @return returns a Path object that contains the absolute
path of the defined folder
	* @throws IOException error checking
	*/
	public Path getCandidateStoragePath(String folderName)
throws IOException {
	Path folderPath = Paths.get("src", "HR", "repository",
"candidateStorage", folderName.toUpperCase());
	if (!Files.exists(folderPath)) {
	throw new FileNotFoundException("Folder not found: " +
folderName.toUpperCase());
	}

	return folderPath.toAbsolutePath();
	}

	/**
	*
	* @param filePath the path to the file to be printed out to
console
	* @throws Exception error checking
	*/
	public void loadFileAndPrint(String filePath) throws
Exception {
	try {
	System.out.println("Attempting to load: " + filePath);
	File file = new File(filePath);
	System.out.println("Loading file: " +
file.getAbsolutePath());

	// Check if the file exists
	if (!file.exists()) {
	throw new FileNotFoundException("File not found: " +
file.getAbsolutePath());
	}

	try (Scanner scanner = new Scanner(file)) {
	while(scanner.hasNextLine()) {
	String line = scanner.nextLine();
	System.out.println(line);
	}
	}
	} catch (Exception e) {
	throw new Exception("Error encountered in
loadFileAndPrint with file: "
	+ filePath + "\n" + e.getMessage(), e);
	}
	}

}


package src.HR;

import src.App;
import src.HR.src.*;
import java.io.IOException;
import java.nio.file.Path;
import java.util.Scanner;
```

```
public class HRDepartment {

	public void start() throws Exception {
	fileStorageHR storage = new fileStorageHR();
	System.out.println("Current file path is: " +
storage.getFilepath());
	employeeRecordManager empHandler = new
employeeRecordManager(storage);
	candidateRecordManager canHandler = new
candidateRecordManager(storage);

	boolean loop = true;

	//start of user interaction
	while(loop) {
	/*
	TODO:
	- Add in new logic capabilities
	- Replace with container system so i dont have to deal
with the switch cases
	- Replace old logic with new updated methods
	- decide whether altering the file path can be done by
user
	*/
	System.out.println("Welcome to the HR Department!");
	System.out.println("Please choose from these options:");
	System.out.println("1. Add Employee");
	System.out.println("2. Remove Employee");
	System.out.println("3. Retrieve Employee");
	System.out.println("4. Update Employee");
	System.out.println("5. Display All Employees");
	System.out.println("6. Display All Employees By
Department");

	System.out.println("7. Add Candidate");
	System.out.println("8. Remove Candidate");
	System.out.println("9. Retrieve Candidate");
	System.out.println("10. Update Candidate");
	System.out.println("11. Display All Candidates");
	System.out.println("12. Display All Candidates By
Status");

	System.out.println("0. Exit");
	Scanner input = new Scanner(System.in);
	int choice = input.nextInt();
	switch (choice) {
	case 1: //add employee
	System.out.println("Enter employee name: ");
	String name = input.next();
	System.out.println("Enter employeeID: ");
	String employeeId = input.next();
	System.out.println("Enter employee department: ");
	for(int i = 0; i < Department.values().length; i++) {

System.out.println(Department.values()[i].name());
	}
	Department department =
Department.valueOf(input.next().toUpperCase());
	System.out.println("Enter employee position: ");
	String position = input.next();
	System.out.println("Enter employee employment status
(i.e. onboarding): ");
	String employmentStatus = input.next();
	System.out.println("Enter employee salary: ");
	int salary = input.nextInt();
	Employee employeeHolder = new Employee(employeeId,
name, department, position, employmentStatus, salary);
	empHandler.addEmployee(employeeHolder);
	System.out.println("Employee added successfully!
Returning to menu...\n\n\n\n");
	break;

	case 2: //remove employee
	System.out.println("Enter employee ID: ");
	String markedEmployee = input.next();
	empHandler.removeEmployee(markedEmployee);
	System.out.println("Employee removed successfully!
Returning to menu...\n\n\n\n");
	break;

	case 3: //retrieve employee
	System.out.println("Enter Employee ID: ");
	String markedEmpID = input.next();
	Path currentEmpFile =
empHandler.findEmployeeFile(markedEmpID);
	storage.loadFileAndPrint(currentEmpFile.toString());
	System.out.println("Employee retrieved successfully!
Returning to menu...\n\n\n\n");
	break;

