COMP2511

Week 10

TUESDAY 9AM - 12PM (T09B)

TUESDAY 1PM - 4PM (T13B)

WEDNESDAY 6PM - 9PM (W18A)

Updates

- Last day to do any lab marking. If its not marked, it will just be 0.
- It is on you to get all your labs marked off

- Assignment-ii feedback will be less detailed.
- You may request for a more detailed feedback once you get your mark.

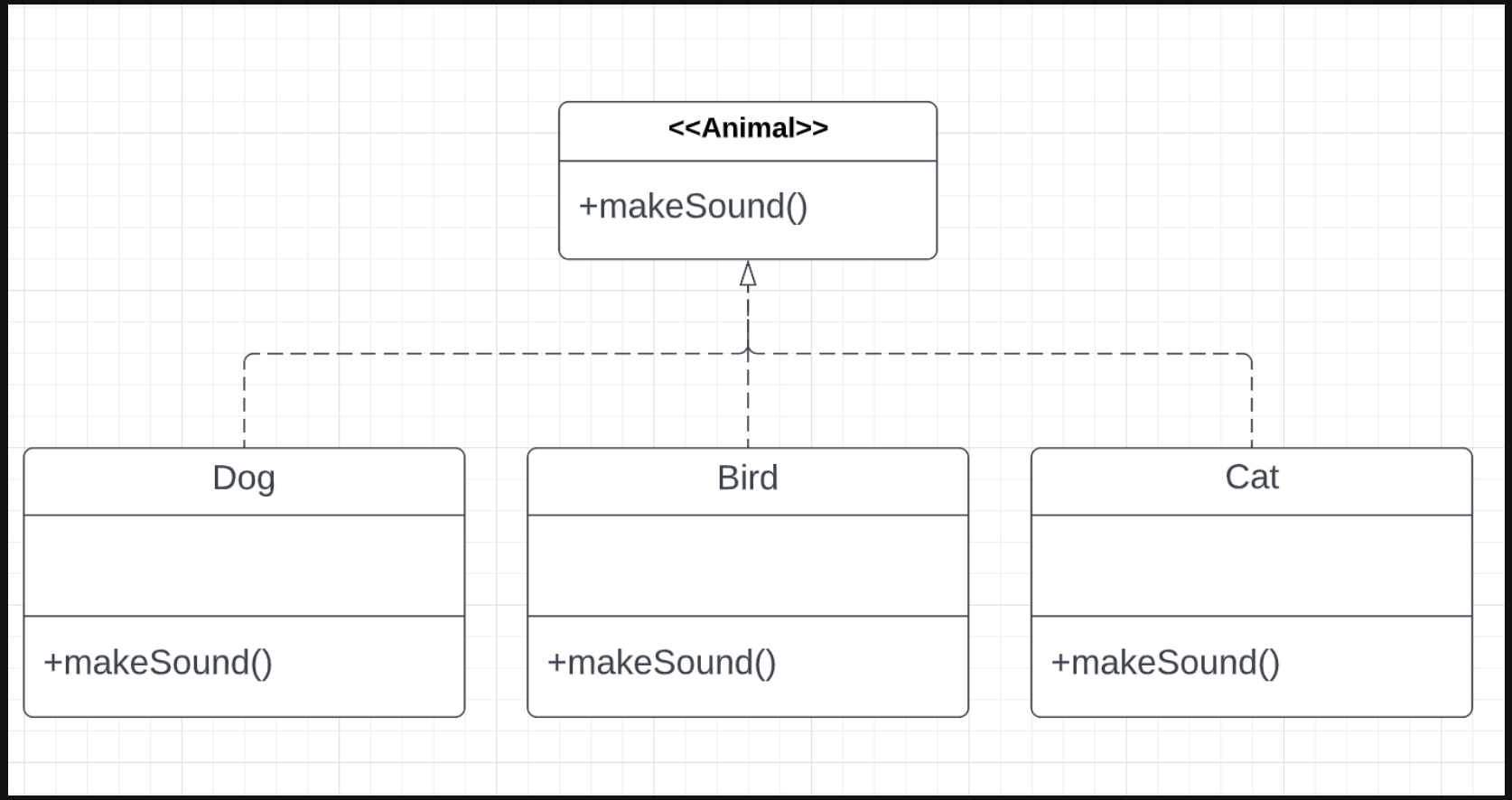
MyExperience

Please fill it in (10 mins)

