Homework 2

Tags	CSCI 3081W
Date	@October 19, 2023

UML diagrams: Class diagram

💻 write at least one page describing and explaining how to use each of the following aspects of UML: Classes, Annotations (notes), and all Relationships.
- Make sure to describe all aspects of each completely!
- Provide examples with diagrams/pictures as necessary
- Cite your sources. (You do not need to cite the lectures as a source.)

Sources:

https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/

https://www.lucidchart.com/pages/uml-class-diagram#:~:text=In UML%2C a class represents,its attributes%2C and its operations.

http://usna86-techbits.blogspot.com/2012/11/uml-class-diagram-relationships.html

https://www.tutorialspoint.com/uml/uml_basic_notations.htm

https://agilemodeling.com/style/note.htm

https://softwareengineering.stackexchange.com/questions/405247/what-is-the-difference-between-containment-and-aggregation-relationship-in-uml

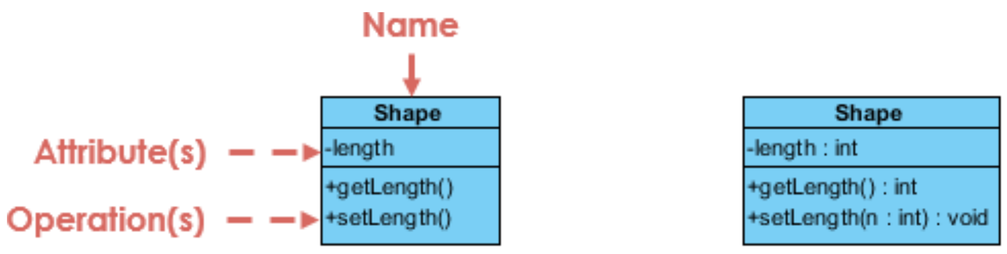
https://www.javatpoint.com/uml-association-vs-aggregation-vs-composition

Structural things:

Class -

💻 A class represents an object that contains **attributes** and **operations**.
→ Object: Can represent a real-world object or an idea.
→ Attributes: values that represent the object's data.
→ Operations: methods that represent the object's behavior.

In UML these are represented by a box with the following:

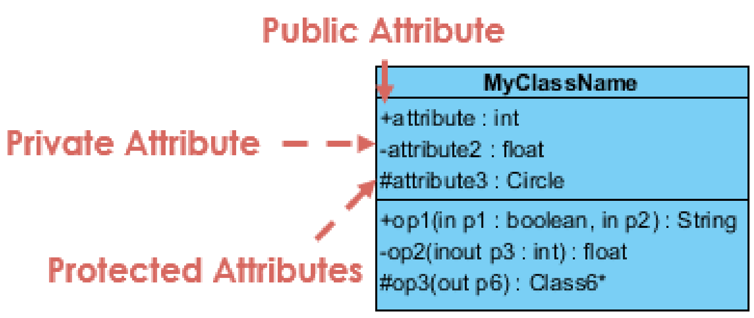


- **Name:** The top section of box
- **Attributes:** The second section in the box.
  - Each attribute is displayed as a list.
  - Each attribute has a type after the attribute name
    - e.g. `length : int`

- **Methods:** The third section in the box.
  - Operations/Methods are displayed as a list.
  - Methods can have parameter types
    - `setLength(n : int) : void`
  - Methods can have return types
    - `getLength() : int`

Class Visibility -

The Symbols `+`, `-`, `#` are placed before an attribute and operation to specify the visibility.



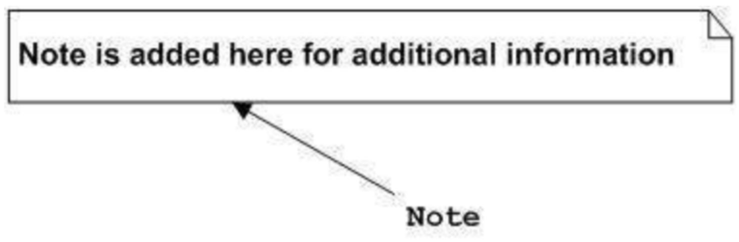
-	Private
#	Protected
+	Public

Annotational things:

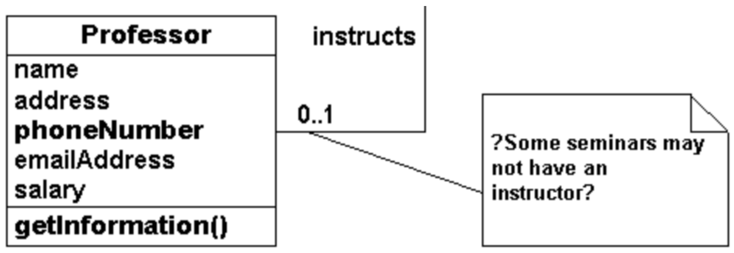
Annotation (also known as notes) -

These are used to provide further clarification or necessary information about a system diagram design.

- Represented as a box with the top-right corner folded.



This is the shape and the content of an annotation.

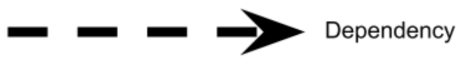


Here we see it is attached to a class diagram box to further explain some details.

Relationships:

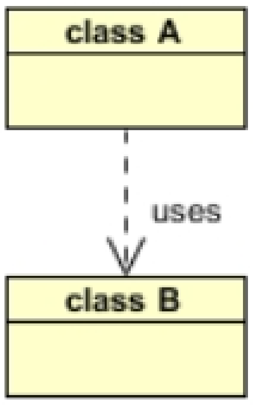
Dependency -

💻 The object of one class may use an object of another class in its methods.
- Dotted Line with Filled arrow.

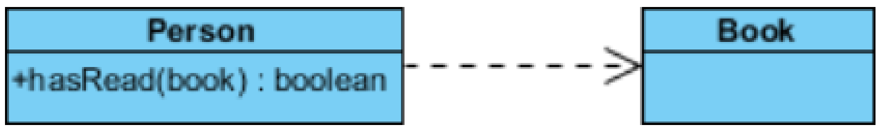


- If it is not stored in any of its attributes.
- A special type of association
- ClassA depends on ClassB

```
class A {
    public:
    void doSomething(B b) {

    }
}
```



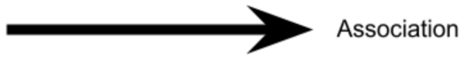
ClassA uses ClassB

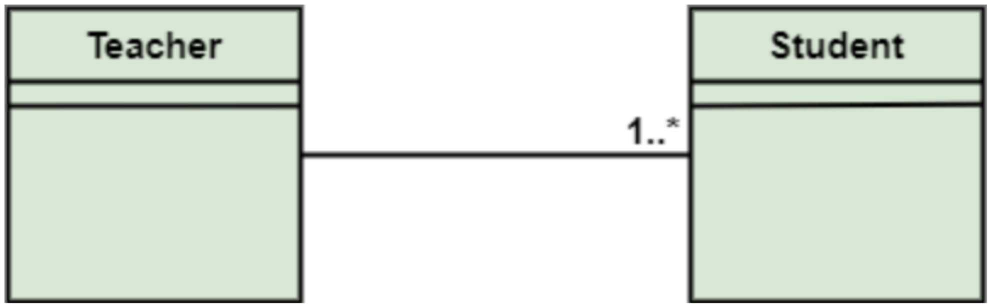


Class Person uses a Book.

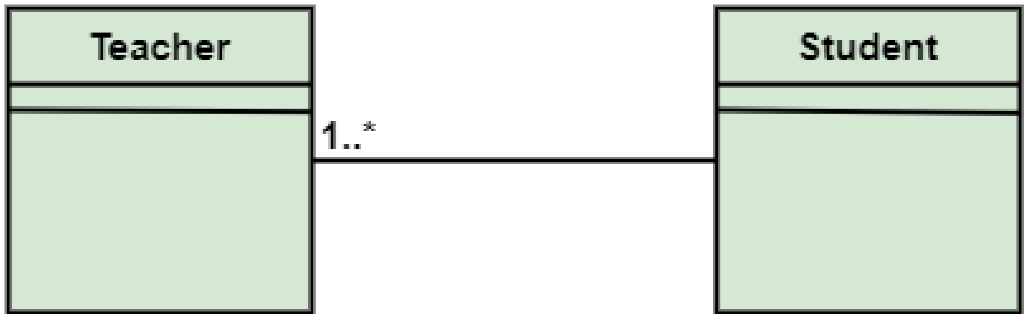
Association -

💻 Specifies a relationships between classes. A structural link between classes that are linked together.
- Solid line with a filled arrow (sometimes no arrow)



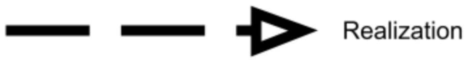


A teacher has multiple students.



