Contract Management System

1. Introduction

The system is designed to manage contracts, clients, and contract types in a business environment. It allows users to perform CRUD operations (Create, Read, Update, Delete) on these entities, ensuring data integrity and user convenience. All operations include validations to maintain data consistency.

2. Functional Requirements

2.1 Function 1: Create Contract (2 Marks)

The system requires input for creating a contract. The information to be provided includes:

- **Contract ID (int):** A unique identifier for the contract.
- **Client ID (int):** A reference to the client's ID.
- **Contract Type ID (int):** A reference to the contract type's ID.
- **Contract Name (String):** The name/title of the contract.
- **Start Date (String):** The contract start date in format YYYY-MM-DD.
- **End Date (String):** The contract end date in format YYYY-MM-DD.
- **Total Value (float):** The value of the contract.

Validation Rules:

1. Contract ID must be unique.
2. Client ID must exist in the list of clients.
3. Contract Type ID must exist in the list of contract types.
4. Contract Name must be at least 3 characters.
5. Start Date and End Date must be valid dates in the format YYYY-MM-DD, and End Date must be after Start Date.
6. Total Value must be positive.

Implementation Considerations:

- The system must have lists for clients and contract types.
- If a validation fails, display an appropriate message and prompt for re-entry.

Example Workflow:

- Enter Contract ID: 3001
- Enter Client ID: 5 (system checks if client 5 exists)
- Enter Contract Type ID: 2 (system checks if type 2 exists)
- Enter Contract Name: "Office Renovation 2024"
- Enter Start Date: 2024-05-01

- Enter End Date: 2024-08-01
- Enter Total Value: 20000

Validation messages:

- Duplicate Contract ID → “Contract ID already exists.”
- Invalid Client ID → “Invalid Client ID.”
- Invalid Contract Type ID → “Invalid Contract Type ID.”
- Invalid contract name → “Contract name must be at least 3 characters.”
- Date errors → “Invalid date format.” or “End Date must be after Start Date.”
- Value errors → “Total value must be greater than 0.”
- On success: “Contract created successfully.”

2.2 Function 2: Display All Contracts (2 Marks)

- Display a list of all contracts, sorted alphabetically by contract name.
- If names match, sort by Contract ID ascending.

2.3 Function 3: Update Contract (2 Marks)

- User enters Contract ID.
- If not found: display “Contract not found.”
- Otherwise, user may update any field (leave blank to keep current).
- Show “Update successful” or “Update failed.”

2.4 Function 4: Delete Contract (1 Mark)

- User enters Contract ID.
- If not found: display “Contract not found.”
- Otherwise, confirm and delete, showing “Delete successful.”

