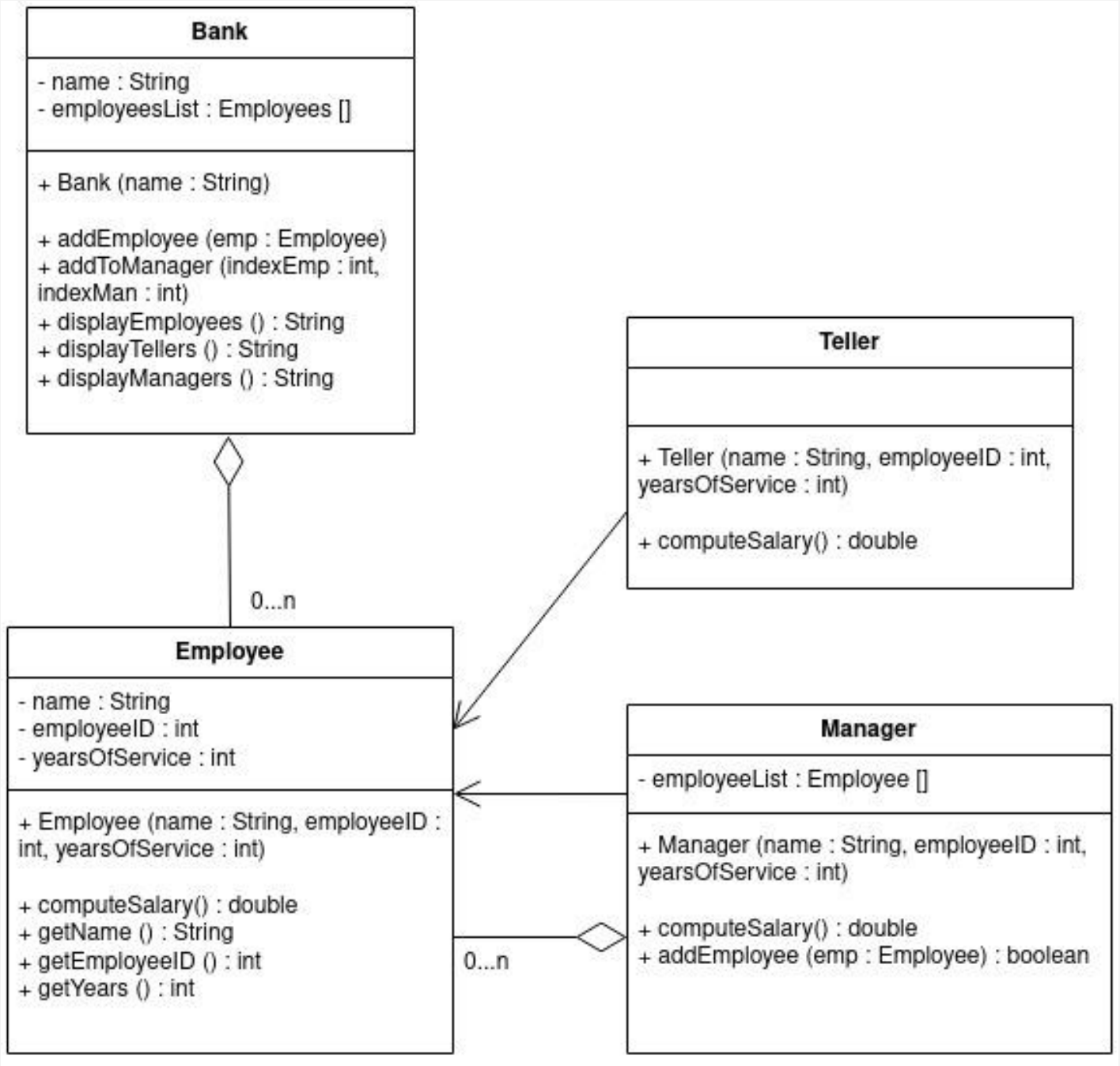
Object-Oriented Programming

Abstraction

Abstract Classes + Methods and Interfaces

Outline

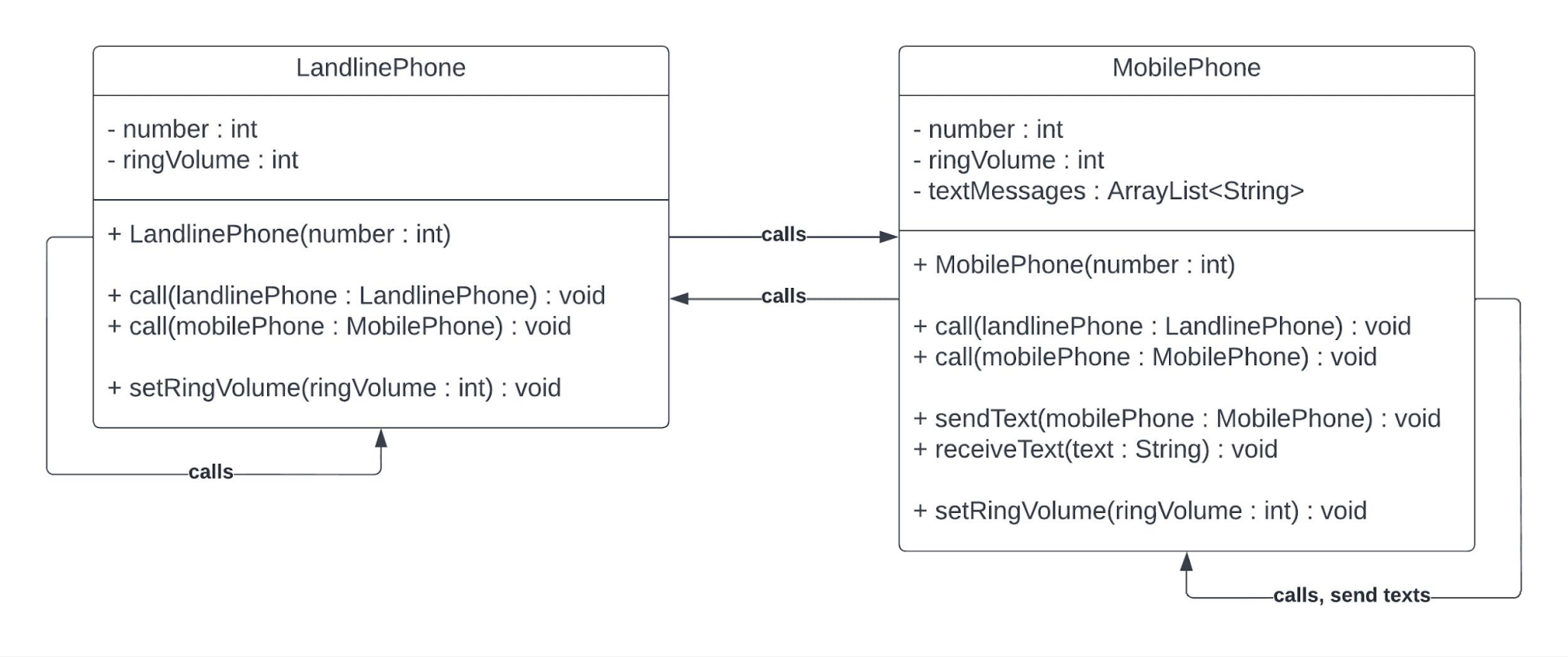
- Sharing of Student-Registrar GUI/MVC
- Review of Exercise 6 (Bank scenario)
- Review of Practice Exercise 9
- Abstraction
  - Abstract Class
  - Abstract Method
  - Interface


Bank
- name : String
- employeesList : Employees []
+ Bank (name : String)
+ addEmployee (emp : Employee)
+ addToManager (indexEmp : int, indexMan : int)
+ displayEmployees () : String
+ displayTellers () : String
+ displayManagers () : String
0...n
Teller
+ Teller (name : String, employeeID : int, yearsOfService : int)
+ computeSalary() : double
Employee
- name : String
- employeeID : int
- yearsOfService : int
+ Employee (name : String, employeeID : int, yearsOfService : int)
+ computeSalary() : double
+ getName () : String
+ getEmployeeID () : int
+ getYears () : int
0...n
Manager
- employeeList : Employee []
+ Manager (name : String, employeeID : int, yearsOfService : int)
+ computeSalary() : double
+ addEmployee (emp : Employee) : boolean

Questions? 