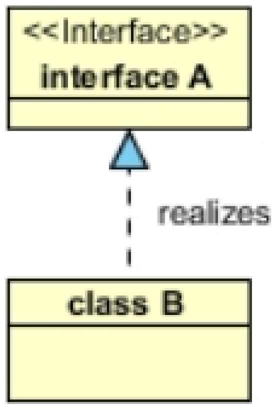
A student has 1 teacher.

Realization/Implementation -



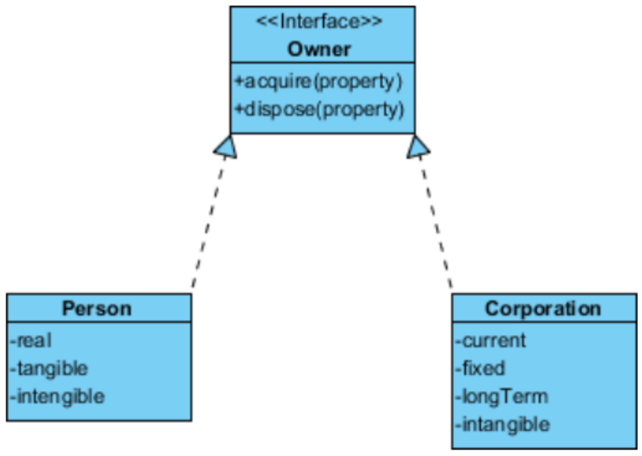
💻 Relationship between an interface class and the class containing its implementation details. The class is said to realize the interface class.
- Dotted Line with a unfilled arrow head (can vary sometimes filled) .

```
public interface A {

} // interface A

public class B implements A {

} // class B
```



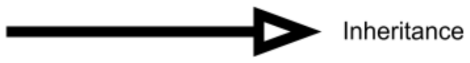
ClassB realizes ClassA

E.g. Interface might specify methods for acquiring property and disposing of property. The Person and Corporation classes need to implement these methods, possibly in very different ways.



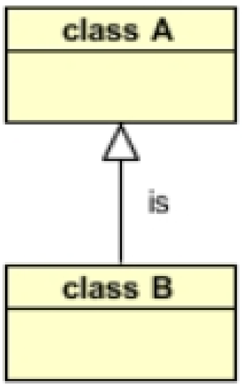
Generalization/Inheritance -

💻 A hierarchical relationship between a general classifier and a more specific classifier.
- Solid line with a unfilled arrow head.



- The specific classifier inherits the features of the general classifier.
    - subclass takes the functionality of a superclass
- “is-a” relationship
- • An abstract class name is shown in italics.
- Symbolized by a connected line with a closed arrowhead pointing to the superclass.

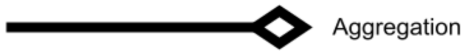
```
class A {
}

class B: public A {
}
```



ClassB is a ClassA

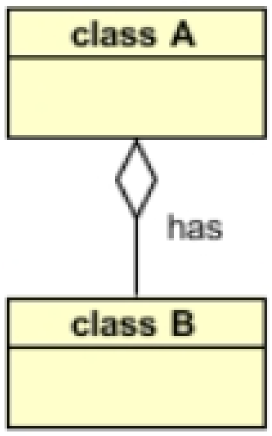
Aggregation -

💻 The target element is part of the source element.
- Solid line with a unfilled diamond head.