	case 4: //update employee //TODO working i think
	System.out.println("Enter employee ID: ");
	String updateEmployeeID = input.next();
	empHandler.updateEmployee(updateEmployeeID);
	System.out.println("Employee updated successfully!
Returning to menu...\n\n\n\n");
	break;

	case 5: //display all employees
	System.out.println("List of All Employees: ");
	for (Department department1 : Department.values()) {

empHandler.displayEmployeesByDepartment(department1);
	}
	System.out.println("END OF LIST, returning to
menu...\n\n\n\n");
	break;

	case 6: //display employees in a department
	System.out.print("Please enter Department folder to list:
");
	Department departmentFolder =
Department.valueOf(input.next().toUpperCase());
	System.out.println();

empHandler.displayEmployeesByDepartment(departmentFolder);
	System.out.println("END OF LIST, returning to
menu...\n\n\n\n");
	break;

	case 7: //add candidate
	System.out.println("Enter candidate name: ");
	String candidateName = input.next();
	System.out.println("Enter candidateId");
	String candidateId = input.next();
	System.out.println("Enter position candidate applied for:
");
	String positionApplied = input.next();
	System.out.print("Enter candidate status:
\nApplied\nApproved\nHiring\nPending\nRejected\nEnter Here: ");
	candidateStatus candidateStatus =
src.HR.src.candidateStatus.valueOf(input.next().toUpperCase());
	Candidate newCandidate = new Candidate(candidateId,
candidateName, positionApplied, candidateStatus);
	canHandler.addCandidate(newCandidate);
	System.out.println("Candidate added successfully!
Returning to menu...\n\n\n\n");
	break;
```

```
                case 8: //remove candidate
                System.out.println("Enter candidate ID to be removed: ");
                String candidateID = input.next();
                canHandler.removeCandidate(candidateID);
                System.out.println("Candidate removed successfully!
Returning to menu...\n\n\n\n");
                break;

                case 9: //retrieve candidate by id
                System.out.println("Enter Candidate ID: ");
                String candidateID3 = input.next();
                Path currentCandidateFile =
canHandler.findCandidateFile(candidateID3);
                storage.loadFileAndPrint(currentCandidateFile.toString());
                break;

                case 10: //update candidate //TODO CHECK IT
                System.out.println("Enter Candidate ID: ");
                String candidateID2 = input.next();
                canHandler.updateCandidate(candidateID2);
                System.out.println("\nReturning to menu...\n\n\n\n");
                break;

                case 11: //list all candidates
                System.out.println("List of All Candidates: ");
                for (src.HR.src.candidateStatus candidate :
src.HR.src.candidateStatus.values()) {

canHandler.displayCandidatesByStatus(candidate + "\n");
                }
                System.out.println("END OF LIST, returning to
menu...\n\n\n\n");
                break;

                case 12: //list candidates by status
                System.out.print("Please enter Status folder to list: ");
                String statusFolder = input.next().toUpperCase();
                System.out.println();
                canHandler.displayCandidatesByStatus(statusFolder);
                System.out.println("END OF LIST, returning to
menu...\n\n\n\n");
                break;

                case 0:
                loop = false;
                input.close();
                System.out.println("EXITING...");
                App.prompt(); //<---- Kicks you back to the main
homepage
                break;
                }
                }
                }

                public Employee getEmployee(String employeeID) throws
IOException {
                //TODO
                return null;
                }

}
```

Manufacturing [Doyle Chism]

```
package src.Manufacturing.src;

import src.Design.src.CustomDesign;
import src.Manufacturing.src.interfaces.ProductInterface;

public class CustomProduct implements ProductInterface {

    CustomDesign design;
    String description;
    String quantity;
    String category;

    public CustomProduct(CustomDesign design) {
        this.design = design;
    }

    @Override
    public void setName(String name) {
        this.design = new CustomDesign(design.getDesignName());
    }

    @Override
    public String getName() {
        return this.design.getDesignName();
    }


    @Override
    public void setDescription(String description) {
        this.description = description;

    }

    @Override
    public String getDescription() {
        return description;
    }

    @Override
    public void setQuantity(String quantity) {
        this.quantity = quantity;