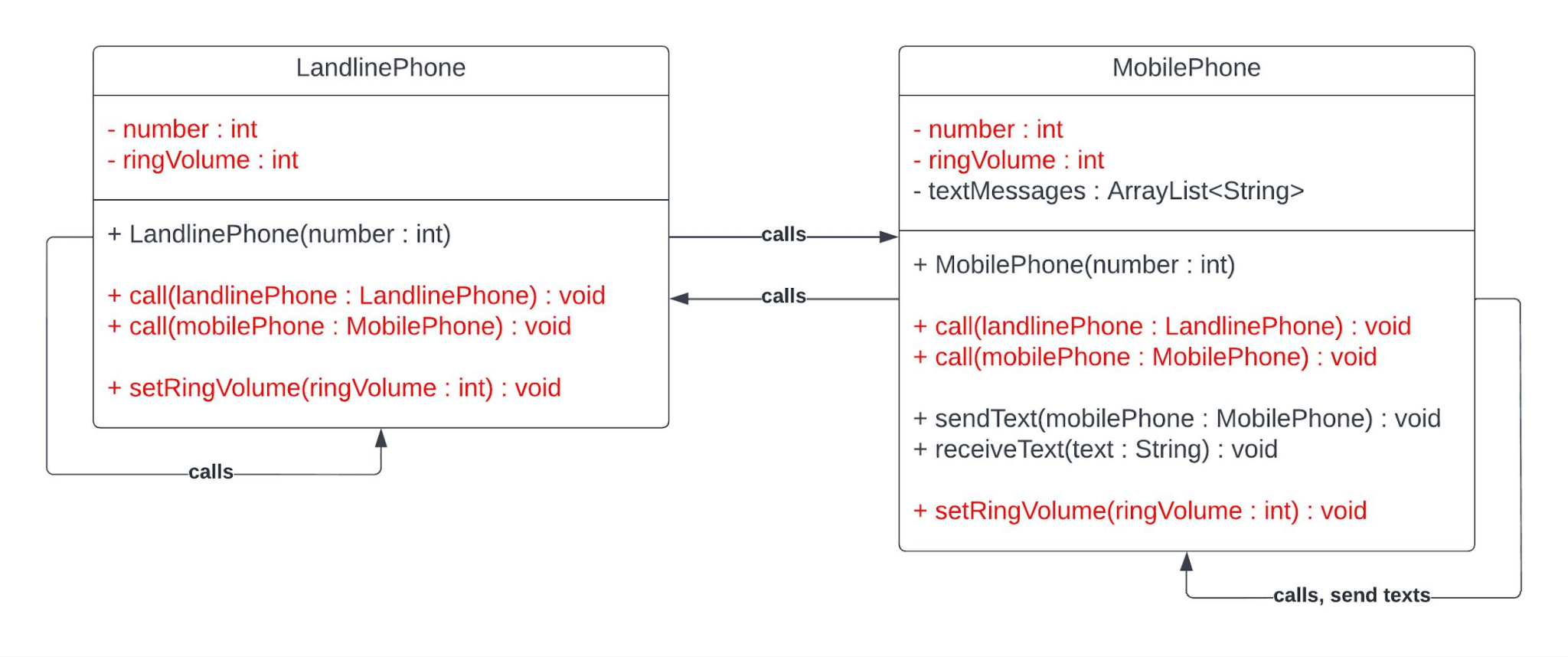
Practice Exercise 9 – Phones

If modeled without inheritance…



Practice Exercise 9 – Phones

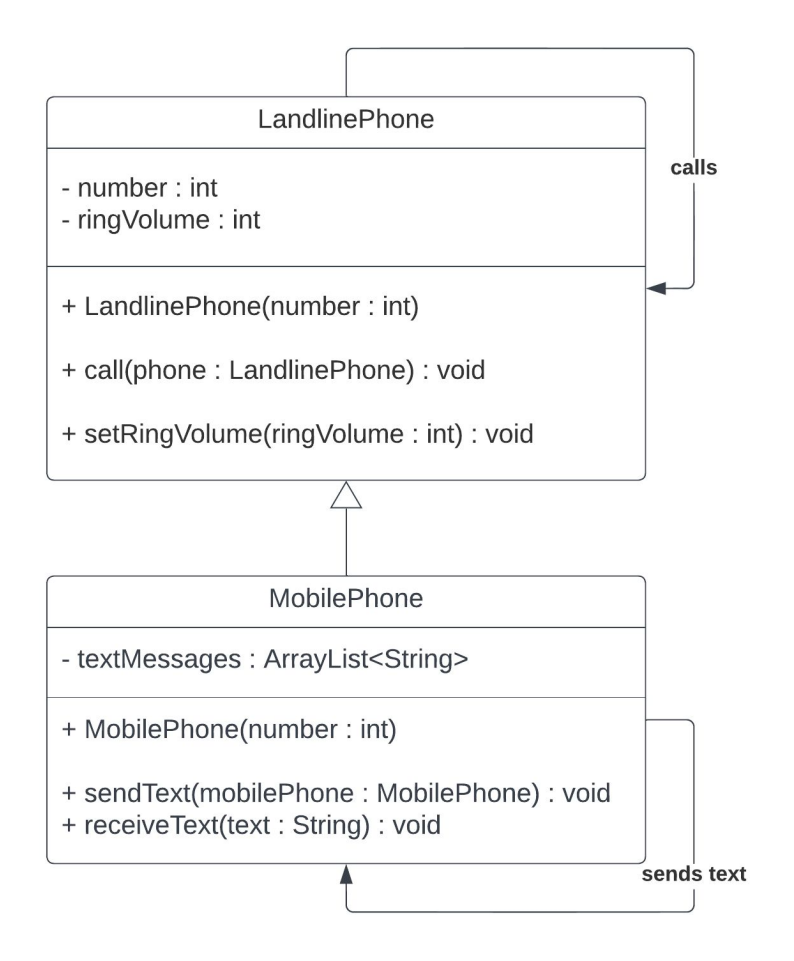
If applying inheritance, look for commonalities…



