CPSC-224
Software Development
Abstraction and Inheritance

Yu Wang

wangy2@gonzaga.edu

Feb 10, 2025

Announcement

- Homework2 deadline: Feb 28)
- Homework0 - Career Development 01 – First Resume Draft
  - Deadline: March 3rd
- Homework1 (Check the updates on Canvas)
  - CPSC 224 - GitHub Classroom Assignment Instructions
  - Deadline for submission: Feb. 14

Daily Attendance (01)

- Scan the QR Code for yourself

Daily Attendance (02)

- Scan the QR Code for yourself

Review - Last Class

GONZAGA UNIVERSITY

- ✓ We learned – What is encapsulation? It basically means we should bundle the data and methods that operate the data in a single unit or object.
- ✓ We learned – Getting methods for retrieving information and Setting methods for changing information.

Abstraction

- This section we will be looking at the second of the four main principles of object-oriented programming:
  - Encapsulation
  - **Abstraction**
  - Inheritance
  - Polymorphism

Abstraction

- This segment we will be looking at the next of the four main principles of object-oriented programming, abstraction.

- Abstraction refers to only showing essential details and keeping everything else hidden.

Abstraction – Car Example

How the steering wheel steers the car
How the gas and brake pedals affect the car
How much gas your car has

Abstraction – Car Example

How exactly the car functions internally

As long as you understand the outcome, the process is not very important to you

Abstraction

- This idea(Car example) extends to object-oriented programming.

- The classes you create should act like your car. Users of your classes should not worry about the inner details of those classes.

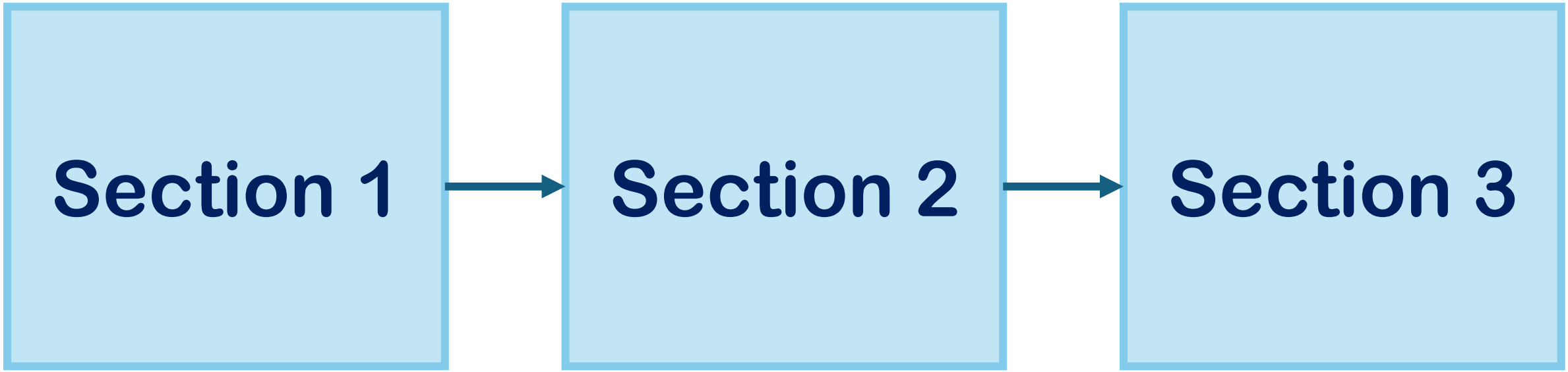
Abstraction

- This idea similar to encapsulation that was discussed last class.

- Classes should not directly interact with other classes' data.

Abstraction

- This is very important when working on your program incrementally.



Abstraction

- Modern programs are very complex to the point where multiple programmers tend to work on one program

- In this case, it's best if the section that you work on is able to function without knowledge of the inner workings of your colleague's section

Abstraction – Interface and Implementation

Interface

Implementation

Abstraction – Interface and Implementation

GONZAGA UNIVERSITY

Interface

- The interface refers to the way sections of code can communicate with one another
- This typically is done through methods that each class is able to access

Abstraction – Interface and Implementation

GONZAGA UNIVERSITY

Implementation

- The implementation of these methods, or how these methods are coded, should be hidden.

