# C# OOP Regular Exam – 05 April 2025

# Access Control System



1. **Overview**

*The* ***Access Control System*** *is designed to regulate employee access to restricted zones within a company. Employees are assigned to departments, each with a defined access level. Security zones require specific access privileges, and access logs track entries based on unique security IDs. The system must enforce security rules while allowing managers to override access restrictions.*

## Setup

* Upload **only the AccessControlSystem** project in every task **except** **Unit Tests.**
* **Do not modify the interfaces or their packages.**
* Use **strong cohesion** and **loose coupling.**
* **Use inheritance and the provided interfaces wherever possible**:
  + This includes **constructors**, **method parameters,** and **return types.**
* **Do not** violate your **interface** **implementations** by adding **more public methods** in the concrete class than the interface has defined.
* Make sure you have **no public fields** anywhere.
* **Exception** and **Output messages** are in the **"Utilities"** folder.
* To solve this problem, use **Visual Studio 2022** and **Net 6.0**
* **Do not use** "\r\n" **for a new line.**

# Task 1: Structure (50 points)

**For this task’s evaluation, logic in the methods isn’t included.**

You are given some **interfaces** and must **implement** their functionality in the **correct classes**.

There are **3** types of entities: **Employee**, **Department**, and **Organization**. You should also implement two generic repositories: **EmployeeRepository** and **SecurityZoneRepository.**

## Employee

The Employee is a **base class** representing any type of employee. It defines the fundamental attributes and behaviors common to all types of employees. This class **cannot be instantiated directly**.

### Data

* **Name** - **string**
  + It must NOT be **null or whitespace.**
  + If **invalid,** throw a new **ArgumentException** with the message:

"Employee name is required."

* **Department – IDepartment**
  + Represents the department to which the employee is assigned
* **SecurityId –** **int**
  + Represents the **unique** security identifier for the employee
  + Must be a **positive number in the range [100-999]** inclusive

If **a number outside the range is passed,** throw a new **ArgumentException** with the message:  
"Security ID must be in the range [100-999]."

### Constructor

An Employee should take the following values upon initialization:

string name, int securityId

### Behavior

#### void AssignToDepartment(IDepartment department)

Assigns the employee to a department by updating the **value of the Department's property**

#### Override ToString() method:

Overrides the existing method ToString()and modifies it, so the returned **string must be on a single line**, in the following format:

"Employee: {Name}, Department: {Department}, Security ID: {SecurityId}"

### Child Classes

There are **two** concrete **types** of **Employees** in our project.

Each employee **inherits** from the Employee base class

#### GeneralEmployee

Represents **regular employees** who can contract **ONLY with the HRDepartment and FinanceDepartment**.

The Constructor of the **GeneralEmployee** should take the following values upon initialization:

string name, int securityId

#### ITSpecialist

Represents **employees working exclusively in the ITDepartment**.

The Constructor of the **ITSpecialist** should take the following values upon initialization:

stringname, intsecurityId

## Department

The Department is a **base class** representing any type of department. It defines the fundamental attributes and behaviors common to all types of departments. This class **cannot be instantiated directly**.

### Data

* SecurityLevel – int
  + Represents the security level associated with the department
  + This property will be set in the child classes
* Employees – IReadOnlyCollection<string>
  + Stores the **names of employees** assigned to this department
  + The collection should be initialized as **empty** when a department is created
* MaxEmployeesCount – int
  + Represents the maximum number of employees allowed in the department
  + This property will be set in the child classes

### Behavior

#### void ContractEmployee(string employeeName)

**Adds** the provided employee **name to** a department's collection of **employees:**

* If the **department has reached its employee limit**, throw an ArgumentException with the message:  
  "Department has reached its maximum employee capacity."
* If the **employee name already exists in the department**, throw a new ArgumentException with the message: "Employee is already added to the department."

### Constructor

A **Department** should take no values upon initialization but should **initialize   
a new instance of the Employee** **collection**

### Child Classes

There are **three** concrete **types** of **Departments** in our project.

Each department **inherits** from the Department base class and **sets its own:**

* SecurityLevel (defines access to security zones)
* MaxEmployeesCount (limits the number of employees per department)

#### ITDepartment

Represents the **IT department**, responsible for managing company networks, servers, and cybersecurity.   
Requires a **high-security level** due to access to sensitive infrastructure.

* SecurityLevel = 5
* MaxEmployeesCount = 8

The Constructor of the **ITDepartment** should take **no parameters** upon initialization.

