**Low Level Design Document**

**for**

**Employee Management System (EMS)**

Version 1.0

**Table of Contents**

Table of Contents……………………….……………………….……………………….……

1. Introduction……………………….……………………….……………………….……….

1.1 Purpose……………………….……………………….……………………….…..

1.2 Reading Suggestions……………………….……………………….……………..

2. Detailed System Design……………………….……………………….…………………...

2.1 Class Diagram……………………….……………………….……………………

2.2 Class Description……………………….……………………….………………...

2.3 Relationship between Classes……………………….……………………….……

**1. Introduction**

**1.1 Purpose**

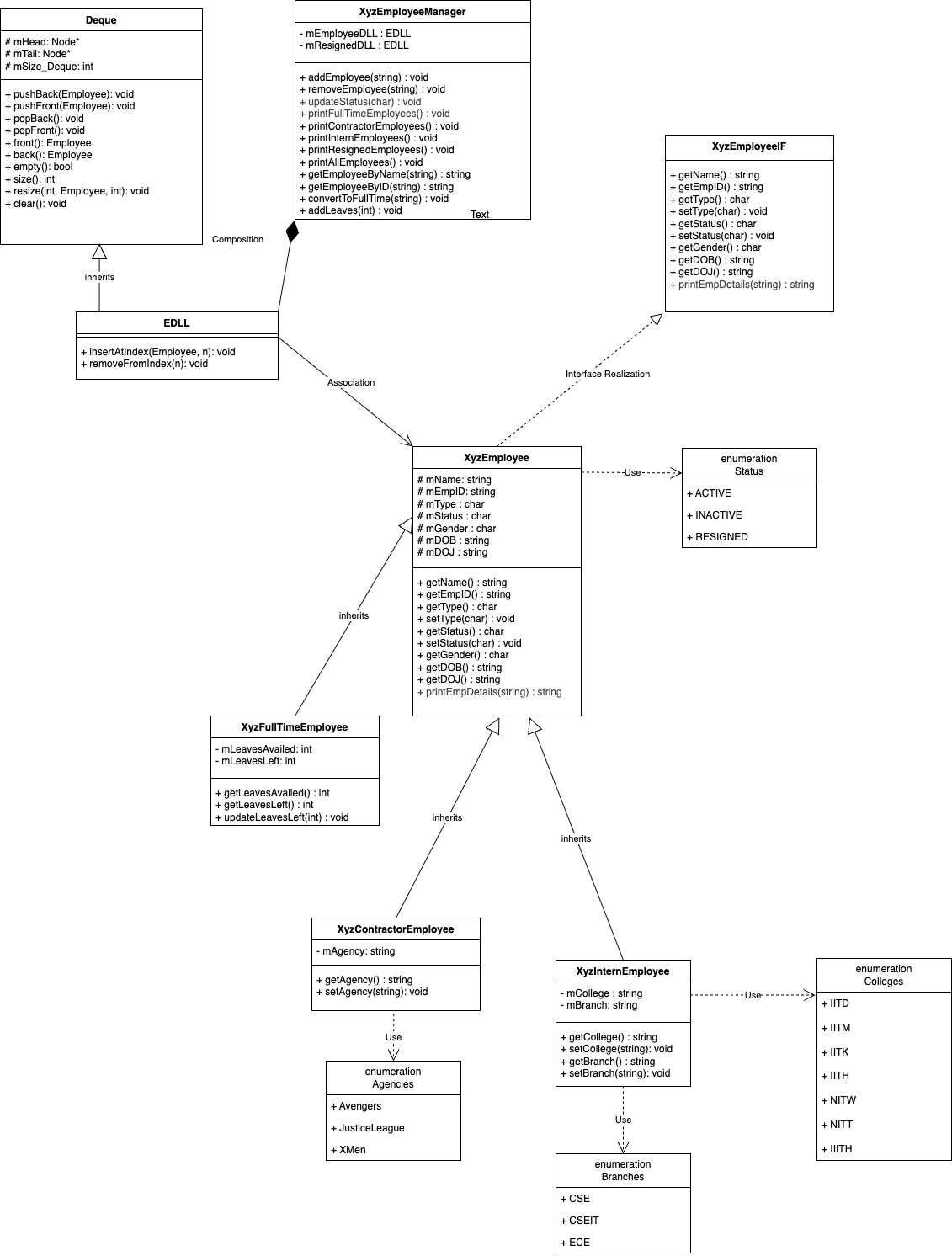
Employee Management System or EMS is created for management of employees of XYZ corporation. This EMS will provide a simplified menu-based interface for employee management operations including adding, removing or modifying employee details.

**1.2 Reading Suggestions**

Document is primarily intended for Review members for C++ Assignment 2.

**2. Detailed System Design**

**2.1 Class Diagram**

****

**2.2 Class Description**

**2.2.1 XyzEmployeeManager**

The management class of Employee Management System (EMS) that is responsible for adding, removing or modifying employee details according to user input from the main menu. XyzEmployeeManager maintains Employee Deque list of Active/Inactive and Resigned employees and performs operations over them.