Visitor Pattern

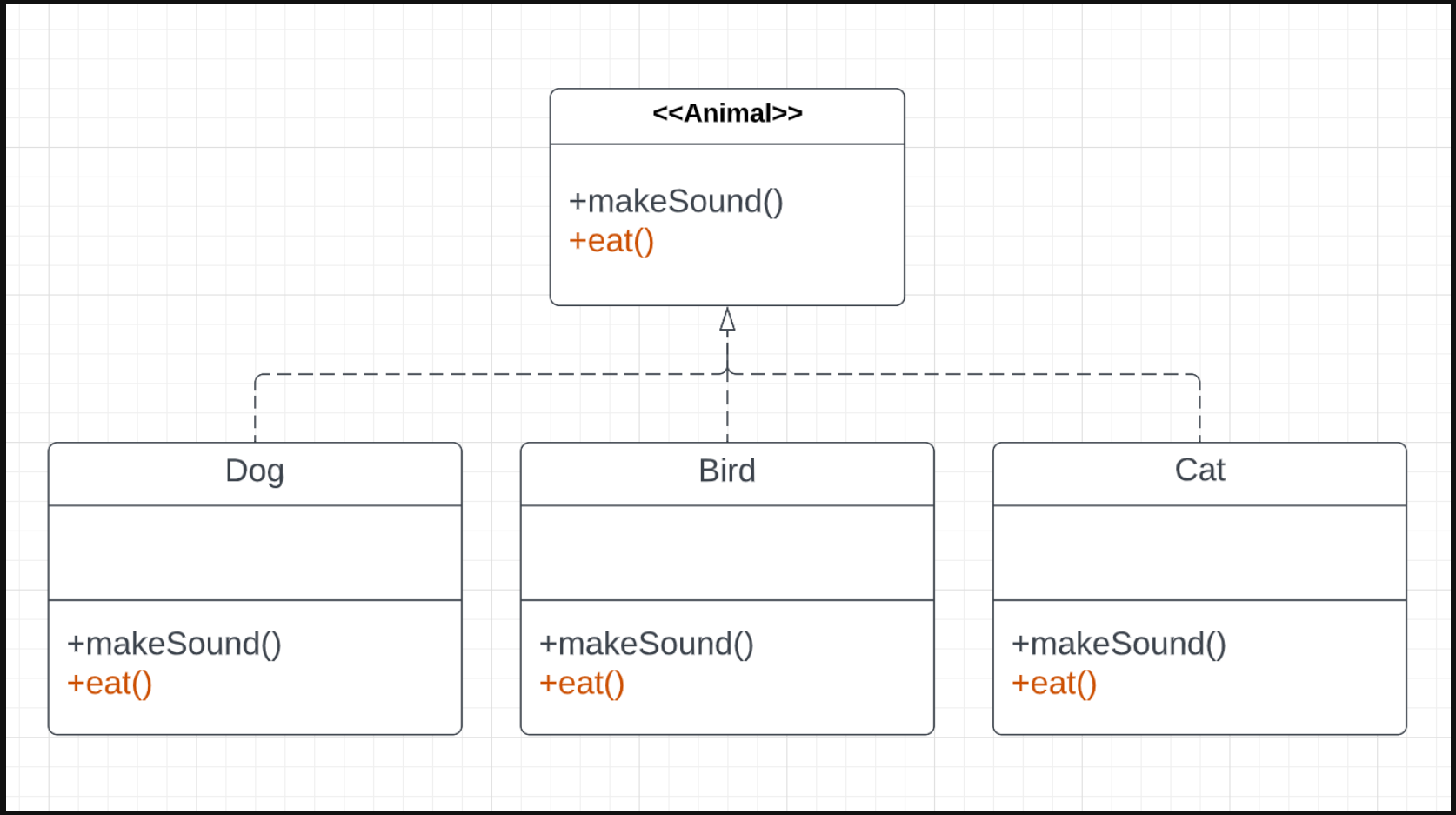
Visitor Pattern

Problem: How do I add extra functionalities to subclasses without violating open/closed principle.