- “Has a” relationship
- Objects of ClassA and ClassB have separate lifetimes.
- If classA stores a reference to classB

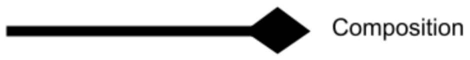
```
public class A {
  private:
    B b;
}
```



ClassA has a ClassB

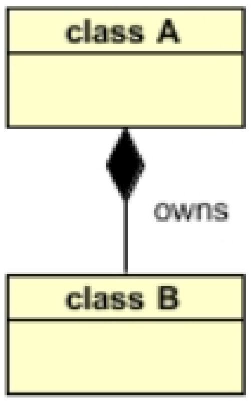
Composition -

💻 The object that contains another object is responsible for the creation and life cycle of the object it contains.
- Line with a filled diamond head.

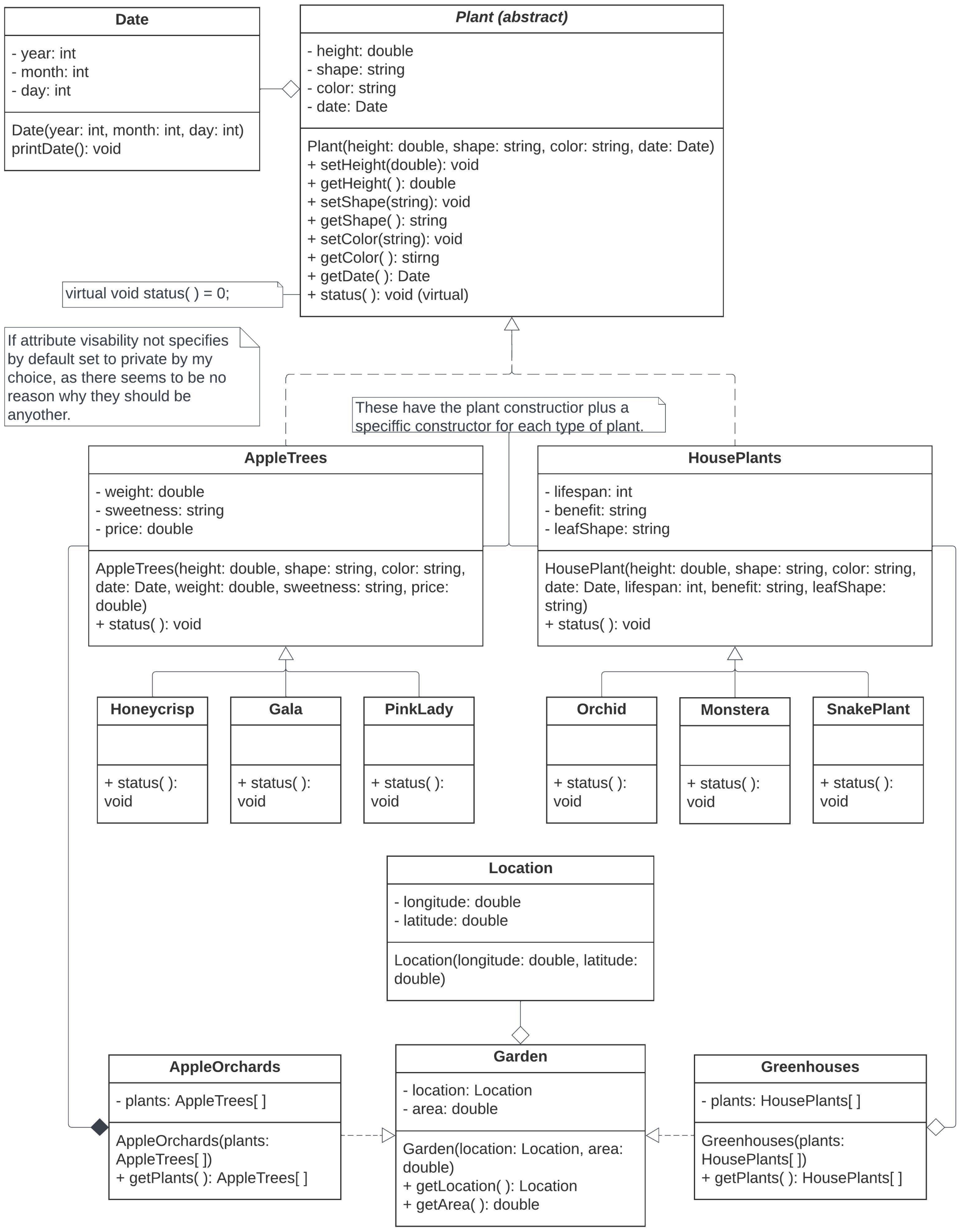


- Form of aggregation that is stronger.
- Objects of ClassB live and die with classA
- ClassB cannot exist by itself