#### HRDepartment

Represents the **Human Resources department**, responsible for employee records, contracts, and personal data. Requires a **moderate security level** to protect sensitive personnel information.

* SecurityLevel = 3
* MaxEmployeesCount = 5

The Constructor of the **HRDepartment** should take **no parameters** upon initialization.

#### FinanceDepartment

Represents the **Finance department**, responsible for handling company funds, transactions, and financial strategies. Requires a **high-security level** due to access to financial accounts and transactions.

* SecurityLevel = 4
* MaxEmployeesCount = 3

The Constructor of the **FinanceDepartment** should take **no parameters** upon initialization.

## SecurityZone

### Data

* **Name** - **string**
  + It must NOT be **null or whitespace.**
  + If **invalid,** throw a new **ArgumentException** with the message:

"Security zone name is required."

* **AccessLevelRequired – int**
  + Must be a **positive integer**
  + If **invalid,** throw a new **ArgumentException** with the message:

"Required access level cannot be a negative number."

* **AccessLog – IReadOnlyCollection<int>**
  + **Stores the security IDs of employees** who are authorized to access this security zone
  + The collection should be initialized as empty when a security zone is created

### Behavior

#### void LogAccessKey(int securityId)

* **Adds the provided security ID to the access log collection**

### Constructor

A **SecurityZone** should take the following values upon initialization:

stringname, intaccessLevelRequired

## EmployeeRepository

The **EmployeeRepository** is an **IRepository<IEmployee>. Collection** of all existing **employees in the application**.

### Data

* **Models – a collection of employees (unmodifiable)**

### Behavior

**void AddNew(IEmployee employee)**

* **Adds** an **employee** to the repository

**IEmployee GetByName(string employeeName)**

* Returns an employee with an **employeeName value, equal to the given parameter**
* If **no such employee is found** in the repository, **returns null**

**int SecurityCheck(string employeeName)**

* **Returns the security level** of the **department** to which the employee is **assigned**
* If the employee is **not assigned to a department**, **returns 0**

## SecurityZoneRepository

The **SecurityZone** is an **IRepository<ISecurityZone>. Collection** of all existing **zones in the application**.

### Data

* **Models – a collection of security zones (unmodifiable)**

### Behavior

**void AddNew(ISecurityZone securityZone)**

* **Adds** a security zone to the repository

**ISecurityZone GetByName(string securityZoneName)**

* **Returns a security zone** with a securityZoneName **value**, **equal to the given parameter**
* **If** no such zone **is found in the repository, it** returns null

**int SecurityCheck(string securityZoneName)**

* **Returns the required access level** of the **zone** with a name equal to the given parameter

# Task 2: Business Logic (150 points)

## The Controller Class

The business logic of the program should be concentrated around several **commands**. You are given interfaces, which you have to implement in the correct classes.

**NOTE: Do not use** "\r\n" **for a new line.**

The first interface is the **IController**. You must create a **Controller** class, which implements the interface and implements all of its methods. The constructor of the **Controller** does not take any arguments. The given methods should have the logic described for each in the Commands section. When you create the **Controller** class, go into the **Engine** class constructor and uncomment the "this.controller = new Controller();" line.

**Data**

You need to keep track of some collections, This is why you need a private field in your controller class:

**Example:**

* **departments – ICollection<IDepartment>**
* **securityZones – SecurityZoneRepository**
* **employees – EmployeeRepository**

**NOTES:**

* **Stick to the assigned types**. You can name the collections as you like, **Judge will not test those properties.**
* **The department's collection can be a List, an Array, or whatever you choose to create and work with**

**Commands**

Several commands control the business logic. They are stated below.

#### AddDepartment Command

##### Parameters

* **departmentTypeName – string**

##### Functionality

The method should **create and add** a **new department** to the **department collection**. You **cannot duplicate departments**, so only **one department of each type** can exist.

* Check if the given departmentTypeName matches a valid child class of Department (ITDepartment, HRDepartment, or FinanceDepartment), if not valid, skip the creation process and return the following message: "{departmentTypeName**}** is not a valid department type."
* If a department of the same type already exists in the application, skip the creation process and return the following message: "{**departmentTypeName}** is already created."
* If none of the above cases occur, create an instance of the appropriate department class based on the departmentTypeName. Add it to the departments' collection and return the following message:   
  "{**departmentTypeName}** is successfully created."