Abstraction – Car Example

car. pushGas() → Car moves forward

The “how” is not important

Abstraction – Code Example

```
abstract class Animal{
    //Abstract method
    abstract void makeSound();

}
class Dog extends Animal{
    //Subclass must implement the abstract method
    void makeSound(){
        System.out.println("Bark!");
    }
}
```

Abstraction – Overview

Abstraction allows the program to be worked on incrementally and prevents it from becoming entangled and complex

Determine specific points of contact that can act as an interface between classes, and only worry about the implementation when coding it.

Abstraction vs Encapsulation

GONZAGA UNIVERSITY

What is the difference between abstraction and encapsulation?

Abstraction focuses on hiding the implementation details and showing only the necessary features, while encapsulation focuses on bundling data and methods that operate on the data into a single unit and restricting access to some of the object's components.

Abstraction

- This section we will be looking at the third of the four main principles of object-oriented programming:
  - Encapsulation
  - Abstraction
  - **Inheritance**
  - Polymorphism

Inheritance

- This segment we will be looking at the next of the four main principles of object-oriented programming: Inheritance.

- Inheritance is the principle that allows classes to derive from other classes.

Inheritance – Game Example

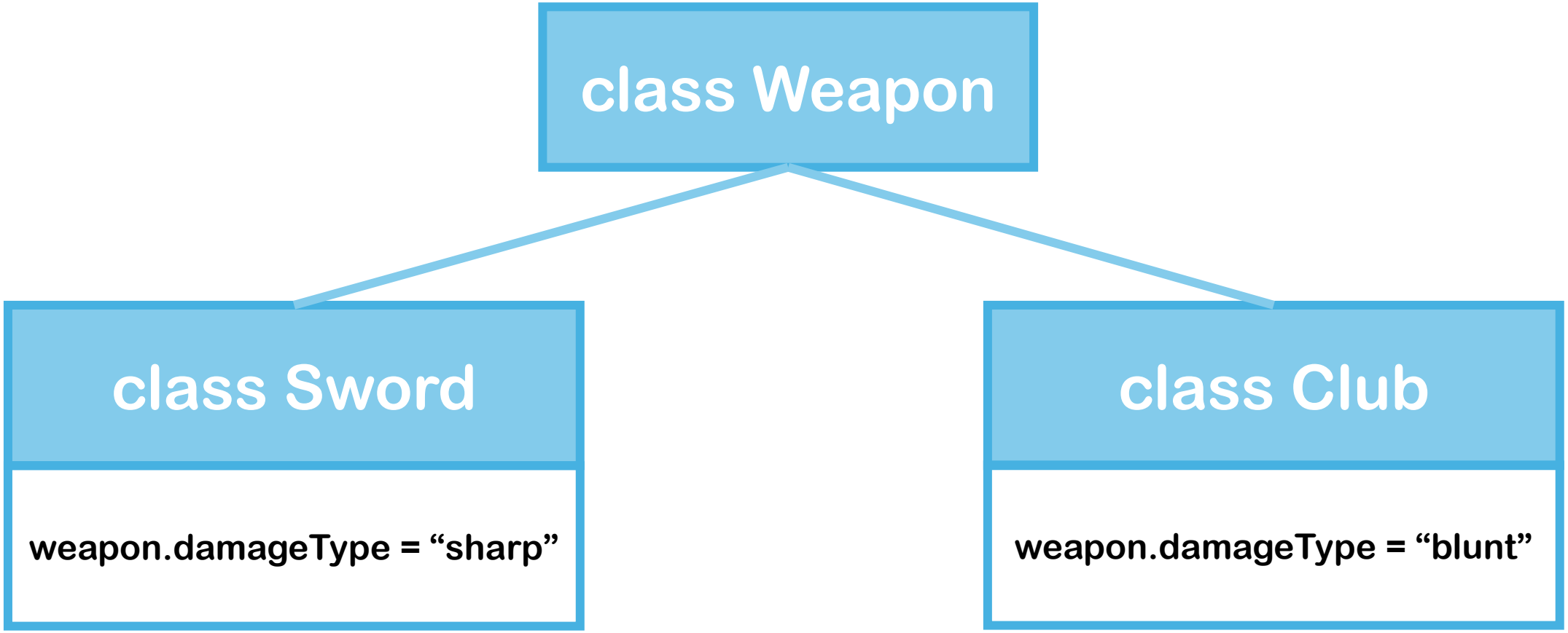
Class Weapon

Contains methods and attributes common to all weapons

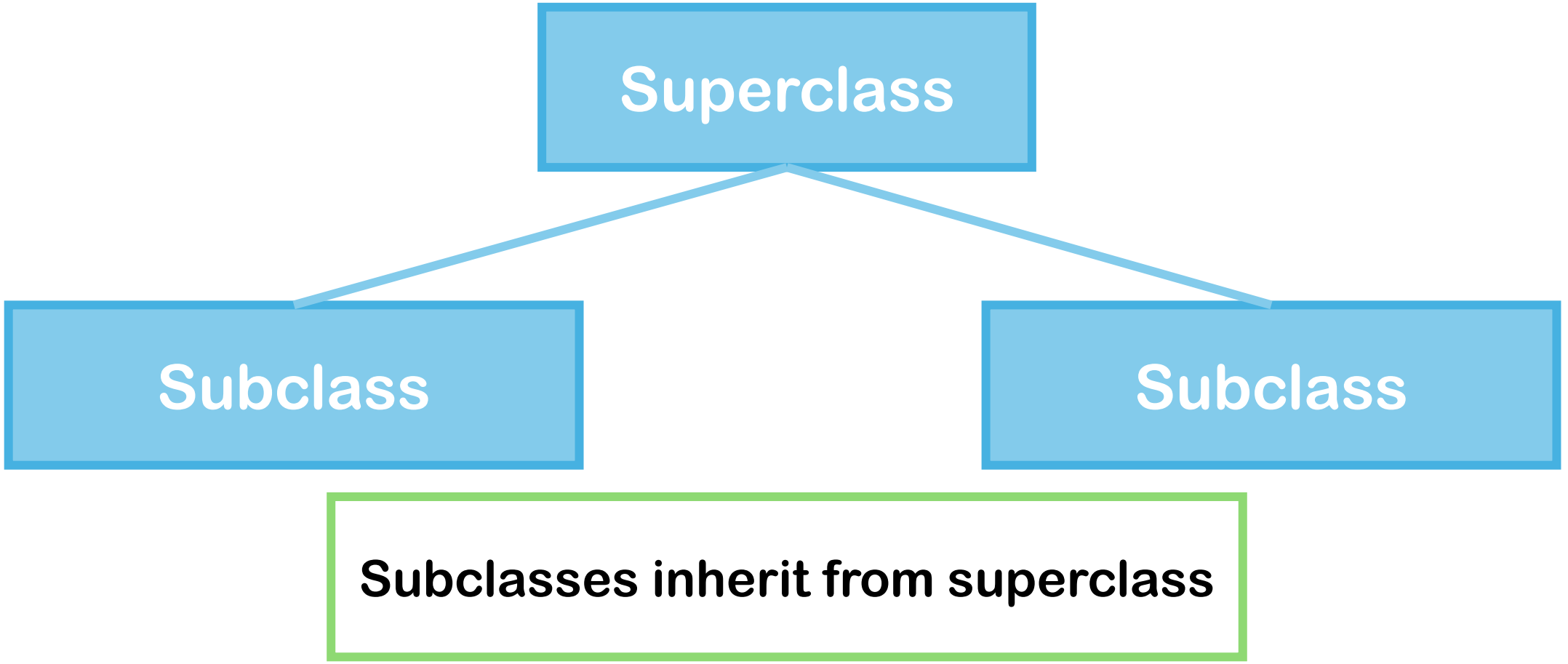
Weapon.Damage

Weapon.Attack()