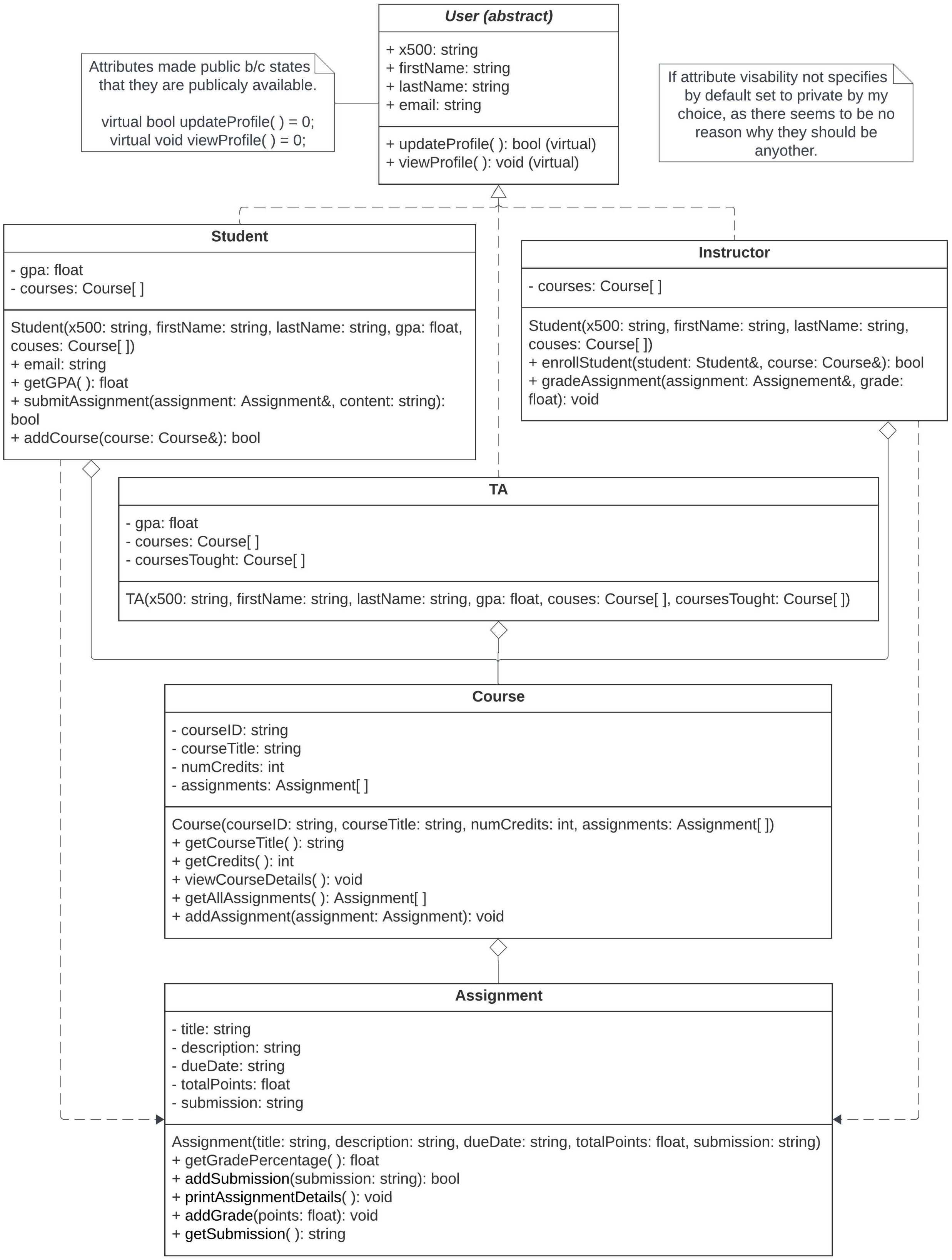
```
class A {
    private B _b = new B();
}

class A {
    private B _b;
    public A() {
        _b = new B();
    } // default constructor
}
```