Practice Exercise 9 – Phones

One option would be to centralize logic in LandlinePhone

This way, LandlinePhone still serves a purpose and MobilePhone extends the capabilities of LandlinePhone



However, all logic inherited from LandlinePhone is dependent on the idea in which the logic inherited will stay the same. Any changes to LandlinePhone will affect MobilePhone.

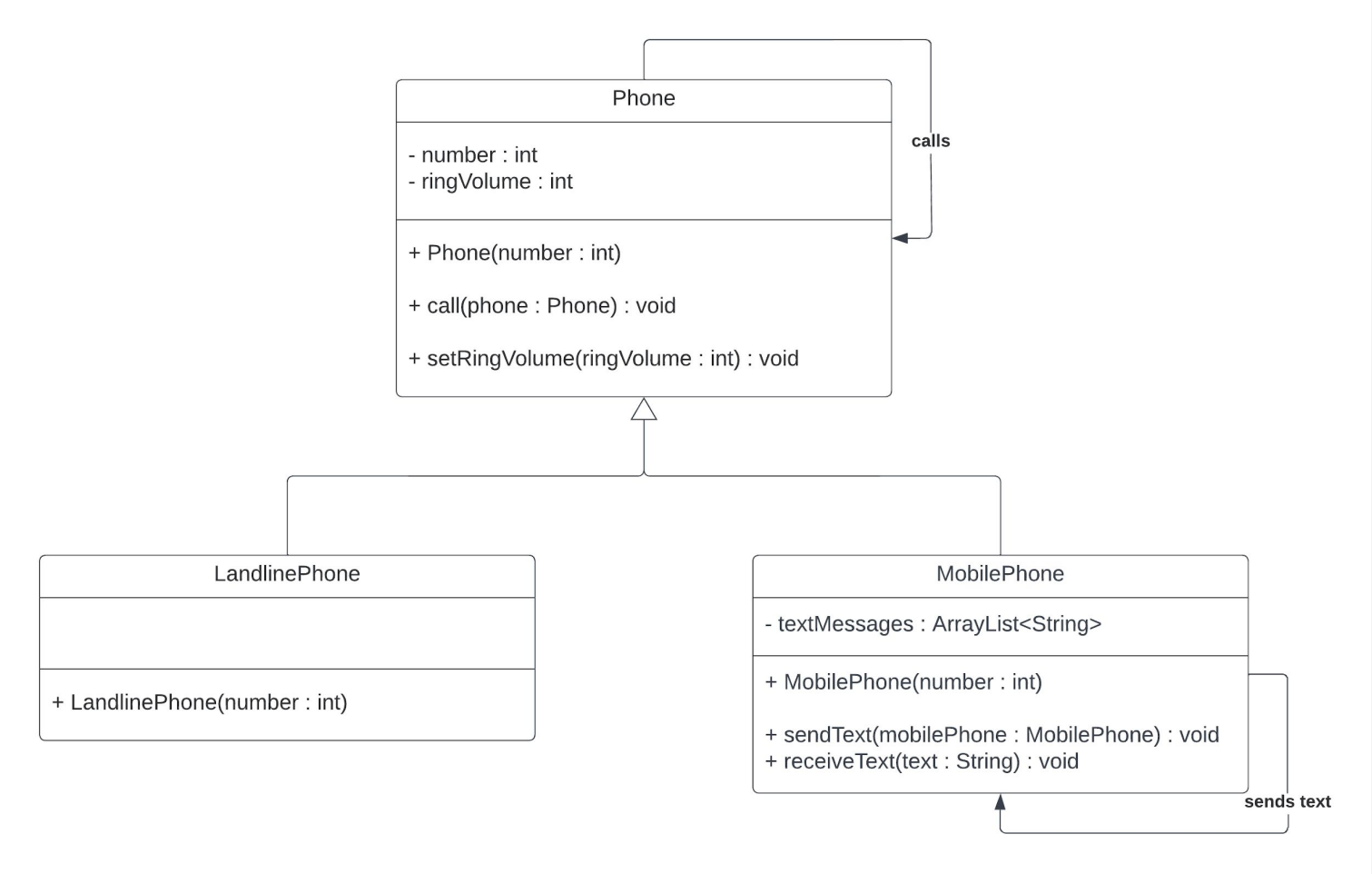
Practice Exercise 9 – Phones

Another option would be to elevate the commonalities to a separate class

This way, MobilePhone is less dependent on LandlinePhone. Any further changes to the subclasses won't affect each other while still maintaining the ability to modify logic from the base Phone class

