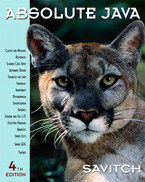
** COSC 1320**

**Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Seat # \_\_\_**

**Estimated Hours 1**

**Actual Hours**

**TA (Textual Analysis) for UML USE CASE Diagram MODEL**

**Programming Assignment 1:**

**(10 points)**

**(Due date found in the COSC 1320 BB Calendar!)**

**You must use Microsoft WORD program. Insert the TEMPLATEs for Actors and Use Cases in this Requirements Document.**

*Please have Step 1, 2, 3, and 4 in this order.*

PLEASE use the “TA for UML USE CASE Diagram MODEL Movie Company System.doc” AS TEMPLATE.

(Do not Show STEP 1,…; do not change Line Numbers; do not change Number of Pages)

Any DIAGRAM that is NOT the result of CUT and PASTE

WILL BE IGNORED. (YOU WILL GET ZERO POINTS)

**Requirements Analysis**: Construct the **UML Use Case Diagram** MODEL

**Requirements Analysis**: Construct the **UML USE CASE** Diagram MODEL

**ProgrammingAssignment1** **APPLICATION** to automate **adding**, **deleting**, and **displaying** the list of employees in a hospital.

UC3: displayEmployees

UC2: deleteEmployee

UC1: addEmployee

The set of **Class**es define these employees of a hospital: hospital employee, doctor, nurse, administrator, surgeon, receptionist, and janitor. **MUST use inheritance** in creating these **Class**es. You can use **arrays** to store objects of the same **Class**, you can assume MAX\_SIZE = 5 or **one array** to store all objects of all the **Class**es, you can assume MAX\_SIZE = 20.

(you are **not** to import java.util.\*; - failure to conform will result in losing 20 points; you are **not** to use **JCL** **ArrayList** or **LinkedList** or **Vector** Classes - failure to conform will result in losing 40 points; you are **to** use an C array - failure to conform will result in losing 30 points)

These employees are initially read and added from the **“Programming Assignment 1 Data.txt”** and are also **saved in same file** when your **ProgrammingAssignment1** **APPLICATION** terminates.

UC4: readFile

UC5: writeFile

Your **ProgrammingAssignment1** **APPLICATION** must **read** in the data file (FILE INPUT) into your **ProgrammingAssignment1** **APPLICATION** memory (**please DO NOT read from File and write to File skipping reading into the program memory**).

Your **ProgrammingAssignment1** **APPLICATION** must allow the user to **delete** a hospital employee, doctor, nurse, administrator, surgeon, receptionist, and janitor given the role and the name.

Your **ProgrammingAssignment1** **APPLICATION** must allow the user to **add** a hospital employee, doctor, nurse, administrator, surgeon, receptionist, and janitor.

Your **ProgrammingAssignment3** **APPLICATION** must allow the user to **display** **the hospital employees** in the format given below.

Your **ProgrammingAssignment3** **APPLICATION** must allow the user to **save** the hospital employees before it exits from the **ProgrammingAssignment3** **APPLICATION** from the memory to the **SAME** data file (FILE OUTPUT) **“Programming Assignment 1 Data.txt”**.

Use **Constructors** to automatically initialize the **instance variables** that MUST be declared **private** - failure to make ALL **member variables** **private** will result in losing 20 points. Appropriate accessor and mutator **methods** for each **private** **instance variable** must be created.

UC6: setName

UC7: setID

UC4: getName

UC5: getID

UC8: toString

Add (**overwrite**) the **methods** to display (must use **toString** method and super - failure to use **toString** will result in losing 10 points; - failure to use super will result in losing 10 points).

**MUST use inheritance** in creating these **Class**es - failure to use **inheritance** will result in losing 20 points).

A sample input file **“Programming Assignment 1 Data.txt”** follows:

h Vito 123

d Michael 234 Heart

n Sonny 456 6

a Luca 567 Business

r Tom 678 Talking Y

j Anthony 789 Maintenance N

d Nicos 891 Bone

The h stands for hospital employee role, Vito for **name** (only one string), and 123 is its hospital employee **number**.

The d stands for doctor, Michael for **name**, 234 for **number**, and Heart for **specialty**.

The s stands for surgeon, Vincent for **name**, 345 for **number**, Brain for **specialty**, and Y for **operating**.

The n stands for nurse, Sonny for **name**, 456 for **number**, and 6 for **numpatients**.

The a stands for administrator, Luca for **name**, 567 for **number**, and Business for **department**.

The r stands for receptionist, Tom for **name**, 678 for **number**, Talking for **department**, and Y for **answering**.

