**CSCI 2302**

**OOT Chapter – Aggregation Lab**

Table of Contents

[Intro 1](#_Toc82354539)

[Learning Goals 1](#_Toc82354540)

[Notes 2](#_Toc82354541)

[Has-a 2](#_Toc82354542)

[Task 2](#_Toc82354543)

[Submit 4](#_Toc82354544)

Intro: Understanding the concepts of Object-Oriented Programming is the purpose of this course. The overall approach of using an OOP approach is to focus on objects verses a procedural approach. By focusing on objects, we can encapsulate states/fields/data and behaviors/methods to create an object that has a tightly coupled data aspect, high cohesion within itself and low cohesion with the program using the object.

Objects can interact with other object and objects be made up of other objects. Understanding the relationship of objects – between them and comprising of them is a core concept of Object-Oriented Thinking, which is the purpose of this chapter. As such, this lab focuses on how an object can be made up of other objects. In specific, we will be looking at the aggregation relationship.

Learning Goals: This lab’s goal is to understand the concept of aggregated objects. To demonstrate this concept, we will create an aggregated object, thus we are applying this concept; using the aggregated object, thus we are understanding and analyzing this concept.

Notes: An aggregation relationship is a special form of association[[1]](#footnote-1) that represents an ownership relationship between two objects.

Both objects can exist on their own – one does not depend on the other to exist.

If we look at the definition of aggregation:

* is a collection, or the gathering of things together
* a group or mass of distinct or varied things
* constituting or amounting to a whole – total

Aggregation comes from the Latin *ad*, meaning to, and *gregare*, meaning herd. If you think of a herd of cattle, a cow or a bull can exist on its own – a cow or bull does not have to be a part of a herd. The herd is the aggregated object made up of cows and bulls.

* Aggregation implies a relationship where the child can exist independently of the parent. Example: ClassroomForCourse (parent) and Student (child). Delete the ClassroomForCourse and the Students still exist.
* It's important to note that the aggregation link doesn't state in any way that Class A owns Class B nor that there's a parent-child relationship (when parent deleted all its child's are being deleted as a result) between the two. Actually, quite the opposite! The aggregation link is usually used to stress the point that Class A instance is not the exclusive container of Class B instance, as in fact the same Class B instance has another container/s.
* An aggregation relationship is usually represented as a data field in the aggregating class.

Aggregating object = owner

Aggregated object = subject, an object that can exist on its own

## Has-a

Aggregation is *has-a* relationship. But, the object that makes up the aggregated object ***can exist*** on its own.

Task: Complete the steps outlined below in a file named AggregatedObject.java.

1. Download the file AggregatedObject.java from the Dropbox in Brightspace by D2L and rename the file with yourMySFAusername\_AggregatedObject.java.
2. Complete the object, Gear, it has the following UML diagram.

|  |
| --- |
| **Gear** |
| -howManyCogs: int  -diameter: double |
| +Gear()  +Gear(howManyCogs: int, diameter: double)  +setHowManyCogs(howManyCogs: int): void  +getHowManyCogs(): int  +setDiameter(diameter: double): void  +getDiameter(): double  +toString(): String |

1. Analyze and understand the aggregated object, WindUpWatch, in the AggregatedObject file based on the following UML diagram. This applies the concept of objects and aggregation of objects.

|  |
| --- |
| **WindUpWatch** |
| -crown: String  -minuteHandIsAt: int  -hourHandIsAt: int  -gear: Gear |
| +WindUpWatch()  +WindUpWatch(crown: String, minuteHandIsAt: int, hourHandIsAt: int, gear: Gear)  +setMinuteHandIsAt(minuteHandIsAt: int): void  +setHourHandIsAt(hourHandIsAt: int): void  +getMinuteHandIsAt(): int  +getHourHandIsAt(): int  +setGearCogs(howManyCogs: int): void  +toString(): String |

1. Apply, analyze, and understand aggregated objects by completing the following in the AggregatedObject file:
   1. Instantiate a Gear object using the identifier of gear, with the following data fields:
      1. howManyCogs: 10
      2. diameter: 5

This is the object that will make up an aggregated object. A Gear can exist on its own, without being a part of a watch; it can also make up drills, lifts, bicycles, and many other mechanical engines and devices.

* 1. Instantiate a WindUpWatch object with the following data fields:
     1. crown: regular
     2. minuteHandIsAt: 00
     3. hourHandIsAt: 10
     4. gear: gear (from the previous step)

This is the aggregated object, it has-a Gear object.

* 1. Invoke the WindUpWatch toString method.

Analyze the statement to see the make-up of the aggregated object, see the Gear object as part of the WindUpWatch object.

* 1. Modify (invoke the setGearCogs method) the howManyCogs to the value of 50

When you want to access the object that makes up the aggregated object, you can create methods to ensure that you are using the proper channels to keep the integrity of the data and adhering to the pillar of OOP – data encapsulation.

* 1. Invoke the WindUpWatch toString method.

Understand the change you did in the aggregated object.

Submit: Submit yourMySFAusername\_AggregatedObject.java in the Dropbox in Brightspace by D2L.

1. An associated relationship is said to be a binary relationship; it is a relationship that describes the activity between 2 classes. Basically, this relationship description provides information of how many & direction of interaction between the objects.

   Definition of association: An organization of people with a common purpose and having a formal structure. [↑](#footnote-ref-1)