    }

    @Override
    public String getQuantity() {
        return quantity;
    }

    @Override
    public void setCategory(String category) {

        this.category = category;
    }

    @Override
    public String getCategory() {
        return category;
    }
}

package src.Manufacturing.src;
```

```
import src.Inventory.src.Product;
import
src.Manufacturing.src.interfaces.HeadOfManufacturingInterface;
import src.Manufacturing.src.interfaces.MachineOperations;

import java.util.List;
import java.util.Map;

/*
@author Doyle Chism
 */


public class HeadOfManufacturing implements
HeadOfManufacturingInterface {


    private HeadOfManufacturing rawMaterials;

    @Override
    public void viewRawMaterials(Map<String, Integer> rawMaterials) {
        if (rawMaterials == null || rawMaterials.isEmpty()) {
            System.out.println("No raw materials collected, collect raw
materials first");
            return;
        }
        int index = 1;
        System.out.println("Select the materials needed");
        for (Map.Entry<String, Integer> entry : rawMaterials.entrySet()) {
            System.out.println(index++ + ". " + entry.getKey() + " collected
with Quantity: " + entry.getValue());
        }
    }

    @Override
    public void selectRawMaterial(int materialNumber, String
rawMaterials) {

        if (materialNumber >= 0 && materialNumber <
rawMaterials.length()) {
            System.out.println("Selected raw material: " + rawMaterials);
        }

    }

    @Override
    public void receiveFinalDesign(Map<String, String> finalDesign) {


        System.out.println("Receiving Final Design Specifications:");
        for (Map.Entry<String, String> entry : finalDesign.entrySet()) {
            System.out.println(entry.getKey() + ": " + entry.getValue());
        }
        System.out.println("Final Design has been received
successfully.");


    }

    @Override
    public void receiveCustomDesign(Map<String, String>
customDesign) {

        System.out.println("Receiving Custom Design Specifications:");
        for (Map.Entry<String, String> entry : customDesign.entrySet()) {
            System.out.println(entry.getKey() + ": " + entry.getValue());
        }
        System.out.println("Custom Design has been received
successfully.");
    }

    @Override
    public void verifyProduct(Product product) {


        System.out.println("Verifying Product: " + product.getId());
        if (product.isQualityApproved()) {
            System.out.println("Product " + product.getId() + " passed
quality verification.");
        } else {
            System.out.println("Product " + product.getId() + " failed
quality verification. It needs to be remade.");
        }


    }

    @Override
    public void verifyCustomProduct(CustomProduct customProduct) {


        System.out.println("Verifying Custom Product: " +
customProduct.getName());
        if (customProduct.isQualityApproved() &&
customProduct.getCategory()) {
            System.out.println("Custom Product " +
customProduct.getName() + " passed verification.");
        } else {
            System.out.println("Custom Product " +
customProduct.getName() + " failed verification. Please review
specifications.");
        }


    }

    @Override
    public void sendToModelling(CustomProduct product) {

        if (!product.isQualityApproved()) {
            System.out.println("Cannot send " + product.getName() + " to
Modelling. Quality verification required.");
            return;
        }
        System.out.println("Sending " + product.getName() + " to the
Modelling Department.");
    }

    @Override
    public void sendToInventory(Product product, String quantity) {


        if (!product.isQualityApproved()) {
            System.out.println("Cannot store " + product.getId() + " in
Inventory. Quality verification required.");
            return;
        }
        System.out.println("Storing " + product.getId() + " in Inventory
with quantity: " + quantity);


    }

package src.Manufacturing.src;

import src.Manufacturing.src.interfaces.MachineOperations;
```

```
public class Machines implements MachineOperations {


    private boolean running;


    @Override
    public void startMachine() {
        this.running = true;
        System.out.println("Starting Machine");
    }

    @Override
    public boolean isRunning() {
        return running;
    }
}

package src.Manufacturing.src;

import src.App;

import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class ManufacturingController {

    private ManufacturingManager manager;
    private HeadOfManufacturing headOfManufacturing;
    private Machines machine;
    private boolean isReady = false;
    private boolean productCreated = false;
    private final ManufacturingFileManager fileManager = new
ManufacturingFileManager();


    public ManufacturingController() {
        this.headOfManufacturing = new HeadOfManufacturing();
        this.machine = new Machines();
        this.manager = new ManufacturingManager();