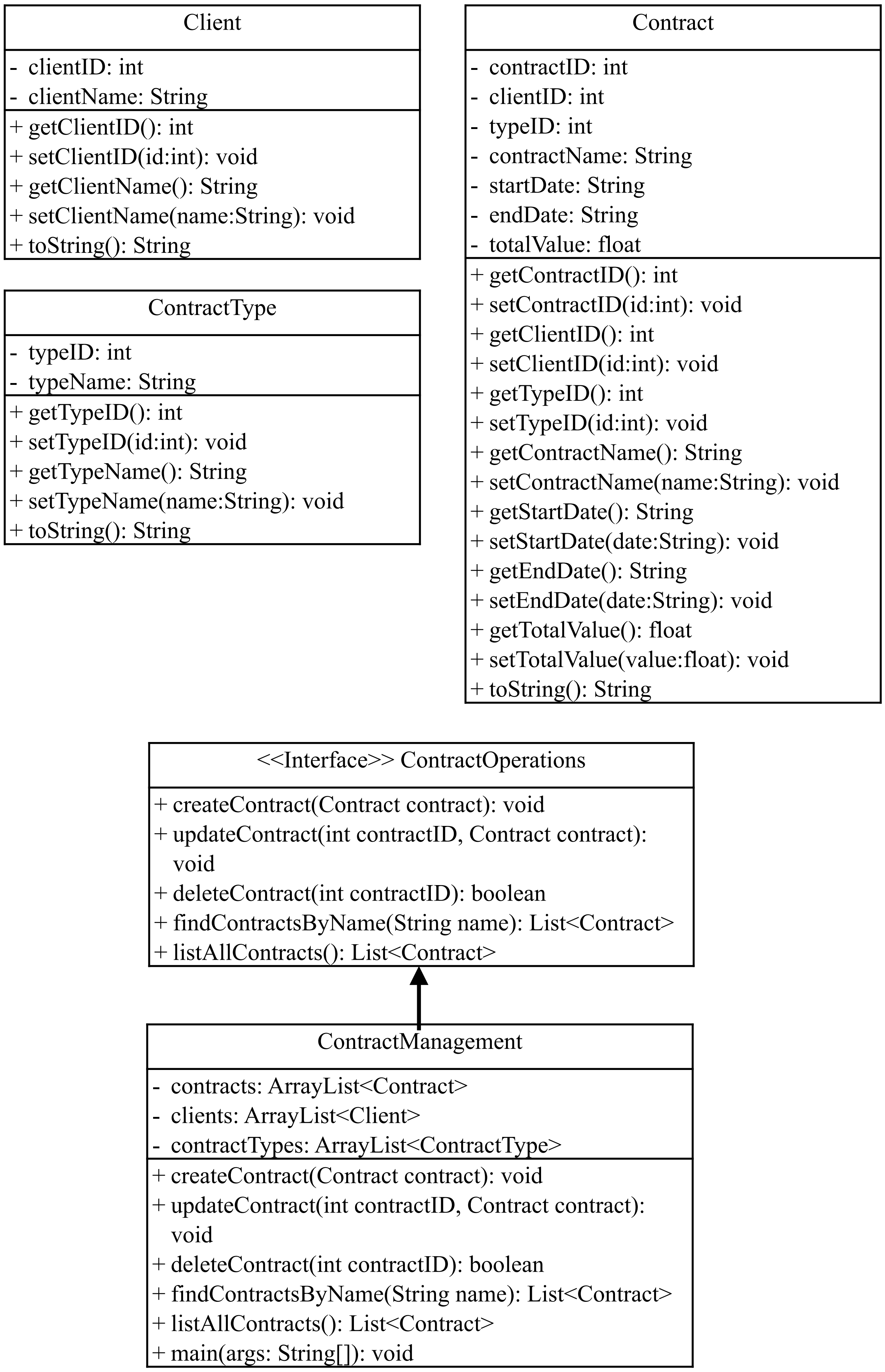
2.5 Function 5: Find Contracts by Name (2 Marks)

- Search for contracts containing a given string in their name (case-insensitive).
- Display all matches, sorted alphabetically.
- If none: “No contracts found.”

2.6 Bonus (1 Mark)

- The system can save contracts data to a file for future use (persistence).

3. UML Class Diagram


Client
- clientID: int
- clientName: String
+ getClientID(): int
+ setClientID(id:int): void
+ getClientName(): String
+ setClientName(name:String): void
+ toString(): String
ContractType
- typeID: int
- typeName: String
+ getTypeID(): int
+ setTypeID(id:int): void
+ getTypeName(): String
+ setTypeName(name:String): void
+ toString(): String
Contract
- contractID: int
- clientID: int
- typeID: int
- contractName: String
- startDate: String
- endDate: String
- totalValue: float
+ getContractID(): int
+ setContractID(id:int): void
+ getClientID(): int
+ setClientID(id:int): void
+ getTypeID(): int
+ setTypeID(id:int): void
+ getContractName(): String
+ setContractName(name:String): void
+ getStartDate(): String
+ setStartDate(date:String): void
+ getEndDate(): String
+ setEndDate(date:String): void
+ getTotalValue(): float
+ setTotalValue(value:float): void
+ toString(): String
<<Interface>> ContractOperations
+ createContract(Contract contract): void
+ updateContract(int contractID, Contract contract): void
+ deleteContract(int contractID): boolean
+ findContractsByName(String name): List<Contract>
+ listAllContracts(): List<Contract>
ContractManagement
- contracts: ArrayList<Contract>
- clients: ArrayList<Client>
- contractTypes: ArrayList<ContractType>
+ createContract(Contract contract): void
+ updateContract(int contractID, Contract contract): void
+ deleteContract(int contractID): boolean
+ findContractsByName(String name): List<Contract>
+ listAllContracts(): List<Contract>
+ main(args: String[]): void