However, we have more code to maintain…

At least it aids in organization and scalability…

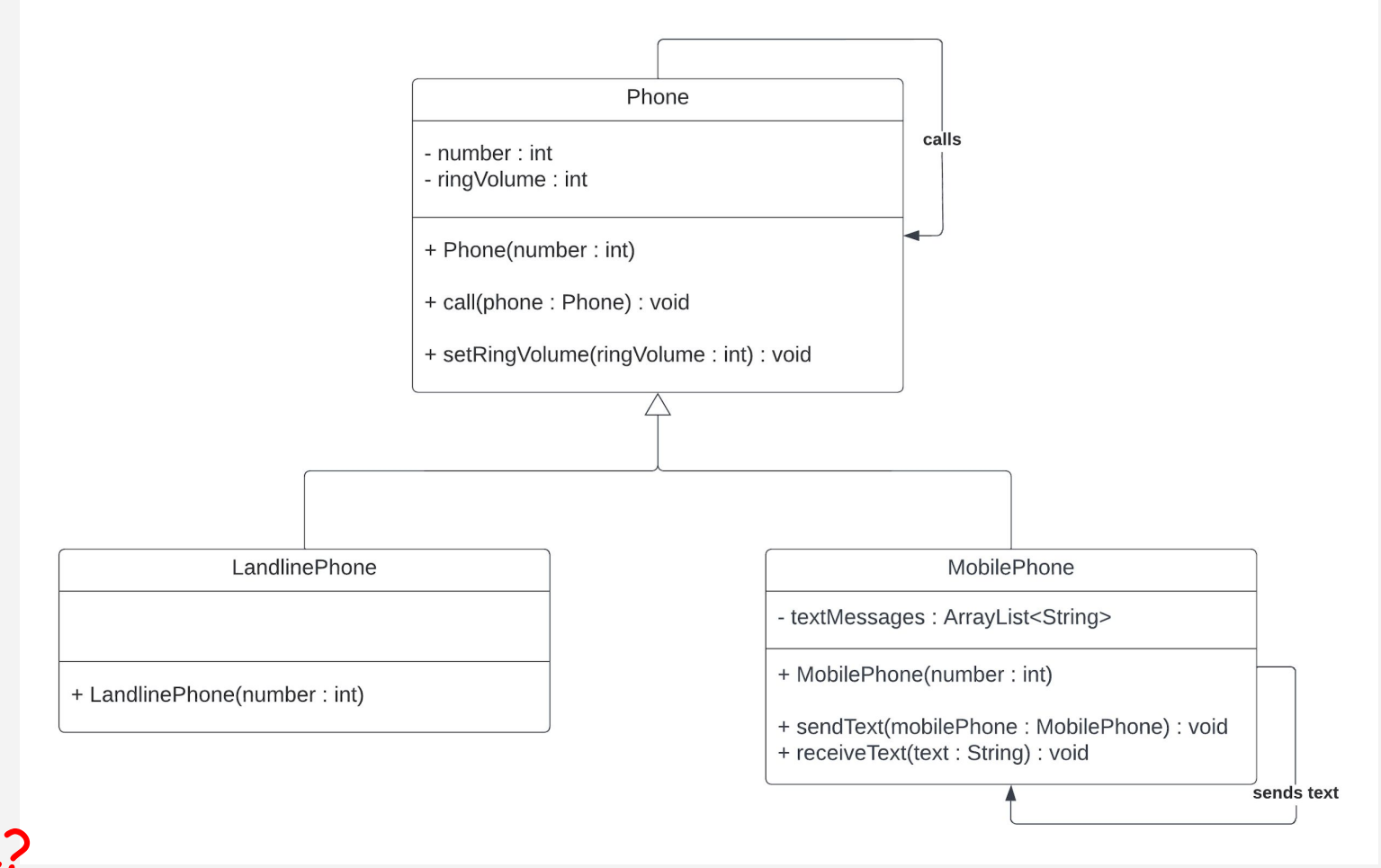


Any questions?

Think…

- The Phone Class does have a purpose…
  - As a superclass
- …but do we imagine ever instantiating a Phone object?

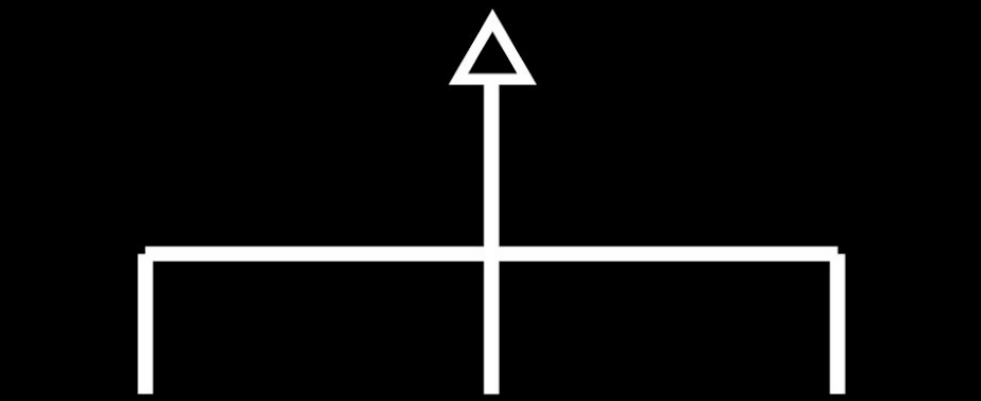
No! What would that even be?



Why do we need to keep the method here if it doesn't make any sense?

Animal

Can make a sound



Cow