    }

    public void run() throws Exception {

        Scanner sc = new Scanner(System.in);
        boolean exit = false;
        System.out.println("Welcome to Manufacturing System");

        while (!exit) {
            System.out.println("Please enter your choice");
            System.out.println("1. Collect Raw Materials");
            System.out.println("2. Verify Raw Materials");
            System.out.println("3. Create Product");
            System.out.println("4. Send Products to Repository");
            System.out.println("5. Exit");

            int choice = sc.nextInt();
            sc.nextLine();
            Map<String, Integer> collectedMaterials = new HashMap<>();

            switch (choice) {
                case 1:
                    System.out.println("Collect the Raw Materials from
Storage");


                    while (true) {
                        System.out.println("Enter 'done' to exit prompt at any
time");
                        System.out.println("Select Type of Raw Material: ");
                        String rawMaterial = sc.nextLine();
                        if (rawMaterial.equals("done")) {
                            break;
                        }

                        System.out.println("Select Quantity of Raw Material: "
+ rawMaterial);
                        int quantity = sc.nextInt();
                        sc.nextLine();
                        collectedMaterials.put(rawMaterial, quantity);
                        System.out.println("You have " + quantity + " items
collected of " + rawMaterial);
                    }
                    manager.collectRawMaterials(collectedMaterials);
                    System.out.println("Raw Materials Collected:");
                    collectedMaterials.forEach((rawMaterial, quantity) -> {
                        System.out.println(quantity + " items of " +
rawMaterial);
                    });
                    break;
                case 2:
                    System.out.println("Verify the Raw Materials from
Storage");

                    if (manager.getCollectedMaterials() == null ||
manager.getCollectedMaterials().isEmpty()) {
//                        System.out.println(manager.getCollectedMaterials());
//displays all materials collected from 1
//
headOfManufacturing.viewRawMaterials(manager.getCollectedMateri
als());
                        System.out.println("You have no items collected yet.");
                        break;
                    }
//                  boolean selectedMaterial = false;

headOfManufacturing.viewRawMaterials(manager.getCollectedMateri
als());
                    for (Map.Entry<String, Integer> entry :
manager.getCollectedMaterials().entrySet()) {
                        String rawMaterial = entry.getKey();
                        int quantity = entry.getValue();

                        System.out.println("Do you want to verify " + quantity +
" items of " + rawMaterial + "? (Y/N)");
                        String verifyRawMaterial = sc.nextLine();

                        if (verifyRawMaterial.equals("Y")) {
                            headOfManufacturing.selectRawMaterial(quantity,
rawMaterial);
                            System.out.println("You have " + quantity + " items
of " + rawMaterial + " verified.");
                            collectedMaterials.put(rawMaterial, quantity);
                            isReady = true;

                        } else if (verifyRawMaterial.equals("N")) {
                            System.out.println("Material " + rawMaterial + " is
not verified");
                            isReady = false;
                        } else {
                            System.out.println("Invalid input, select Y or N");
                        }
                    }
                    break;
                case 3:

                    if (isReady) {
```

```
                machine.startMachine();
            }
            if (!machine.isRunning()) {
                System.out.println("Make sure to properly verify all the
raw materials before beginning to create a product");
                break;
            }
            System.out.println("Machine is running");
            System.out.println("...................................................");
            System.out.println("Manager is creating Product...");
            System.out.println("...................................................");
            productCreated =
manager.createProduct(collectedMaterials);
            if (productCreated) {
                System.out.println("Product created. Deliver to head of
manufacturing...");
                String productName = manager.getProductName();
                Product product = new Product(productName);
                manager.deliverProduct(product);
            } else {
                System.out.println("Product creation failed, Try
again");
            }
            break;

        case 4:
            fileManager.sendDataToRepo();

        case 5:
            System.out.println("Exit Program");
            exit = true;
            App.prompt();
//              break;

        default:
            System.out.println("Please enter a valid choice");
      }
    }

    sc.close();
  }
}
```

```
package src.Manufacturing.src;

import src.Design.src.CustomDesign;
import src.Design.src.FinalDesign;
import src.TextEditor.PoorTextEditor;

import java.io.File;
import java.util.HashMap;
import java.util.Map;

public class ManufacturingFileManager {

  private final PoorTextEditor editor = new PoorTextEditor();
  private final String repo = "src/Manufacturing/repository";

  public void sendDataToRepo() {
    File f = new File(repo + "final_product.txt");
    if (f.exists()) {
      editor.processTextFile(repo + "final_product.txt");
      finalDesign = editor.getRepository();
    }
    f = new File(repo + "custom_product.txt");
    if (f.exists()) {
      editor.processTextFile(repo + "custom_product.txt");
      customDesign = editor.getRepository();
    }
    f = new File(repo + "rawMaterials.txt");
    if (f.exists()) {
      editor.processTextFile(repo + "rawMaterials.txt");
      rawMaterials = editor.getRepository();
    }