The j stands for janitor, Anthony for **name**, 789 for **number**, Maintenance for **department**, and Y for **sweeping**.

UC9: equals

Format to **display** the Hospital Employees use the following format (**must use** the **toString**() **method**):

**The Hospital has the following employees:**

**Hospital Employees: 1**

**Name: Vito Employee Number: 123**

**Doctors: 2**

**Name: Michael Employee Number: 234 Specialty: Heart**

**Name: Nicos Employee Number: 891 Specialty: Bone**

**Surgeons: 0**

**Nurses: 1**

**Name: Sonny Employee Number: 456 Number of Patients: 6**

**Administrators: 1**

**Name: Luca Employee Number: 567 Department: Business**

**Receptionists: 1**

**Name: Tom Employee Number: 678 Department: Talking Answering: Y**

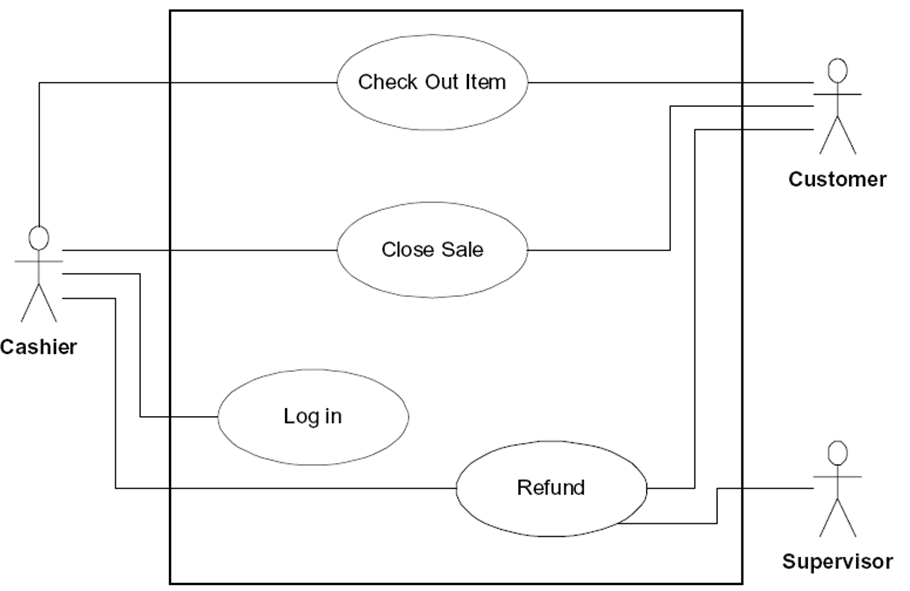
**Janitors: 1**

**Name: Anthony Employee Number: 789 Department: Maintenance Sweeping: N**

**Total number of Employees: 7**

Create a **ProgrammingAssignment1.java** that contains the **main method**.

|  |
| --- |
| Controller  UC1: addEmployee  UC2: deleteEmployee  UC3: displayEmployees  UC4: readFile  UC5: writeFile  UC6: getName  UC7: getID  UC8: setName  UC9: setID  UC10: toString  UC11: equals |



User

|  |  |
| --- | --- |
| Name: | UC1: addEmployee |
| Actor: | User |
| Description: | This use case describes the process used by User to Add another employee |
| Successful Completion: | User requests adding an employee by name, ID and role   1. HospitalEmployeeSystem checks to see if not already there and there is room 2. Employee’s name, ID and role are added and successful message is sent to the User. Different object based on role |
| Alternative: | User requests adding an employee by name, ID and role   1. HospitalEmployeeSystem checks to see if not already there and there is room 2. If employee either ALREADY present or no more room MAX\_ARTISTS Artist artistName is NOT added and UNsuccessful message is sent to the User |
| Pre-Condition: | User requests Add Employee |
| Post-Condition: | Employee name, idNumber and role Added successfully or UNsuccessfully |
| Assumptions: | None |

|  |  |
| --- | --- |
| Name: | UC2: deleteEmployee |
| Actor: | User |
| Description: | This use case describes the process used by User to delete an employee |
| Successful Completion: | User requests deleting employee by name and ID   1. HospitalEmployeeSystem checks to see if employee exists 2. Employee is deleted and successful message is sent to the User |
| Alternative: | User requests deleting employee by name and ID   1. HospitalEmployeeSystem checks to see if employee is exists 2. If employee does not exists an UNsuccessful message is sent to the User |
| Pre-Condition: | User requests delete Employee |
| Post-Condition: | Artist artistName Added successfully or UNsuccessfully |
| Assumptions: | None |