Moo!

Chicken

Kroo!

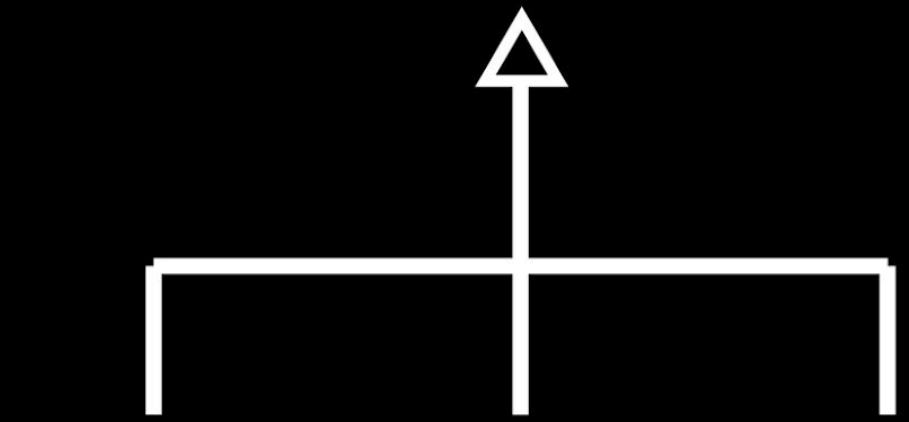
Goat

Meeh!

There is no sense instantiating an animal object.

Animal

Can make a sound



Cow

Moo!

Chicken

Kroo!

Goat

Meeh!

The Four OOP Principles

ENCAPSULATION

Objects must be the only entity responsible for maintaining their own states

ABSTRACTION

Only relevant data and behavior are exposed at any given time, the rest are abstracted away

What we'll talk about today!