Inheritance – Game Example



Inheritance – Game Example

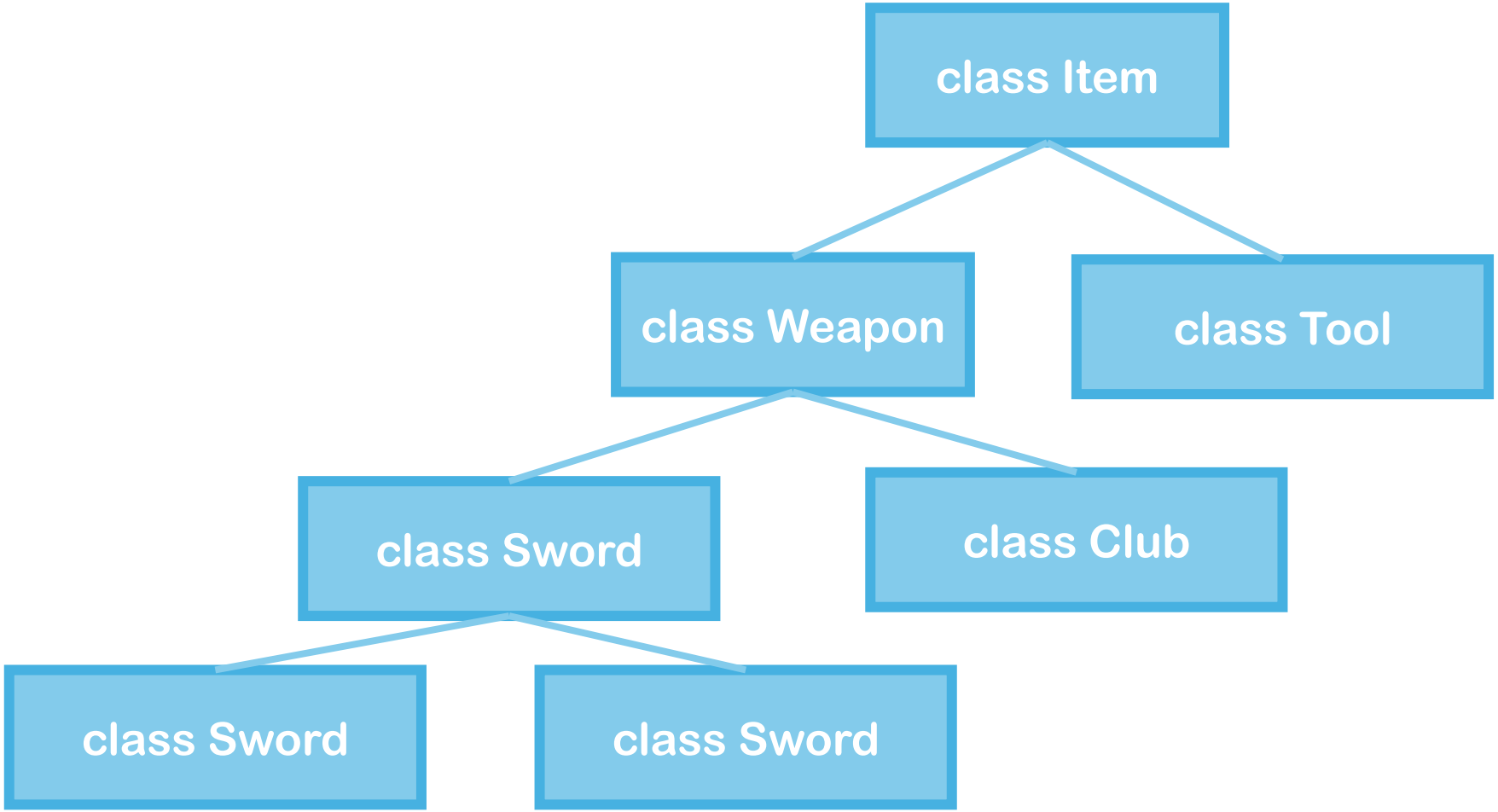


Inheritance – Class Hierarchy

- Any sword or club would require the methods and attributes present in the weapon class in order to function.