Visitor Pattern

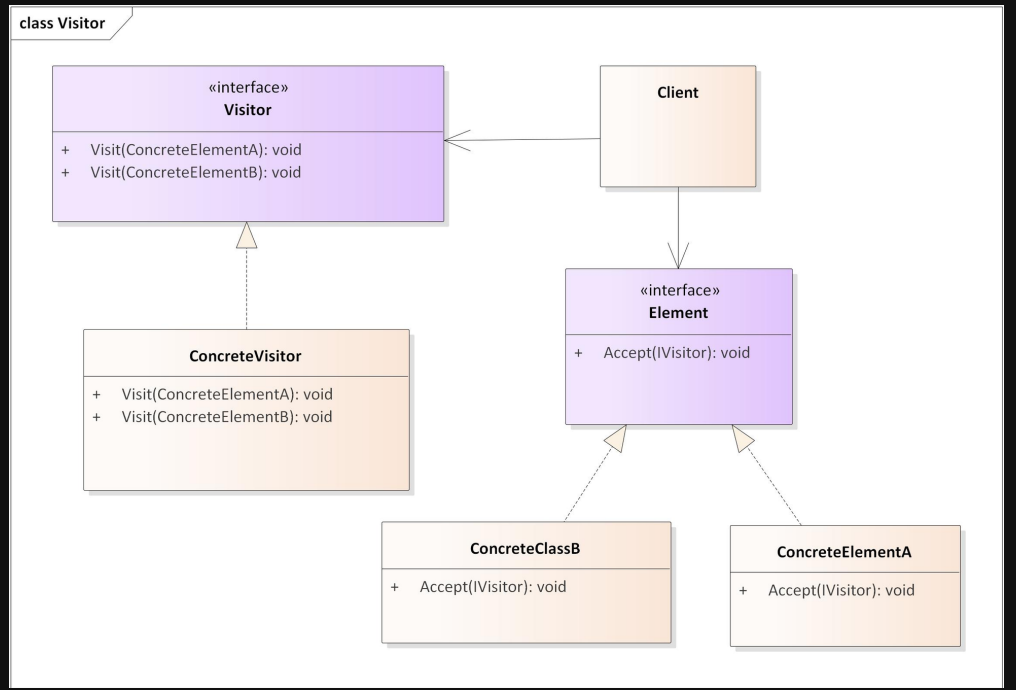
Problem: How do I add extra functionalities to subclasses without violating open/closed principle.