INHERITANCE

Objects may acquire some, if not all, the properties and behaviors of another object, typically their parent object

POLYMORPHISM

Objects may have multiple types; apart from their object type, they also have the object type of their parent objects

Slide c/o Ms. Candy Espulgar

Abstraction

- Abstraction is the process of hiding certain details and showing only essential information to the user
- In Java, abstraction can be achieved with either abstract classes, abstract methods, or interfaces

Slide c/o Ms. Candy Espulgar

Abstract Classes

- Provide common functionality across a set of related classes, while allowing default method implementations
  - They can’t be instantiated
    - They need to be extended to be used (i.e. only serve as a parent)
    - Once abstract classes are extended, all their methods need to be implemented
  - They may have abstract and concrete (non-abstract) methods
    - As opposed to non-abstract classes, which can only have concrete methods

Slide c/o Ms. Candy Espulgar

Abstract Classes

- When do we use Abstract Classes?
  - When you want to provide properties and methods across related classes, but instantiating the parent class makes no sense
  - When you want all related classes to contain a specific implementation of some methods, but also want the implementation of other methods to be determined by specific child classes

Abstract Methods

- Do not have a method body
- The declaration of abstract methods are done in an abstract class, but their implementation is defined in a child class
- The first non-abstract class that extends an abstract class with abstract methods must define the implementation of all undefined abstract methods

In code…

Notice how there is no implementation here!

```
public abstract class Animal {
    public Animal() {
        //constructor
    }

    abstract public void makeSound();

    public void doSomething() {
        // something
    }
}
```

```
public class Dog extends Animal {

}
```

Will compiling a blank class work?

Nope!
Needs to implement all abstract methods

And don't worry, you can interchange abstract and the access modifier ☺

In code…

```
public abstract class Animal {
    public Animal() {
        //constructor
    }

    abstract public void makeSound();

    public void doSomething() {
        // something
    }
}
```

```
public class Dog extends Animal {
    public void makeSound() {
        // Bark
    }
}
```

Will this compile now?

Yup!
You can also call doSomething()
or even override it

Abstract Class + Methods

- Helps “force” classes to have certain methods
  - Applies constraints that allow subclasses to conform to the parent’s design
- Once a method is an abstract, the whole class should be an abstract

In code…

```
public abstract class Animal {
    public Animal() {
        //constructor stuff
    }

    abstract public void makeSound();

    public void doSomething() {
        // something
    }
}
```

```
// driver class' main

Animal animal = new Animal();
```

Will this compile work?

Nope!
Remember, it cannot be instantiated

In code…

```
public abstract class Animal {
    public Animal() {
        //constructor stuff
    }

    abstract public void makeSound();

    public void doSomething() {
        // something
    }
}
```

```
// driver class' main

Animal animal = new Dog();
```

How about this?

Yup!
Dog is instantiated, not Animal

In code…

```
public abstract class Animal {
    public Animal() {
        //constructor stuff
    }

    abstract public void makeSound();

    public void doSomething() {
        // something
    }
}
```

```
// driver class' main


ArrayList<Animal> animal;
animal = new ArrayList<Animal>();
```

How about this?

Yup!
Here, an arrayList of an abstract class was instantiated

Questions?

(whether abstract or regular)

Can I extend multiple classes?

Nope. Recall our discussion on multiple inheritance. ☺️

[Recall] Disclaimer: Multiple Inheritance

- In Java, a class can only have one direct superclass
- Multiple inheritance the concept of inheriting members from multiple superclasses
  - There are issues to this – such as tracing which members belong to which class and how memory is managed
  - There are ways to achieve this in Java, such as using Interfaces – which we'll discuss in today's session
- However, other OO languages, like C++, have some kind of support for multiple inheritance

Interfaces

- Very similar to classes, but only contain method declarations with no implementation
  - i.e. total abstraction
- Sole purpose is to ensure that any class that implements an interface is to have specific methods
  - Think of it as an abstract class with only abstract methods

Interfaces

Should these objects have a super class?

Maybe? Maybe not share direct parents?

What might they have in common?

They can all die 💀 ...

Interfaces

Since interfaces normally force method implementation, they're known to force behavior, thus giving a class an "ability" or a X-able

```
public interface Killable {
    public void die();
}
```

Must implement this method

```
public class Monster implements Killable {
    public void die() {
        //Game mechanics
    }
}
```

Interfaces

Wait! If the method is like an abstract, should it also declare the word "abstract" like this?

```
public interface Killable {
    public void die();
    //public abstract void die();
}
```

```
public class Monster implements Killable {
    public void die() {
        //Game mechanics
    }
}
```

No need to. All methods in an interface are abstracts. But there is no harm in declaring it too. ☺️

Why Interfaces?

