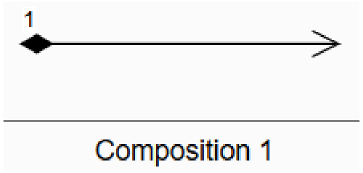
**CSCI165 Computer Science II**
**Lecture Demonstration: Object Oriented Composition**

**Objectives:**

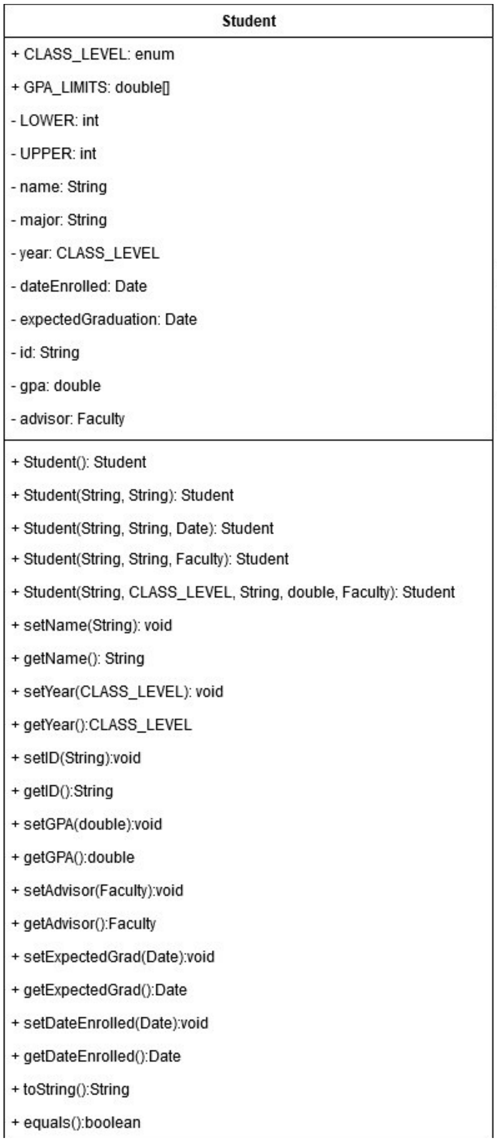
- Understand and identify the object oriented design concept: **Composition**
- Perform close analysis of privacy protection
- Identify when privacy of an internal object may be leaked
- Prove a privacy leak exists by circumventing domain validation
- Fix privacy leaks by modifying the way private objects are accepted and returned into objects

You have been provided with a small collection of custom Java classes that are designed using the composition pattern. Below you will find the UML diagram illustrating the composition notation.

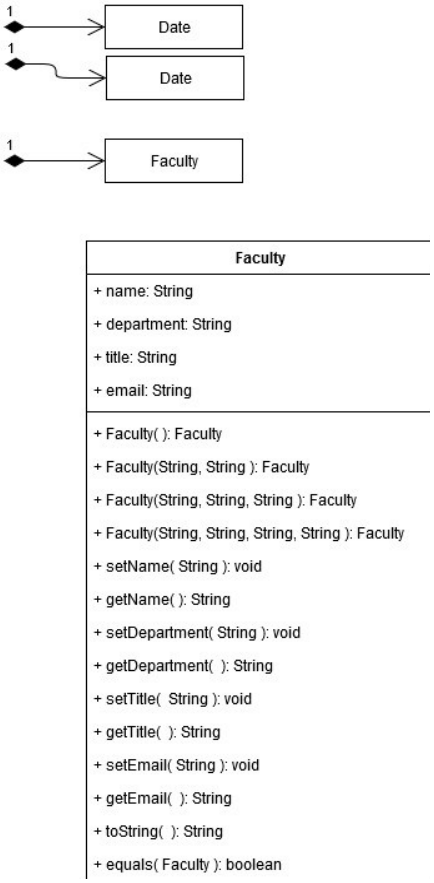
The composition exists in the Student class as depicted using the UML composition notation.