#### AddEmployeeToApplication Command

##### Parameters

* **employeeName - string**
* **employeeTypeName – string**
* **securityId - int**

##### Functionality

The method should **create and add a new employee** to the employee repository. Each employee must have a **unique security ID** (handled by the Employee class constructor) and must belong to a **valid employee type:**

* Check if the given employeeTypeName matches a valid child class of Employee (GeneralEmployee or ITSpecialist), if it is not valid, skip the creation process and return the following message: "{employeeTypeName**}** is not a valid employee type."
* If an employee with the same name already exists in the application, skip the creation process and return the following message: "{**employeeName}** is already added to the application."
* If an employee with the same security id already exists in the application, skip the creation process and return the following message: "Security ID {**securityId}** is already taken."
* If none of the above cases is reached, create an instance of the appropriate child class (GeneralEmployee or ITSpecialist) based on the employeeTypeName. Add the newly created IEmployee to the EmployeeRepository.Return the following message:   
  "{**employeeName}** is successfully added to the application."

#### AddEmployeeToDepartment Command

##### Parameters

* **employeeName – string**
* **departmentTypeName - string**

##### Functionality

The method should assign an **existing employee to a department**.

Each employee **can only be assigned to one department**, and the **department must have available capacity**.

* Check if an employee with the given employeeName exists in the application. If the employee does NOT exist, return the following message:   
  "{**employeeName}** is not added to the application."
* Check if the given departmentTypeName matches a valid child class of Department (ITDepartment, HRDepartment, or FinanceDepartment), if not valid, skip the process and return the following message: "{departmentTypeName**}** is not a valid department type."
* **Check** if the employee type is allowed in the given department.
  + ITSpecialist can only be assigned to the ITDepartment
  + GeneralEmployee can only be assigned to the HRDepartment or FinanceDepartment
  + If the **employee type is not compatible with the department type**, return the following message:  
    **"{employeeTypeName**} cannot be added to **{departmentTypeName**}."
* If a **department of the given valid department type does NOT exist** in the application, skip the process and return the following message:   
  **"{departmentTypeName**} is not available.**"**
* **Extract** the **employee** with the given name from the EmployeeRepository and the **department** **matching the given type name** from the collection of departments:
  + If the employee is **already assigned to any department,** return the following message: **"{employeeName**} is already added to department.**"**
  + If the **department does NOT have available capacity for new employees,** return the following message: **"{departmentTypeName**} is full.**"**
* If **none of the above cases** is reached:
  + **Add the employee's name to the department’s employee list**
  + **Assign the department as the employee’s Department property**
  + **Return the following message: "{employeeTypeName**} is successfully assigned to **{departmentTypeName**}.**"**

#### AddSecurityZone Command

##### Parameters

* **securityZoneName – string**
* **accessLevelRequired - int**

##### Functionality

The method **adds a new security zone** to the system.

* If a security zone with the same name already exists in the SecurityZoneRepository, skip the process and return the following message:   
  "{**securityZoneName}** is already created."
* If the above case is not reached, create a new instance of SecurityZone with the given securityZoneName and accessLevelRequired. Add the newly created security zone to the securityZones repository. Return the following message:   
  "{securityZoneName} is successfully created."

#### AuthorizeAccess Command

##### Parameters

* **securityZoneName - string**
* **employeeName - string**

##### Functionality

The method authorizes an employee to **access a security zone based on their department’s security level**.   
**If the employee meets the required access level**, their **security ID is logged in the security zone’s access log**.

* Check if a security zone with the same name exists in the SecurityZoneRepository.   
  If the security zone does NOT exist in the application, return the following message:   
  "{**securityZoneName}** is not found."
* Check if an employee with the given employeeName exists in the EmployeeRepository**. If the employee does NOT exist, return the following message :**"{employeeName}is not added to the application."
* If the employee is not assigned to a department, or their department’s security level is lower than the access level required by the zone, return:  
  "{employeeName} is denied access to {**securityZoneName}**."
* Check if the employee’s security ID is already present in the security zone’s access log. If the employee has already been authorized, return:  
  "{employeeName} is already authorized to access {**securityZoneName}**."
* If none of the above cases occur, log the employee’s security ID in the security zone’s access log and return the following message:  
  "{employeeName} is authorized to access {**securityZoneName}**."

#### SecurityReport Command

##### Functionality

The method generates a detailed report of **all security zones and the employees who have been authorized** to access them.