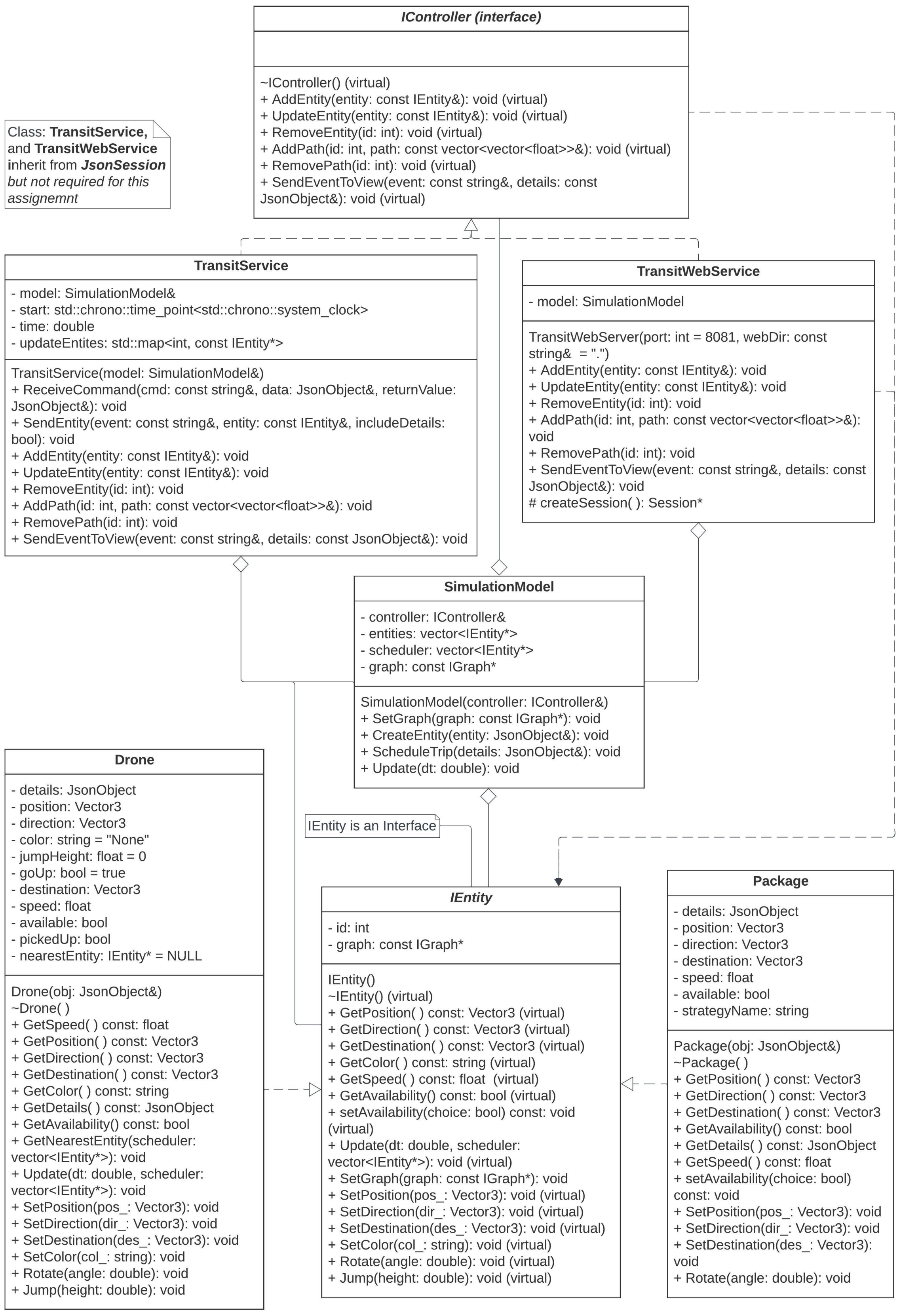
ClassA owns ClassB


Date
- year: int
- month: int
- day: int
Date(year: int, month: int, day: int)
printDate(): void
Plant (abstract)
- height: double
- shape: string
- color: string
- date: Date
Plant(height: double, shape: string, color: string, date: Date)
+ setHeight(double): void
+ getHeight( ): double
+ setShape(string): void
+ getShape( ): string
+ setColor(string): void
+ getColor( ): stirng
+ getDate( ): Date
+ status( ): void (virtual)
virtual void status( ) = 0;
If attribute visability not specifies by default set to private by my choice, as there seems to be no reason why they should be anyother.
These have the plant constructior plus a speciffic constructor for each type of plant.
AppleTrees
- weight: double
- sweetness: string
- price: double
AppleTrees(height: double, shape: string, color: string, date: Date, weight: double, sweetness: string, price: double)
+ status( ): void
HousePlants
- lifespan: int
- benefit: string
- leafShape: string
HousePlant(height: double, shape: string, color: string, date: Date, lifespan: int, benefit: string, leafShape: string)
+ status( ): void
Honeycrisp
+ status( ): void
Gala
+ status( ): void
PinkLady
+ status( ): void
Orchid
+ status( ): void
Monstera
+ status( ): void
SnakePlant
+ status( ): void
Location
- longitude: double
- latitude: double
Location(longitude: double, latitude: double)
AppleOrchards
- plants: AppleTrees[ ]
AppleOrchards(plants: AppleTrees[ ])
+ getPlants( ): AppleTrees[ ]
Garden
- location: Location
- area: double
Garden(location: Location, area: double)
+ getLocation( ): Location
+ getArea( ): double
Greenhouses
- plants: HousePlants[ ]
Greenhouses(plants: HousePlants[ ])
+ getPlants( ): HousePlants[ ]

UML Problem 2


User (abstract)
+ x500: string