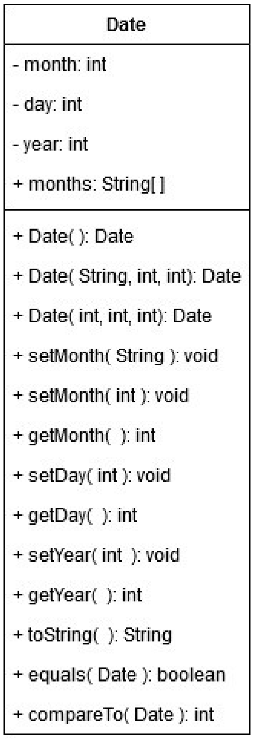
Composition 1



| Student |
|---|
| + CLASS_LEVEL: enum |
| + GPA_LIMITS: double[] |
| - LOWER: int |
| - UPPER: int |
| - name: String |
| - major: String |
| - year: CLASS_LEVEL |
| - dateEnrolled: Date |
| - expectedGraduation: Date |
| - id: String |
| - gpa: double |
| - advisor: Faculty |
| + Student(): Student |
| + Student(String, String): Student |
| + Student(String, String, Date): Student |
| + Student(String, String, Faculty): Student |
| + Student(String, CLASS_LEVEL, String, double, Faculty): Student |
| + setName(String): void |
| + getName(): String |
| + setYear(CLASS_LEVEL): void |
| + getYear():CLASS_LEVEL |
| + setID(String):void |
| + getID():String |
| + setGPA(double):void |
| + getGPA():double |
| + setAdvisor(Faculty):void |
| + getAdvisor():Faculty |
| + setExpectedGrad(Date):void |
| + getExpectedGrad():Date |
| + setDateEnrolled(Date):void |
| + getDateEnrolled():Date |
| + toString():String |
| + equals():boolean |



| Faculty |
|---|
| + name: String |
| + department: String |
| + title: String |
| + email: String |
| + Faculty( ): Faculty |
| + Faculty(String, String ): Faculty |
| + Faculty(String, String, String ): Faculty |
| + Faculty(String, String, String, String ): Faculty |
| + setName( String ): void |
| + getName( ): String |
| + setDepartment( String ): void |
| + getDepartment(  ): String |
| + setTitle(  String ): void |
| + getTitle(  ): String |
| + setEmail( String ): void |
| + getEmail(  ): String |
| + toString(  ): String |
| + equals( Faculty ): boolean |



| Date |
|---|
| - month: int |
| - day: int |
| - year: int |
| + months: String[ ] |
| + Date( ): Date |
| + Date( String, int, int): Date |
| + Date( int, int, int): Date |
| + setMonth( String ): void |
| + setMonth( int ): void |
| + getMonth(  ): int |
| + setDay( int ): void |
| + getDay(  ): int |
| + setYear( int  ): void |
| + getYear(  ): int |
| + toString(  ): String |
| + equals( Date ): boolean |
| + compareTo( Date ): int |

Understand that composition also exists when using Java objects like the String, arrays and enum shown above. I purposefully did not include those in the composition diagram for simplicity.

**Some notes on domain validation:**

1. The expected graduation date of a student should always be after the date of enrollment
2. The student's advisor should always be from the same department as the student's major.

**Exercise:** In the provided implementation of these classes there are multiple privacy leaks. Your task is to

1. Prove that they exist by demonstrating that the affected properties can be set to values that violate policy. I would like to see a Driver file that shows that these privacy leaks can be exploited. Include comments that describe why there is a privacy leak.
2. Fix the privacy leaks by changing the way internal objects are returned from classes and accepted into classes.
3. Include screen shots showing the exploits and also screen shots of the same Driver showing how the exploits were fixed. If done properly you should be able to run the same driver that proves the exploits and they will disappear.