4. Initial Data for List of Clients and Contract Types

4.1 List of Clients

```
ArrayList<Client> clients = new ArrayList<>();
clients.add(new Client(1, "Vietcombank"));
clients.add(new Client(2, "FPT Software"));
clients.add(new Client(3, "Vinamilk"));
clients.add(new Client(4, "Hoa Phat Group"));
clients.add(new Client(5, "Mobile World"));
clients.add(new Client(6, "Masan Group"));
clients.add(new Client(7, "Vingroup"));
clients.add(new Client(8, "Viettel"));
clients.add(new Client(9, "PetroVietnam"));
clients.add(new Client(10, "SABECO"));
```

4.2 List of Contract Types

```
List<ContractType> contractTypes = new ArrayList<>();
contractTypes.add(new ContractType(1, "Service"));
contractTypes.add(new ContractType(2, "Supply"));
contractTypes.add(new ContractType(3, "Consulting"));
contractTypes.add(new ContractType(4, "Maintenance"));
contractTypes.add(new ContractType(5, "Outsourcing"));
contractTypes.add(new ContractType(6, "Leasing"));
contractTypes.add(new ContractType(7, "Other"));
```

5. User Interface Requirements

- The system will be a console-based application, with a menu-driven interface. The user will be presented with a menu that allows them to select from the available functions (Create, Read, Update, Delete, and Search).
- Sample Main Menu:
  1. Create Contract
  2. Display All Contracts
  3. Update Contract
  4. Delete Contract
  5. Find Contracts by Name
  6. Save to file (Optional)
  7. Exit

-- THE END --

AI Tools & Code Quality Requirements

1. Introduction

In addition, students are required to use **AI tools (Snyk AI and ChatGPT)** for code analysis, debugging, and algorithm/pseudocode development.

2. AI Tools & Code Quality Requirements

2.1 AI Code Analysis & Debugging (Snyk AI) – (5 Marks)

- Run Snyk AI (via IDE plugin) on your project to check for security vulnerabilities, code quality issues, and bugs.
- Save and submit the generated Snyk AI analysis report (as a PDF or screenshot).
- Fix at least one critical or high-severity issue found by Snyk AI, and describe the fix in a short paragraph (3–5 sentences) in your final report.
- **Sample Snyk AI Scan Evidence and Description**

1. Snyk AI Scan Evidence (Example Screenshot Text)

```
Snyk Scan Results – Event Management System Project

[High Severity]
Issue: Insecure use of java.util.Random
File: src/util/EventCodeGenerator.java, line 23
Description: Use of java.util.Random for security-sensitive operations can lead to
predictable values. Consider using java.security.SecureRandom.
Recommendation: Replace java.util.Random with java.security.SecureRandom

[Medium Severity]
Issue: Potential NullPointerException
File: src/EventManagement.java, line 88
Description: Possible dereference of null object 'foundEvent' before calling
toString().
Recommendation: Add a null check before calling methods on potentially null
objects.
```

2. Description of the Fix (for Report)

    1. During the development of the Event Management System, I ran a Snyk AI scan using the Snyk IDE plugin for Netbeans.
       Snyk identified a high-severity issue: insecure use of java.util.Random in EventCodeGenerator.java for event code generation.
       To fix this, I replaced Random rand = new Random(); with SecureRandom rand = new SecureRandom(); as recommended.
    2. After applying the fix and rescanning, Snyk reported no further high-severity issues.
    3. Additionally, Snyk found a possible NullPointerException in EventManagement.java on line 88. I resolved this by adding a null check before calling toString() on the foundEvent object.