##### Structure

The report is structured as follows:

* All security zones are listed in the report, ordered by:
  + Descending AccessLevelRequired
  + Then alphabetically by Name
* Below each security zone, **list all employees whose security IDs are recorded in the zone’s access log**
  + Each employee is displayed using their ToString() method

"Security Report:

-**{securityZone1.Name}** (Access level required: **{accessLevelRequired}**)

--Employee: **{Employee1.Name}**, Department: **{name}**, Security ID: **{SecurityId}**

--Employee: **{Employee2.Name}**, Department: **{name}**, Security ID: **{SecurityId}**

**…**

--Employee: **{Employeen.Name}**, Department: **{name}**, Security ID: **{SecurityId}**

-**{securityZone2.Name}** (Access level required: **{accessLevelRequired}**)

**…**

-**{securityZonen.Name}** (Access level required: **{accessLevelRequired}**)"

**NOTE: Do not use** "\r\n" **for a new line.**

#### Exit Command

##### Functionality

Ends the program.

### Input / Output

You are provided with a single interface that will guide you through the correct execution process of your program. The interface is Engine, and the class implementing this interface should read the input. When the program finishes, this class should print the output.

#### Input

Below, you can see the **format** in which **each command** will be given in the input:

* **AddDepartment {departmentTypeName}**
* **AddEmployeeToApplication {employeeName} {employeeTypeName} {securityId}**
* **AddEmployeeToDepartment {employeeName} {departmentTypeName}**
* **AddSecurityZone {securityZoneName} {accessLevelRequired}**
* **AuthorizeAccess {securityZoneName} {employeeName}**
* **SecurityReport**
* **Exit**

#### Output

Print the output from each command when issued. Print the exception message if an exception is thrown during any of the commands' execution.