|  |  |
| --- | --- |
| Name: | UC3: displayEmployees |
| Actor: | User |
| Description: | This use case describes the process used by User to display all Employees |
| Successful Completion: | User requests to display all Employees   1. HospitalEmployeeSystem checks to see if Employee array is not empty 2. Employee list is displayed |
| Alternative: | User requests to display all Employees   1. HospitalEmployeeSystem checks to see if not already there and there is room 2. If artistName either ALREADY present or NO MORE ROOM MAX\_ARTISTS Artist artistName is NOT added and UNsuccessful message is sent to the User |
| Pre-Condition: | User requests display Employees |
| Post-Condition: | Employees are displayed or not displayed |
| Assumptions: | None |

|  |  |
| --- | --- |
| Name: | UC4: readFile |
| Actor: | User |
| Description: | This use case describes the process used by User to add Employees by reading from a file |
| Successful Completion: | User submits file to be read   1. HospitalEmployeeSystem reads file by line, checks to see if file exists 2. Employee data read by line and calls addEmployee and successful message is sent to the User |
| Alternative: | User submits file to be read   1. HospitalEmployeeSystem reads file by line, checks to see if info already exists 2. If Employee data already exists or there no no more room, new Employee is NOT added and UNsuccessful message is sent to the User |
| Pre-Condition: | User requests HospitalEmployeeSystem to read file and add Employees |
| Post-Condition: | Employee Added successfully or UNsuccessfully |
| Assumptions: | None |

|  |  |
| --- | --- |
| Name: | UC5: writeFile |
| Actor: | User |
| Description: | This use case describes the process used by User to save current list of Employees to a file |
| Successful Completion: | User requests saving Employee records to a file   1. HospitalEmployeeSystem checks to see if file exists 2. Employee data is written to a file and successful message is sent to the User |
| Alternative: | User requests saving Employee records to a file   1. HospitalEmployeeSystem creates file 2. Employee data is written to a file and successful message is sent to the User |
| Pre-Condition: | User requests to save Employee records to a file |
| Post-Condition: | Employee records are written to a file |
| Assumptions: | None |

|  |  |
| --- | --- |
| Name: | UC6: getName |
| Actor: | User |
| Description: | This use case describes the process used by User to Add another Artist |
| Successful Completion: | User requests Adding Artist by artistName   1. HospitalEmployeeSystem returns Employee’s name |
| Alternative: | None. |
| Pre-Condition: | User requests Add Artist |
| Post-Condition: | Artist artistName Added successfully or UNsuccessfully |
| Assumptions: | None |

|  |  |
| --- | --- |
| Name: | UC7: getID |
| Actor: | User |
| Description: | This use case describes the process used by User to Add another Artist |
| Successful Completion: | User requests Adding Artist by artistName   1. HospitalEmployeeSystem returns Employee’s idNumber |
| Alternative: | None |
| Pre-Condition: | User requests Add Artist |
| Post-Condition: | Artist artistName Added successfully or UNsuccessfully |
| Assumptions: | None |

|  |  |
| --- | --- |
| Name: | UC8: setName |
| Actor: | User |
| Description: | This use case describes the process used by User to Add another Artist |
| Successful Completion: | User requests Adding Artist by artistName   1. User can set the name of a given Employee |
| Alternative: | None. |
| Pre-Condition: | User requests Add Artist |
| Post-Condition: | Artist artistName Added successfully or UNsuccessfully |
| Assumptions: | None |

|  |  |
| --- | --- |
| Name: | UC9: setID |
| Actor: | User |
| Description: | This use case describes the process used by User to set the idNumber of an Employee |
| Successful Completion: | User requests setting idNumber of an Employee   1. User can set the idNumber of an Employee |
| Alternative: | None. |
| Pre-Condition: | User requests Add Artist |
| Post-Condition: | Artist artistName Added successfully or UNsuccessfully |
| Assumptions: | None |

|  |  |
| --- | --- |
| Name: | UC10: toString |
| Actor: | User |
| Description: | This use case describes the process used by User to Add another Artist |
| Successful Completion: | User requests String of object   1. Returns stringified version of object’s data |
| Alternative: | None |
| Pre-Condition: | User requests to display a string of the objects data. |
| Post-Condition: | Artist artistName Added successfully or UNsuccessfully |
| Assumptions: | None |

|  |  |
| --- | --- |
| Name: | UC11: equals |
| Actor: | User |
| Description: | This use case describes the process used by User to Add another Artist |
| Successful Completion: | User requests to check if two objects are the same.   1. Takes an object as a parameter and compares its variables with the variables of that object. |
| Alternative: | None. |
| Pre-Condition: | User requests to check if two objects are the same. |
| Post-Condition: | True or false is returned if both objects are the same or different, respectively. |
| Assumptions: | None |