|  |  |
| --- | --- |
| **Component** | **Description** |
| mEmployeeDLL | Deque for Storing Active/Inactive Employees of all types – Full Time, Contractors and Interns |
| mResignedDLL | Deque for storing resigned employees of all types |
| addEmployee(string) | Method to add Employee to mEmployeeDLL |
| removeEmployee(string) | Method to add Employee to mResignedDLL |
| updateStatus(char) | Method to update Status of an Employee to Active, Inactive or Resigned. In case of Resignation, it is responsible to pop Employee from mEmployeeDLL and push it to mResignedDLL. |
| printFullTimeEmployees() | Prints all Full time employees’ details |
| printContractorEmployees() | Prints all Contractor employees’ details |
| printInternEmployees() | Prints all Intern employees’ details |
| printResignedEmployees() | Prints all Resigned employees’ details |
| printAllEmployees() | Prints all employees’ details |
| getEmployeeByName(string) | Prints employee’s details by name |
| getEmployeeByID(string) | Prints employee’s details by employee ID |
| convertToFullTime(string) | Converts Contractor or Interns to Full Time Employee |
| addLeaves(int) | Update leaves for all full time employees |

**2.2.2 XyzEmployeeIF**

Employee Interface class which will be used to realize EmployeeBase class for creating and maintaining employee objects.

|  |  |
| --- | --- |
| **Component** | **Description** |
| getName() | Interface functions |
| getEmpID() |
| getType() |
| setType(char) |
| getStatus() |
| setStatus(char) |
| getGender() |
| getDOB() |
| getDOJ() |
| printEmpDetails(string) |

**2.2.3 XyzEmployee**

Employee Base class which has all employee details’ instances.

|  |  |
| --- | --- |
| **Component** | **Description** |
| mName | Instance to store Employee’s Name |
| mEmpID | Instance to store Employee’s ID |
| mDOB | Instance to store Employee’s Date of Birth |
| mDOJ | Instance to store Employee’s Date of Joining |
| mType | Instance to store Employee’s Type – Full Time, Contractor, Intern |
| mStatus | Instance to store Employee’s Status – Active/Inactive/Resigned |
| mGender | Instance to store Employee’s Name |
| getName() | Returns Employee’s Name |
| getEmpID() | Returns Employee’s ID |
| getType() | Returns Employee’s Type |
| setType(char) | Sets Employee’s Type |
| getStatus() | Returns Employee’s Status |
| setStatus(char) | Sets Employee’s Status |
| getGender() | Returns Employee’s Gender |
| getDOB() | Returns Employee’s Date of Birth |
| getDOJ() | Returns Employee’s Date of Joining |
| printEmpDetails(string) | Prints Employee’s details based on Employee ID |

**2.2.4 XyzFullTimeEmployee**

Full Time Employee class having XyzEmployee class instances and methods along with extended capabilities including leaves details and related methods.

|  |  |
| --- | --- |
| **Component** | **Description** |
| mLeavesAvailed | Stores leaves availed by an employee |
| mLeavesLeft | Stores leaves left with an employee |
| getLeavesAvailed() | Returns leaves availed |
| getLeavesLeft() | Returns leaves left |
| updateLeavesLeft() | Updates by adding leaves to ‘leaves left’ with an employee |

**2.2.5 XyzContractorEmployee**

Contractor Employee class having XyzEmployee class instances and methods along with extended capabilities including agency details and related methods.

|  |  |
| --- | --- |
| **Component** | **Description** |
| mAgency | Stores Agency of the Contractor |
| getAgency() | Returns Agency of the Contractor |
| setAgency(string) | Sets Agency for the Contractor |

**2.2.6 XyzInternEmployee**

Intern Employee class having XyzEmployee class instances and methods along with extended capabilities including college and branch details and related methods.

|  |  |
| --- | --- |
| **Component** | **Description** |
| mCollege | Stores college name of intern |
| mBranch | Stores branch of Intern |
| getCollege() | Returns college of Intern |
| setCollege(string) | Sets college of Intern |
| getBranch() | Returns branch of intern |
| setBranch(string) | Sets branch of intern |

**2.2.7 Deque**

Standard Deque template class implementation which will be used as a base class to implement EDLL class to store employee data nodes.

|  |  |
| --- | --- |
| **Component** | **Description** |
| mHead | Pointer to head element of Deque |
| mTail | Pointer to tail element of Deque |
| mSize\_Deque | Stores size of Deque |
| pushBack(Employee) | Pushes Employee object to back of deque |
| pushFront(Employee) | Pushes Employee object to front of deque |
| popBack() | Pops from back of deque |
| popFront() | Pops from front of deque |
| front() | Prints first element of deque |
| back() | Prints last element of deque |
| empty() | Returns true if deque is empty |
| clear() | Clears deque and sets its head and tail pointers to null |
| size() | Returns size of deque |
| resize(int, Employee, int) | Resizes deque based on direction and new size. |

**2.2.8 EDLL**

Extended Doubly Linked List Class is used to create a Deque structure with extended capabilities which will be used by XyzEmployeeManager to create Deque tp store employee data nodes.

|  |  |
| --- | --- |
| **Component** | **Description** |
| insertAtIndex(Employee, n) | Inserts a new employee at index n |
| removeFromIndex(n) | Removes the employee from deque at index n |

**2.3 Relationship between Classes**

**2.3.1 Interface Realization**

XyzEmployee class realizes XyzEmployeeIF interface to create a base class to create employee objects and hold its details.

**2.3.2 Inheritance**

XyzEmployee base class is inherited into XyzFullTimeEmployee, XyzContractorEmployee, and XyzInternEmployee.

**2.3.3 Composition**

There is a composition relationship from EDLL to XyzEmployeeManager as XyzEmployeeManager has 2 class instances of EDLL class and EDLL class can only exist inside XyzEmployeeManager class.

**2.3.4 Association**

There is an association relation between EDLL and XyzEmployee class as an EDLL class is associated with one or more XyzEmployee class objects.

**2.3.5 Dependency**

XyzContractorEmployee class depends upon or uses Agencies enum.

XyzInternEmployee class depends upon or uses Colleges and Branches enum.

XyzEmployee class depends upon or uses Status enum.