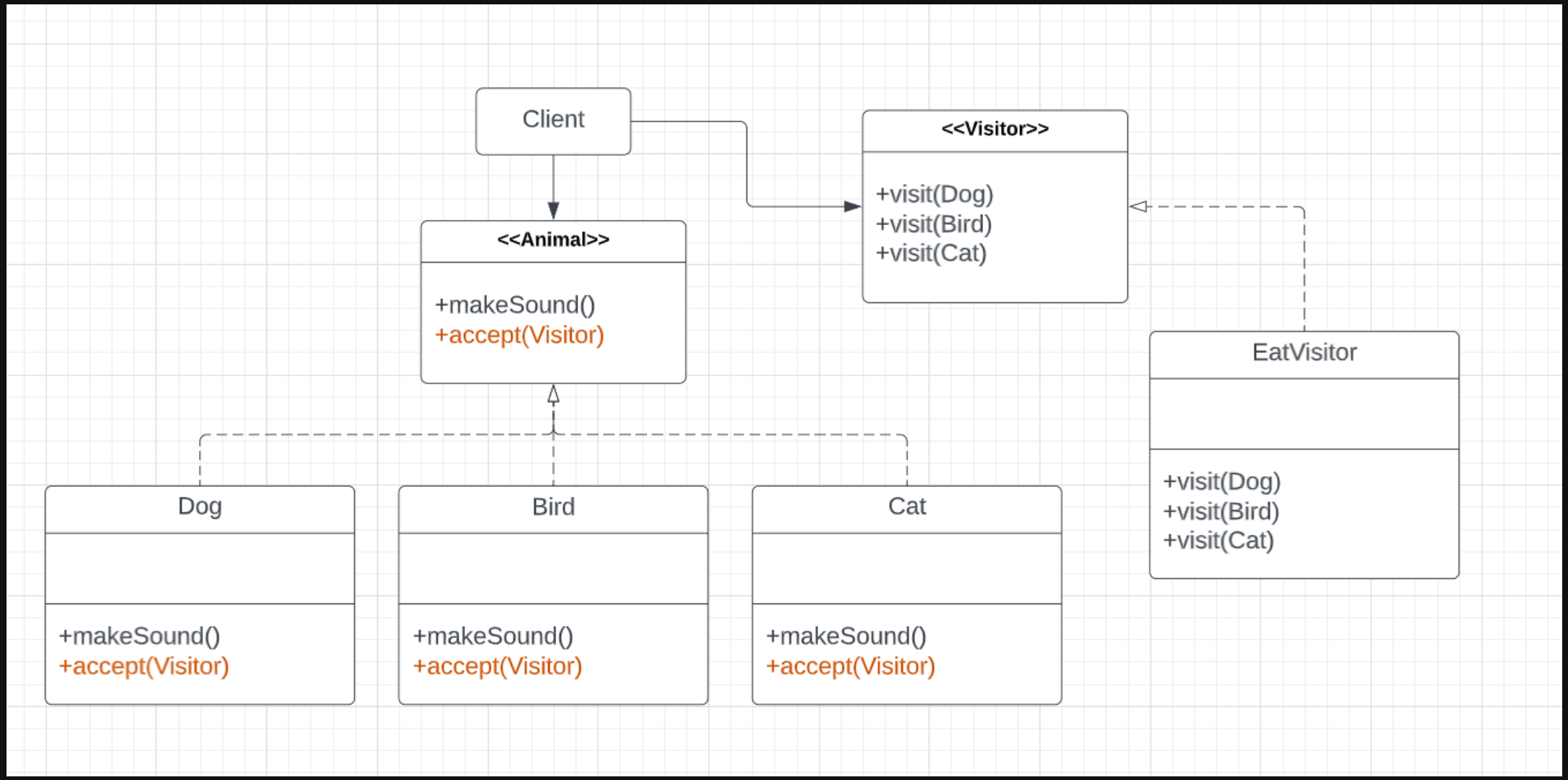


Visitor Pattern

Behavioural Pattern

- Adds extra functionality to class without modifying the original (abides by open closed principle)
- One class/interface (visitor) defines a computation/operation and another (visitable) is responsible for providing data access




Client
<<Visitor>>
+visit(Dog)
+visit(Bird)
+visit(Cat)
<<Animal>>
+makeSound()
+accept(Visitor)
EatVisitor
+visit(Dog)
+visit(Bird)
+visit(Cat)
Dog
+makeSound()
+accept(Visitor)
Bird
+makeSound()
+accept(Visitor)
Cat
+makeSound()
+accept(Visitor)

Kahoot

Exam Tips

Exam Tips

- Practice good exam techniques.
- Get familiar with writing Java & Generics
- Learn the patterns and the differences (YouTube has some really good resources if you don't understand certain ones).
- Know code smells and methods of refactoring

Code Demo

Computer.java

Code Demo

In this scenario we have Computers, Keyboards and Mouses which all are of type **`ComputerComponent`**. We want to be able to 'visit' different types of Computer components by logging the following messages:

```
1 Looking at computer Corelli with memory 500 GB.
2 Looking at keyboard Mechanical keyboard which has 36 keys.
3 Looking at mouse Bluetooth mouse.
```

In particular though, anyone which is visiting a **`Computer`** must be **validated** prior to being able to visit.

Extend/modify the starter code to use the Visitor Pattern to allow different computer components to be visited.

Revision

1. Behavioural Patterns

> Behavioural patterns are patterns concerned with **algorithms** and the **assignment of responsibility** between object

Behavioural design patterns are design patterns that identify **common communication patterns** among objects and realize these patterns. By doing so, these patterns increase **flexibility** carrying out this communication.

- **Iterator pattern**: Iterators are used to access the elements of an **aggregate object** sequentially without exposing its underlying representation
- **Observer pattern**: Objects register to observe an event that may be raised by another object. Also known as publish/subscribe or **event listener**
- **Strategy pattern**: Algorithms can be selected at runtime, using composition
- **State pattern**: A clean way for an object to partially change its type at runtime
- **Template method pattern**: Describes the program skeleton of a program; algorithms can be selected at runtime using inheritance
- **Visitor pattern**: A way to separate an algorithm from an object

1. Creational Pattern

> Provides object creation mechanisms which increase the flexibility and reuse of existing code

Creational patterns provide various **object creation** mechanisms, which increase flexibility and reuse of existing code.

It tries to create objects in a manner **suitable** to the situation.

- **Factory method**: provides an interface for creating objects in a superclass, but allows subclass to alter the types of objects that will be created
- **Abstract factory**: lets user produce families of related objects without specifying their concrete classes
- **Builder**: lets users construct complex objects step by step. The pattern allows user to produce different types and representations of an object using the same construction code
- **Singleton**: lets user ensure that a class has only one instance, while providing a global access point to this instance

1. Structural Patterns

> Explain how to assemble objects and classes into larger structures, while keeping these strructures flexible and efficient

Structural design patterns are design patterns that ease the design by identifying a simple way to **realize relationships** among entities

- **Adapter pattern**: 'adapts' one interface for a class into one that a client expects
- **Composite pattern**: a tree structure of objects where every object has the **same interface**
- **Decorator pattern**: add additional functionality to a class at **runtime** where sub-classing would result in an exponential rise of new classes

Attendance

Feedback

https://forms.gle/R4sMTTQzPC4vqXSN8