### Examples

|  |
| --- |
| **Input** |
| **AddEmployeeToApplication Alice GeneralEmployee 123**  **AddEmployeeToDepartment Alice HRDepartment**  **AddDepartment ITDepartment**  **AddDepartment HRDepartment**  **AddDepartment FinanceDepartment**  **AddDepartment ITDepartment**  **AddDepartment MarketingDepartment**  **AddEmployeeToApplication Alice GeneralEmployee 123**  **AddEmployeeToApplication Bob ITSpecialist 234**  **AddEmployeeToApplication Charlie Pilot 345**  **AddEmployeeToApplication Alice GeneralEmployee 456**  **AddEmployeeToApplication Dave GeneralEmployee 123**  **AddEmployeeToApplication David GeneralEmployee 105**  **AddEmployeeToApplication Emma ITSpecialist 106**  **AddEmployeeToApplication Frank GeneralEmployee 107**  **AddEmployeeToApplication Grace ITSpecialist 108**  **AddEmployeeToApplication Henry GeneralEmployee 109**  **AddEmployeeToApplication Irene ITSpecialist 110**  **AddEmployeeToDepartment Zoe HRDepartment**  **AddEmployeeToDepartment Alice MarketingDepartment**  **AddEmployeeToDepartment Emma FinanceDepartment**  **AddEmployeeToDepartment Grace HRDepartment**  **AddEmployeeToDepartment David ITDepartment**  **AddEmployeeToDepartment Alice HRDepartment**  **AddEmployeeToDepartment Bob ITDepartment**  **AddEmployeeToDepartment Frank FinanceDepartment**  **AddEmployeeToDepartment Henry HRDepartment**  **AddEmployeeToDepartment Charlie HRDepartment**  **AddEmployeeToDepartment Irene ITDepartment**  **AddEmployeeToDepartment Emma ITDepartment**  **AddEmployeeToDepartment Grace ITDepartment**  **AddEmployeeToDepartment Bob ITDepartment**  **AddEmployeeToDepartment Irene ITDepartment**  **AddEmployeeToDepartment Emma ITDepartment**  **AddEmployeeToDepartment Grace ITDepartment**  **AddEmployeeToDepartment Bob ITDepartment**  **AddSecurityZone Alpha 1**  **AddSecurityZone Bravo 2**  **AddSecurityZone Charlie 3**  **AddSecurityZone Delta 4**  **AddSecurityZone Echo 5**  **AddSecurityZone Foxtrot 6**  **AddSecurityZone Hotel 0**  **AddSecurityZone Alpha 5**  **AuthorizeAccess UnknownZone Alice**  **AuthorizeAccess Echo Zoe**  **AuthorizeAccess Delta Frank**  **AuthorizeAccess Echo Alice**  **AuthorizeAccess Echo Alice**  **AuthorizeAccess Alpha Alice**  **AuthorizeAccess Bravo Bob**  **AuthorizeAccess Charlie Emma**  **AuthorizeAccess Delta Grace**  **AuthorizeAccess Echo Irene**  **AuthorizeAccess Foxtrot Emma**  **AuthorizeAccess Foxtrot Grace**  **AuthorizeAccess Foxtrot Irene**  **AuthorizeAccess Foxtrot Bob**  **SecurityReport**  **Exit** |
| **Output** |
| **Alice is successfully added to the application.**  **HRDepartment is not available.**  **ITDepartment is successfully created.**  **HRDepartment is successfully created.**  **FinanceDepartment is successfully created.**  **ITDepartment is already created.**  **MarketingDepartment is not a valid department type.**  **Alice is already added to the application.**  **Bob is successfully added to the application.**  **Pilot is not a valid employee type.**  **Alice is already added to the application.**  **Security ID 123 is already taken.**  **David is successfully added to the application.**  **Emma is successfully added to the application.**  **Frank is successfully added to the application.**  **Grace is successfully added to the application.**  **Henry is successfully added to the application.**  **Irene is successfully added to the application.**  **Zoe is not added to the application.**  **MarketingDepartment is not a valid department type.**  **ITSpecialist cannot be added to FinanceDepartment.**  **ITSpecialist cannot be added to HRDepartment.**  **GeneralEmployee cannot be added to ITDepartment.**  **GeneralEmployee is successfully assigned to HRDepartment.**  **ITSpecialist is successfully assigned to ITDepartment.**  **GeneralEmployee is successfully assigned to FinanceDepartment.**  **GeneralEmployee is successfully assigned to HRDepartment.**  **Charlie is not added to the application.**  **ITSpecialist is successfully assigned to ITDepartment.**  **ITSpecialist is successfully assigned to ITDepartment.**  **ITSpecialist is successfully assigned to ITDepartment.**  **Bob is already added to department.**  **Irene is already added to department.**  **Emma is already added to department.**  **Grace is already added to department.**  **Bob is already added to department.**  **Alpha is successfully created.**  **Bravo is successfully created.**  **Charlie is successfully created.**  **Delta is successfully created.**  **Echo is successfully created.**  **Foxtrot is successfully created.**  **Hotel is successfully created.**  **Alpha is already created.**  **UnknownZone is not found.**  **Zoe is not added to the application.**  **Frank is authorized to access Delta.**  **Alice is denied access to Echo.**  **Alice is denied access to Echo.**  **Alice is authorized to access Alpha.**  **Bob is authorized to access Bravo.**  **Emma is authorized to access Charlie.**  **Grace is authorized to access Delta.**  **Irene is authorized to access Echo.**  **Emma is denied access to Foxtrot.**  **Grace is denied access to Foxtrot.**  **Irene is denied access to Foxtrot.**  **Bob is denied access to Foxtrot.**  **Security Report:**  **-Foxtrot (Access level required: 6)**  **-Echo (Access level required: 5)**  **--Employee: Irene, Department: ITDepartment, Security ID: 110**  **-Delta (Access level required: 4)**  **--Employee: Frank, Department: FinanceDepartment, Security ID: 107**  **--Employee: Grace, Department: ITDepartment, Security ID: 108**  **-Charlie (Access level required: 3)**  **--Employee: Emma, Department: ITDepartment, Security ID: 106**  **-Bravo (Access level required: 2)**  **--Employee: Bob, Department: ITDepartment, Security ID: 234**  **-Alpha (Access level required: 1)**  **--Employee: Alice, Department: HRDepartment, Security ID: 123**  **-Hotel (Access level required: 0)** |

# Task 3: Unit Tests (100 points)

You will receive a **ZoneControlPanel** skeleton with **ControlPanel**, **SecureZone,** and **Employee** classes inside. **ControlPanel** class will have some methods, fields, and constructors that are working properly. You are **NOT ALLOWED** to change any class. **Cover the whole class** (only **ControlPanel**) with unit tests to make sure that the class is working as intended. You are provided with a **unit test project** in the **project skeleton**.

* **Do NOT CHANGE OR REMOVE ANY namespaces or usings.**
* **Do not use** "\r\n" **for a new line.**
* Do **NOT** use **Mocking** in your unit tests!
* In Judge, you upload a **.zip** of the **unit test project** from the **skeleton.**

A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a computer

Description automatically generated

A screenshot of a computer program

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.