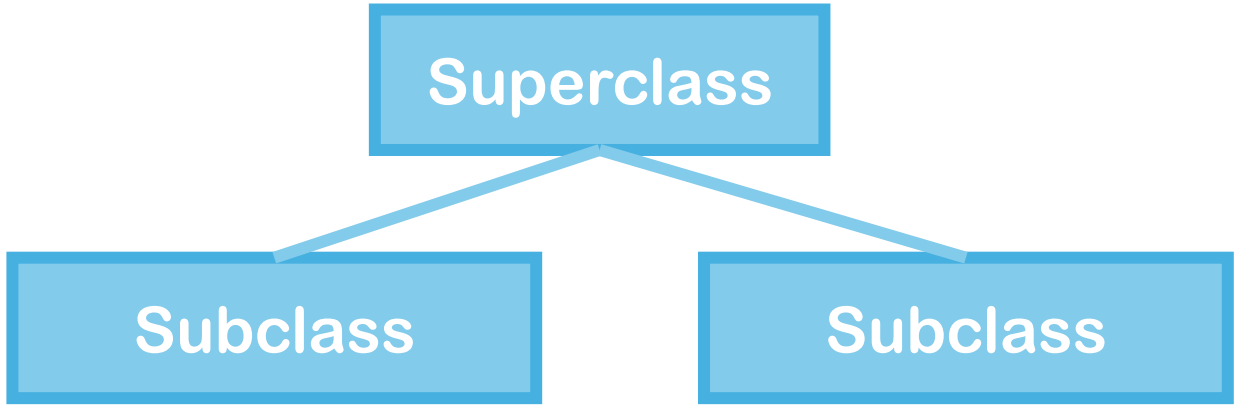
- In most cases the class hierarchy you create will have many more layers with many more classes in each layer.

Inheritance – Class Hierarchy



Inheritance – Class Hierarchy

The class hierarchy acts as a web of classes with different relationships to one another



Inheritance – Access Modifiers

GONZAGA UNIVERSITY

Access modifiers change which classes have access to other classes, methods, or attributes

The three main access modifiers we will be covering are:

- Public
- Private
- Protected

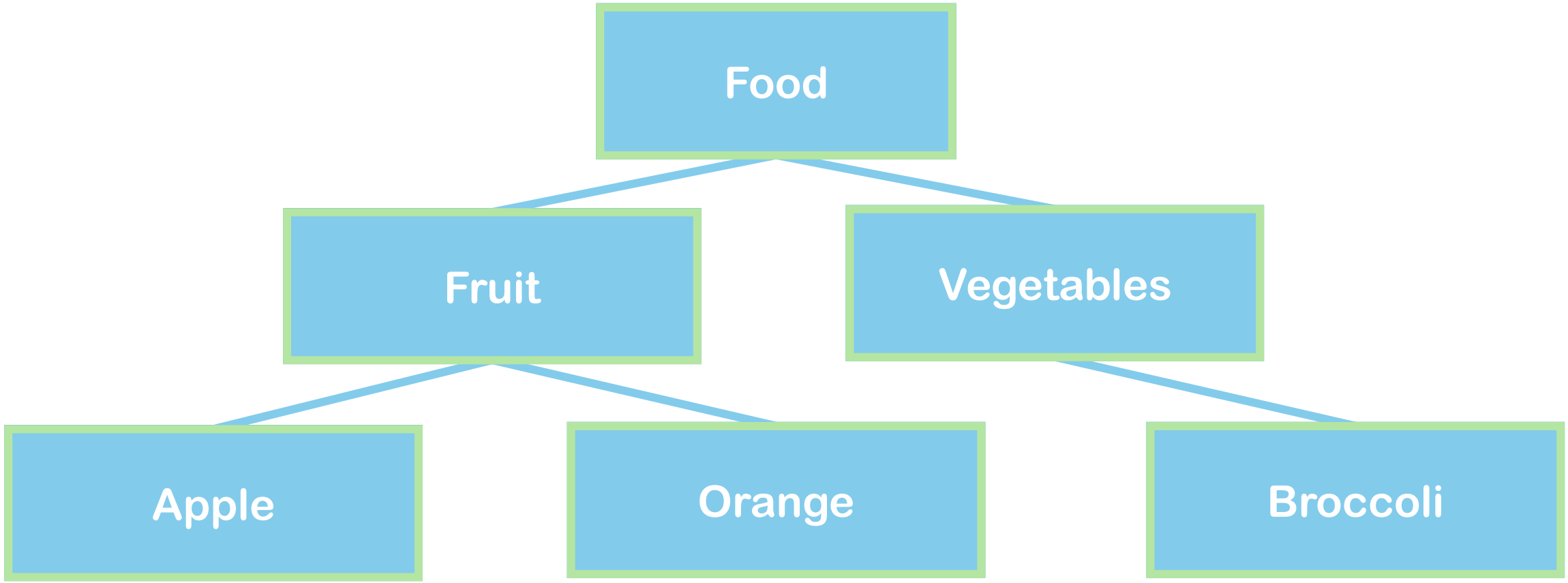
Inheritance – Access Modifiers

GONZAGA UNIVERSITY

Public members can be accessed from anywhere in your program

This includes anywhere both inside of the class hierarchy it is defined as well as outside in the rest of the program

Inheritance – Example



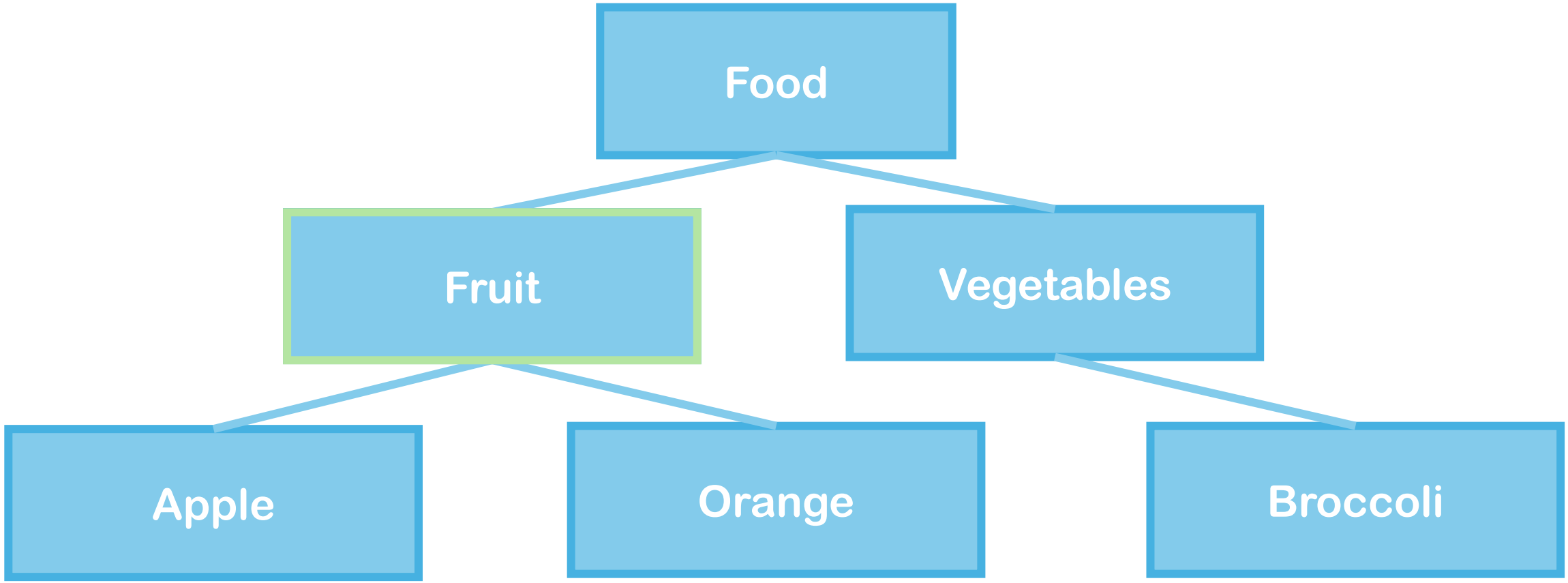
Inheritance – Access Modifiers

GONZAGA UNIVERSITY

Private members can only be accessed from within the same class that the member is defined

This allows you to create multiple private members of the same name in different locations so that they do not conflict with one another