+ firstName: string
+ lastName: string
+ email: string
+ updateProfile( ): bool (virtual)
+ viewProfile( ): void (virtual)
Attributes made public b/c states that they are publicaly available.
virtual bool updateProfile( ) = 0;
virtual void viewProfile( ) = 0;
If attribute visability not specifies by default set to private by my choice, as there seems to be no reason why they should be anyother.
Student
- gpa: float
- courses: Course[ ]
Student(x500: string, firstName: string, lastName: string, gpa: float, couses: Course[ ])
+ email: string
+ getGPA( ): float
+ submitAssignment(assignment: Assignment&, content: string): bool
+ addCourse(course: Course&): bool
Instructor
- courses: Course[ ]
Student(x500: string, firstName: string, lastName: string, couses: Course[ ])
+ enrollStudent(student: Student&, course: Course&): bool
+ gradeAssignment(assignment: Assignement&, grade: float): void
TA
- gpa: float
- courses: Course[ ]
- coursesTought: Course[ ]
TA(x500: string, firstName: string, lastName: string, gpa: float, couses: Course[ ], coursesTought: Course[ ])
Course
- courseID: string
- courseTitle: string
- numCredits: int
- assignments: Assignment[ ]
Course(courseID: string, courseTitle: string, numCredits: int, assignments: Assignment[ ])
+ getCourseTitle( ): string
+ getCredits( ): int
+ viewCourseDetails( ): void
+ getAllAssignments( ): Assignment[ ]
+ addAssignment(assignment: Assignment): void
Assignment
- title: string
- description: string
- dueDate: string
- totalPoints: float
- submission: string
Assignment(title: string, description: string, dueDate: string, totalPoints: float, submission: string)
+ getGradePercentage( ): float
+ addSubmission(submission: string): bool
+ printAssignmentDetails( ): void
+ addGrade(points: float): void
+ getSubmission( ): string