  }

  public void addFinalDesign(FinalDesign design) {
    finalDesign.put("Final Design " + design.getDesignName(),
design.mapObjects());

    editor.setRepository(finalDesign);
    editor.writeToTextFile(repo + "final_product.txt");
  }

  public void addCustomDesign(CustomDesign customDesign) {
    customDesign.put("Custom Design " +
customDesign.getDesignName(), customDesign.mapObjects());

    editor.setRepository(customDesign);
    editor.writeToTextFile(repo + "custom_product.txt");
  }

  public void addRawMaterials(RawMaterials materials) {
    rawMaterials.put("Raw Materials " +
materials.getRawMaterials(), materials.mapObjects());

    editor.setRepository(materials);
    editor.writeToTextFile(repo + "rawMaterials.txt");
  }

  public Map<String, Object> getRawMaterials() {
    Map<String, Object> rawMaterials = new HashMap<>();
    for (Map.Entry<String, Object> entry : rawMaterials.entrySet()) {
      rawMaterials.put(entry.getKey(), entry.getValue());
    }
    return rawMaterials;
  }

  public Map<String, Object> getFinalDesign() {
    Map<String, Object> finalDesign = new HashMap<>();
    for (Map.Entry<String, Object> entry : finalDesign.entrySet()) {
      finalDesign.put(entry.getKey(), entry.getValue());
    }
    return finalDesign;
  }

  public Map<String, Object> getCustomDesign() {
    Map<String, Object> customDesign = new HashMap<>();
    for (Map.Entry<String, Object> entry : customDesign.entrySet())
{
      customDesign.put(entry.getKey(), entry.getValue());
    }
    return customDesign;
  }

}
```

```
package src.Manufacturing.src;

import src.Manufacturing.src.interfaces.ManagerInterface;

import java.util.HashMap;
import java.util.Map;

/*
@author
 */
public class ManufacturingManager implements ManagerInterface {
```

```
    private Map<String, Integer> collectedMaterials = new
HashMap<>();
    private Product product;


    @Override
    public boolean createProduct(Map<String, Integer>
verifiedMaterials) {

        System.out.println("Creating product with the following materials:
");
        verifiedMaterials.forEach((material, value) ->
System.out.println(value + "items of " + material));
        this.product = new Product(verifiedMaterials.toString());
        return true;
    }

    @Override
    public void deliverProduct(Product product) {

        if (product == null) {
            System.out.println("Product has not been created");
            return;
        }
        System.out.println("Delivered to the head of manufacturing
complete!");



    }

    @Override
    public void collectRawMaterials(Map<String, Integer> materials) {
        collectedMaterials.putAll(materials);
        System.out.println("Collected Raw Materials: " +
collectedMaterials);
    }

    public Map<String, Integer> getCollectedMaterials() {
        return collectedMaterials;
    }

    public String getProductName() {
        return product.getName();
    }

}

package src.Manufacturing.src;

import src.Design.src.CustomDesign;
import src.Design.src.FinalDesign;
import src.Manufacturing.src.interfaces.ProductInterface;

public class Product implements ProductInterface {

    FinalDesign design;
    String description;
    String quantity;
    String category;

    public Product(FinalDesign design) {
        this.design = design;
    }

    @Override
    public void setName(String name) {

        this.design = new FinalDesign(design.getDesignName());
    }

    @Override
    public String getName() {
        return design.getDesignName();
    }

    @Override
    public void setDescription(String description) {
        this.description = description;

    }

    @Override
    public String getDescription() {
        return description;
    }

    @Override
    public void setQuantity(String quantity) {
        this.quantity = quantity;
    }

    @Override
    public String getQuantity() {
        return quantity;
    }

    @Override
    public void setCategory(String category) {
        this.category = category;
    }

    @Override
    public String getCategory() {
        return category;
    }
}
```