Inheritance – Example



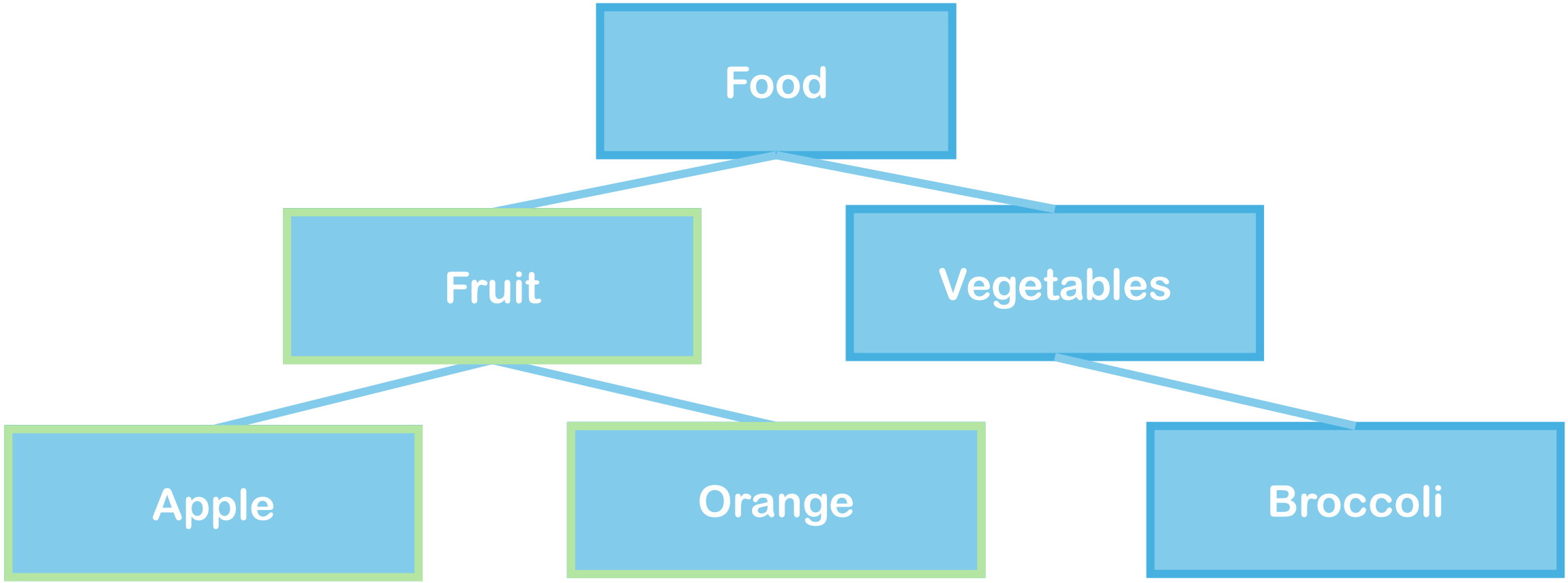
Inheritance – Access Modifiers

GONZAGA UNIVERSITY

Protected members can only be accessed within the class it is defined, as well as any subclasses of that class.

This essentially makes protected members private to the hierarchy in which they are defined.

Inheritance – Example



Question

Can an abstract method be private, static, or final in Java?

Why or why not?

Question(Answer)

GONZAGA UNIVERSITY

No, we **cannot** use `private`, `static`, or `final` modifiers in abstraction in Java. Here's why:

1. `private`:
   - Abstraction is achieved using **abstract classes** and **interfaces**, where methods are meant to be **overridden** by subclasses.
   - A `private` method is **not accessible** outside the class, making it **impossible to override** in a subclass.
   - Hence, **abstract methods cannot be private**.
2. `static`:
   - An abstract method is meant to be **overridden** by subclasses, but a `static` method **belongs to the class itself** and **cannot be overridden**.
   - Since abstraction depends on polymorphism (method overriding), **abstract methods cannot be static**.
3. `final`:
   - A `final` method **cannot be overridden**, while an abstract method **must be overridden**.
   - This creates a contradiction, so **abstract methods cannot be final**.