UML Problem 3


IController (interface)
~IController() (virtual)
+ AddEntity(entity: const IEntity&): void (virtual)
+ UpdateEntity(entity: const IEntity&): void (virtual)
+ RemoveEntity(id: int): void (virtual)
+ AddPath(id: int, path: const vector<vector<float>>&): void (virtual)
+ RemovePath(id: int): void (virtual)
+ SendEventToView(event: const string&, details: const JsonObject&): void (virtual)
Class: TransitService, and TransitWebService inherit from JsonSession but not required for this assignemnt
TransitService
- model: SimulationModel&
- start: std::chrono::time_point<std::chrono::system_clock>
- time: double
- updateEntites: std::map<int, const IEntity*>
TransitService(model: SimulationModel&)
+ ReceiveCommand(cmd: const string&, data: JsonObject&, returnValue: JsonObject&): void
+ SendEntity(event: const string&, entity: const IEntity&, includeDetails: bool): void
+ AddEntity(entity: const IEntity&): void
+ UpdateEntity(entity: const IEntity&): void
+ RemoveEntity(id: int): void
+ AddPath(id: int, path: const vector<vector<float>>&): void
+ RemovePath(id: int): void
+ SendEventToView(event: const string&, details: const JsonObject&): void
TransitWebService
- model: SimulationModel
TransitWebServer(port: int = 8081, webDir: const string& = ".")
+ AddEntity(entity: const IEntity&): void
+ UpdateEntity(entity: const IEntity&): void
+ RemoveEntity(id: int): void
+ AddPath(id: int, path: const vector<vector<float>>&): void
+ RemovePath(id: int): void
+ SendEventToView(event: const string&, details: const JsonObject&): void
# createSession( ): Session*
SimulationModel
- controller: IController&
- entities: vector<IEntity*>
- scheduler: vector<IEntity*>
- graph: const IGraph*
SimulationModel(controller: IController&)
+ SetGraph(graph: const IGraph*): void
+ CreateEntity(entity: JsonObject&): void
+ ScheduleTrip(details: JsonObject&): void
+ Update(dt: double): void
Drone
- details: JsonObject
- position: Vector3
- direction: Vector3
- color: string = "None"
- jumpHeight: float = 0
- goUp: bool = true
- destination: Vector3
- speed: float
- available: bool
- pickedUp: bool
- nearestEntity: IEntity* = NULL
Drone(obj: JsonObject&)
~Drone( )
+ GetSpeed( ) const: float
+ GetPosition( ) const: Vector3
+ GetDirection( ) const: Vector3
+ GetDestination( ) const: Vector3
+ GetColor( ) const: string
+ GetDetails( ) const: JsonObject
+ GetAvailability() const: bool
+ GetNearestEntity(scheduler: vector<IEntity*>): void
+ Update(dt: double, scheduler: vector<IEntity*>): void
+ SetPosition(pos_: Vector3): void
+ SetDirection(dir_: Vector3): void
+ SetDestination(des_: Vector3): void
+ SetColor(col_: string): void
+ Rotate(angle: double): void
+ Jump(height: double): void
IEntity is an Interface
IEntity
- id: int
- graph: const IGraph*
IEntity()
~IEntity() (virtual)
+ GetPosition( ) const: Vector3 (virtual)
+ GetDirection( ) const: Vector3 (virtual)
+ GetDestination( ) const: Vector3 (virtual)
+ GetColor( ) const: string (virtual)
+ GetSpeed( ) const: float (virtual)
+ GetAvailability() const: bool (virtual)
+ setAvailability(choice: bool) const: void (virtual)
+ Update(dt: double, scheduler: vector<IEntity*>): void (virtual)
+ SetGraph(graph: const IGraph*): void
+ SetPosition(pos_: Vector3): void (virtual)
+ SetDirection(dir_: Vector3): void (virtual)
+ SetDestination(des_: Vector3): void (virtual)
+ SetColor(col_: string): void (virtual)
+ Rotate(angle: double): void (virtual)
+ Jump(height: double): void (virtual)
Package
- details: JsonObject
- position: Vector3
- direction: Vector3
- destination: Vector3
- speed: float
- available: bool
- strategyName: string
Package(obj: JsonObject&)
~Package( )
+ GetPosition( ) const: Vector3
+ GetDirection( ) const: Vector3
+ GetDestination( ) const: Vector3
+ GetAvailability() const: bool
+ GetDetails( ) const: JsonObject
+ GetSpeed( ) const: float
+ setAvailability(choice: bool) const: void
+ SetPosition(pos_: Vector3): void
+ SetDirection(dir_: Vector3): void
+ SetDestination(des_: Vector3): void
+ Rotate(angle: double): void

UML Problem 4