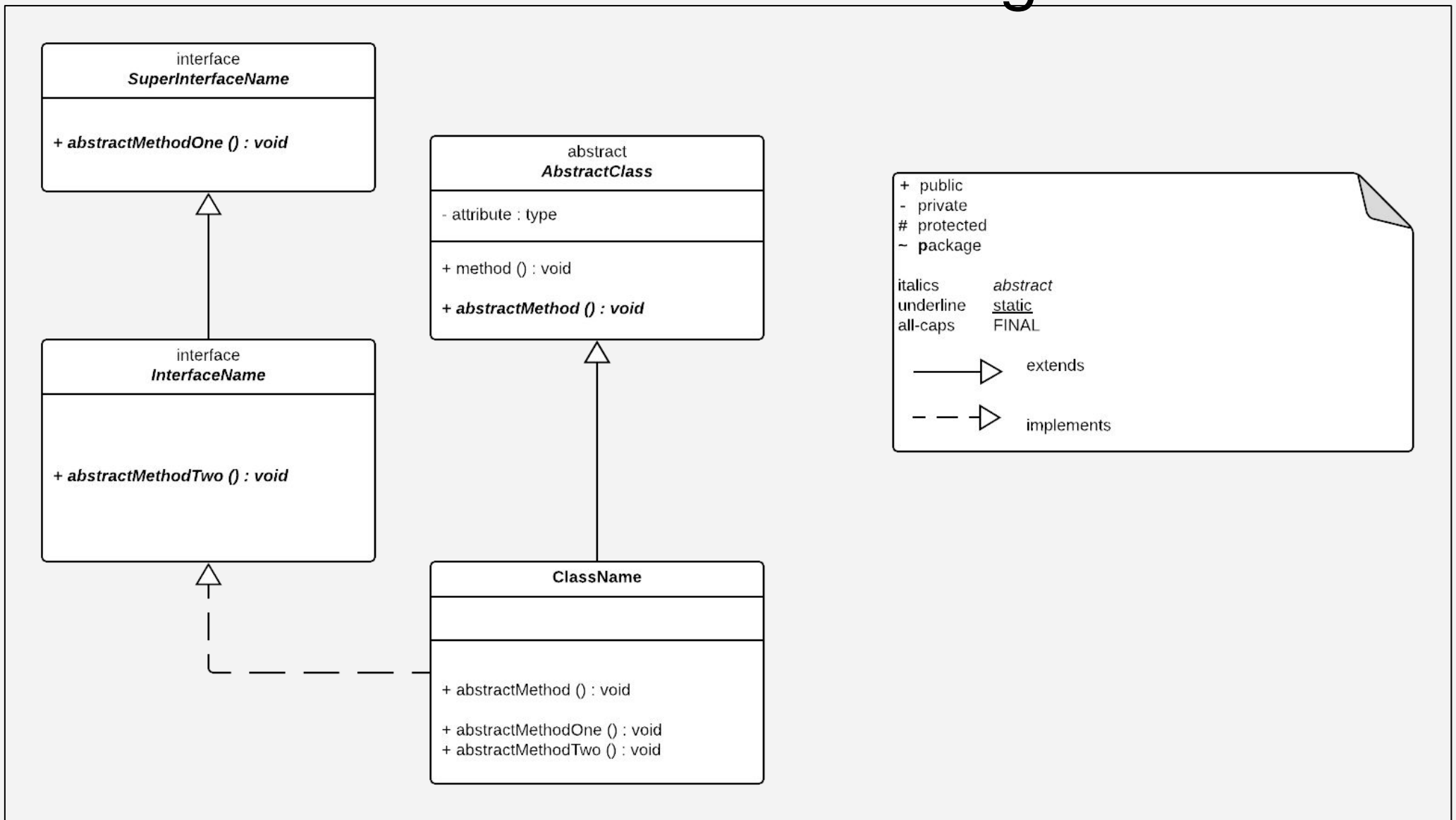
- Have been known to be more useful than simple inheritance
  - Can implement multiple interfaces
- Better maintains the integrity of a class' attributes and methods
- Ensures common behavior between two otherwise unrelated classes

(whether abstract or non-abstract)

Can classes implement interfaces?

Yup! 😊

We update what we know about the different symbols found in UML Class Diagrams



Questions? ☺️

Next meeting…

- Practice Exercise 10(Code) & 11(UML) ***discussion***
- Graded Exercise 7 (Implementation)

Keep learning…