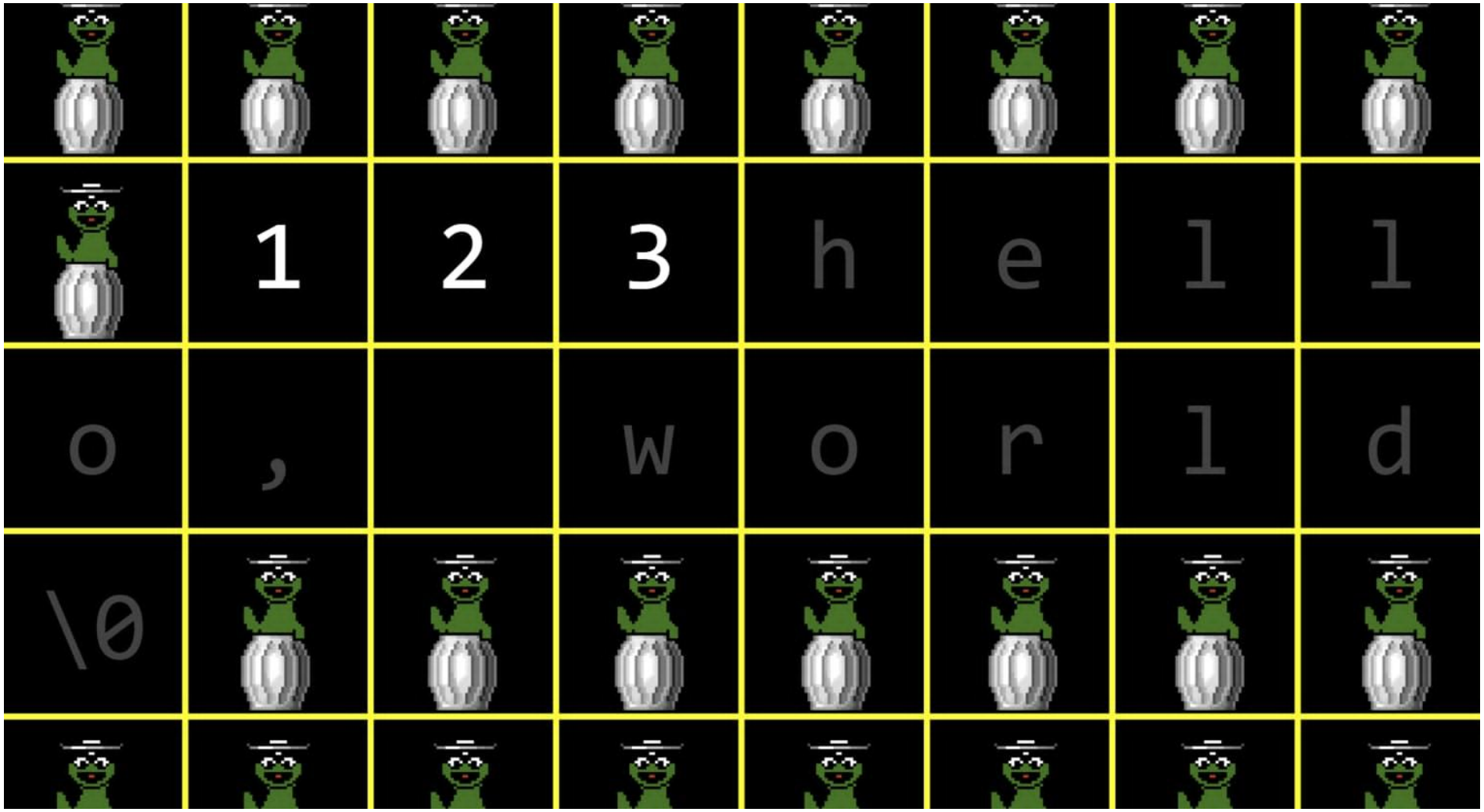
Supplement - Garbage Collection

What is Automatic Garbage Collection?

- Automatic garbage collection is the process of looking at heap memory, identifying which objects are in use and which are not, and deleting the unused objects. An in use object, or a referenced object, means that some part of your program still maintains a pointer to that object. An unused object, or unreferenced object, is no longer referenced by any part of your program. So the memory used by an unreferenced object can be reclaimed.

- In a programming language like C, allocating and deallocating memory is a manual process. In Java, process of deallocating memory is handled automatically by the garbage collector. The basic process can be described as follows.

Supplement – Memory Allocation



Any Questions?