Problem Statement: Employee Data Management & File Processing System (Using BufferedReader & PrintWriter)

Develop a Core Java application to manage employee data, demonstrating efficient file I/O, data transformation, and user input handling with robust validation. **You must utilize java.io.BufferedReader for reading text files and java.io.PrintWriter for writing to text files.**

Phase 1: Employee Data Ingestion & Console Display

- **Requirement:** Read employee records from a designated text file (employees.txt).
- **Input Format:** The file uses $ as the delimiter between fields. Each line represents an employee with the following fields, in order: ID, First Name, Last Name, Mobile Number, Email Address, Joining Date (YYYY-MM-DD), Active Status (true/false).
- **Tooling:** Use BufferedReader to read the input file line by line.
- **Output:** Parse the data from each line, create appropriate employee objects (or data structures), and print each employee's details in a formatted, human-readable way to the console.
- **Project Structure to be followed**

  EmployeeManager/

  ├── src/

  │ └── com/litmus7/employeemanager/

  │ ├── app/

  │ │ └── EmployeeManagerApp.java

  │ ├── controller/

  │ │ └── EmployeeController.java

  │ ├── dto/

  │ │ └── Employee.java

  │ └── util/

  │

  │ ├── TextFileUtil.java

```
│       ├── ValidationUtil.java
│
├── employees.txt  (Input file for Phase 1 - place directly in project root)
```

Phase 2: Data Transformation & CSV Export

- **Requirement:** Convert the parsed employee data from Phase 1 into a standard comma-separated values (CSV) format.
- **Tooling:** Use PrintWriter to write the converted data to a new file named employees.csv.
- **Output:** Each employee record should be written as a new line in the employees.csv file, with fields separated by commas. Ensure proper CSV escaping if any field content could contain commas.

Phase 3: Interactive Data Entry & Appending

- **Requirement:** Enable users to input new employee data interactively via the console.
- **Tooling:** Use BufferedReader (e.g., wrapped around System.in) for console input. Use PrintWriter (in append mode) for file writing.
- **Validation:** Implement robust input validation for all fields to ensure data integrity. Examples include:
    - ID: Must be a unique positive integer.
    - First Name, Last Name, Mobile Number, Email Address: Cannot be empty.
    - Mobile Number: Must be a valid numerical format (e.g., 10 digits).
    - Email Address: Must follow a basic email format (e.g., user@domain.com).
    - Joining Date: Must be a valid date in YYYY-MM-DD format and not a future date.
    - Active Status: Must be true or false (case-insensitive).
- **Output:** Upon successful validation, append the newly entered employee record as a new line to the employees.csv file. The application should continue prompting for new entries until the user explicitly indicates they are done.

Appendix: Java Naming Conventions

Consistent naming conventions are crucial for code readability, maintainability, and collaboration in Java projects. Following these widely accepted conventions (from Oracle's Java Code Conventions) will make your code immediately familiar to other Java developers.

Category	Convention	Examples	Explanation
Packages	**lowercase.separated.by.dots**	com.globalmart.catalog.model, java.util.concurrent	All lowercase letters. Words are separated by dots. Should be unique (often based on reversed domain name).
Classes & Interfaces	**PascalCase** (UpperCamelCase)	Employee, OrderProcessor, CatalogService, Serializable	Starts with an uppercase letter, and the first letter of each subsequent word is capitalized. Should be nouns or noun phrases.
Methods	**camelCase** (lowerCamelCase)	calculateTotal(), getActiveCatalog(), processInput(), toString()	Starts with a lowercase letter, and the first letter of each subsequent word is capitalized. Should be verbs or verb phrases.

Category	Convention	Examples	Explanation
Variables	**camelCase** (lowerCamelCase)	employeeId, firstName, mobileNumber, tempResult	Same as methods. Should be short, meaningful, and indicate the variable's purpose. Avoid single-letter variable names unless in a very short loop (e.g., i, j, k).
Constants	**SCREAMING_SNAKE_CASE**	MAX_CAPACITY, DEFAULT_VALUE, PI, DEBUG_MODE	All uppercase letters. Words are separated by underscores (_). Used for public static final fields.
Enums	**PascalCase**	Status, UserRole, Color	Same as classes. Enum constants within an enum (e.g., Status.ACTIVE) follow SCREAMING_SNAKE_CASE like constants.
Type Parameters	**SingleUppercaseLetter**	<T>, <E>, <K>, <V>, <N>	Used in generics. T for Type, E for Element, K for Key, V for Value, N for Number. If

Category	Convention	Examples	Explanation
			multiple, use S, U, V. For comparable types, C. For generic return types, R.
Exceptions	**PascalCase**	IllegalArgumentException, FileNotFoundException	Same as classes, typically ending with Exception.
Local Variables	**camelCase**	line, reader, parsedData	Same as fields. Short-lived variables within a method.
Interfaces (Functional)	**PascalCase**	Runnable, Comparator, Predicate	Same as classes. Often describes a capability or an action.