2.2 Pseudo Code & Algorithm Analysis (ChatGPT) – (5 Marks)

- **Requirement:**
  For at least one core function, students must use ChatGPT to:
  - Generate pseudo code for the function, and include the prompt & response in the report.
  - Analyze (may ask ChatGPT directly for suggestions).
- **Expected Deliverables:**
  - The original ChatGPT prompt and response for pseudo code generation.
  - Algorithm Analysis (can be from ChatGPT).
- **Sample: Pseudo Code & Algorithm Analysis (findEventsByName)**
  - Prompt Sent to ChatGPT

```
Please write the pseudo code for a function named findEventsByName that searches for
events containing a search string in their name (case-insensitive), from a list of Event
objects. Each Event has eventID, eventName, and other fields. The function should return a
list of matching events sorted alphabetically by eventName.
```

  - ChatGPT Response (Pseudo Code)

```
Pseudo Code: findEventsByName

FUNCTION findEventsByName(eventList, searchString):
    resultList ← empty list
    LOWER_SEARCH ← lowercase(searchString)
    FOR each event IN eventList:
        IF lowercase(event.eventName) contains LOWER_SEARCH:
            add event TO resultList
    SORT resultList BY eventName (alphabetically)
    RETURN resultList
```

- Algorithm Analysis
  - **Case-insensitive Search**: The algorithm converts both the event's name and the search string to lowercase to ensure case-insensitive matching.
  - **Sorting**: Sorting ensures the output is user-friendly and meets assignment requirements.
  - **Edge Cases**:
    - If no events match, the function returns an empty list.
    - If search string is empty, all events will be returned (could be restricted by an extra check).

3. **How to Submit**

Along with your Java project (code and report), include:

- Snyk AI Scan Evidence and Description report (PDF file).
- A section with ChatGPT prompt, pseudo code and algorithm analysis (PDF file).

How to Use Snyk AI Plugin in Visual Studio Code

1. Install the Snyk Extension

- Open **Visual Studio Code -> Go to the Extensions panel (Ctrl+Shift+X).**
- Search for "**Snyk**" -> Click "**Install**" on the **Snyk Security** extension by Snyk.

2. Authenticate Snyk in VS Code

- After installation, you'll see a Snyk icon in the Activity Bar on the left -> Click the Snyk icon. You'll be prompted to authenticate:
    - Click "Sign in with Snyk."
    - Complete the login via your browser (create a free Snyk account if needed).

3. Open Your Project

- In VS Code, open the folder containing your project (Java, etc.).

4. Run a Snyk Scan

- Click the **Snyk icon** to open the Snyk panel. -> Click **"Test"** (or **"Scan Project"**) to scan your open folder.
- Wait for the scan to complete (this may take a few seconds).

5. Review the Results

- Results are displayed directly in the Snyk panel.
    - **Security vulnerabilities** in dependencies (e.g., Maven, npm, etc.).
    - **Code quality issues & bugs** (if Snyk Code is enabled).
- Each issue displays:
    - **Severity** (Low/Medium/High/Critical)
    - **File & line number**
    - **Description and recommended fix**

6. Fix Issues and Re-scan

- Click on an issue to jump to the line of code -> Apply the suggested fix (e.g., update a library, improve code).
- Re-run the scan to check that the issue is resolved.

7. Export or Screenshot the Results (for Reports)

- You can **take a screenshot** of the Snyk panel showing found issues.
- Optionally, copy and paste the results into your assignment report.

8. References

- https://docs.snyk.io/scm-ide-and-ci-cd-integrations/snyk-ide-plugins-and-extensions/visual-studio-code-extension
- https://snyk.io/

-- THE END --