CSCI 132:
Basic Data Structures and Algorithms

OOP Conclusion, UML

Reese Pearsall

Spring 2023

Announcements

Program 1 due **SUNDAY** at 11:59 PM

- If you worked with a partner, make sure you clearly indicate that in your submission

Lab 4 due **TUESDAY** at 11:59 PM

- Interfaces. We will talk more about it on Monday

Inheritance is a mechanism in Java that allows for a class to acquire instance fields and methods from another class

```
public class Programmer extends Employee {
}
```

Inheritance is great when you have **shared** attributes and methods across different classes

Inheritance is a mechanism in Java that allows for a class to acquire instance fields and methods from another class

```
public class Programmer extends Employee {
}
```

Inheritance is great when you have **shared** attributes and methods across different classes

```
public Abstract class Employee {
}
```

We can inherit from **Abstract** classes, but we can't *create instances* of **Abstract** classes (can't use new keyword)

Interfaces are abstract classes that only contain methods with no body

```
public class Ferrari implements Vehicle {
}
```

Interfaces are great when you need **shared functionality** with **different implementations**

When a class implements an interface, that class MUST define and write the bodies of the interface methods

```
public interface Vehicle {
        void accelerate(int a);
        void slowdown(int a);
        void refuel(int a);
}
```

Inheriting from a class	Implementing an Interface
Class inherits **instance fields** and **methods**	Class inherits methods with no bodies
Can only inherit from one class	Can implement multiple interfaces
Sub class is **not required** to override methods	Sub class is **required** to override methods

Polymorphism is the ability of a class to provide different implementations of a method, depending on the *type of object* that is passed to the method.

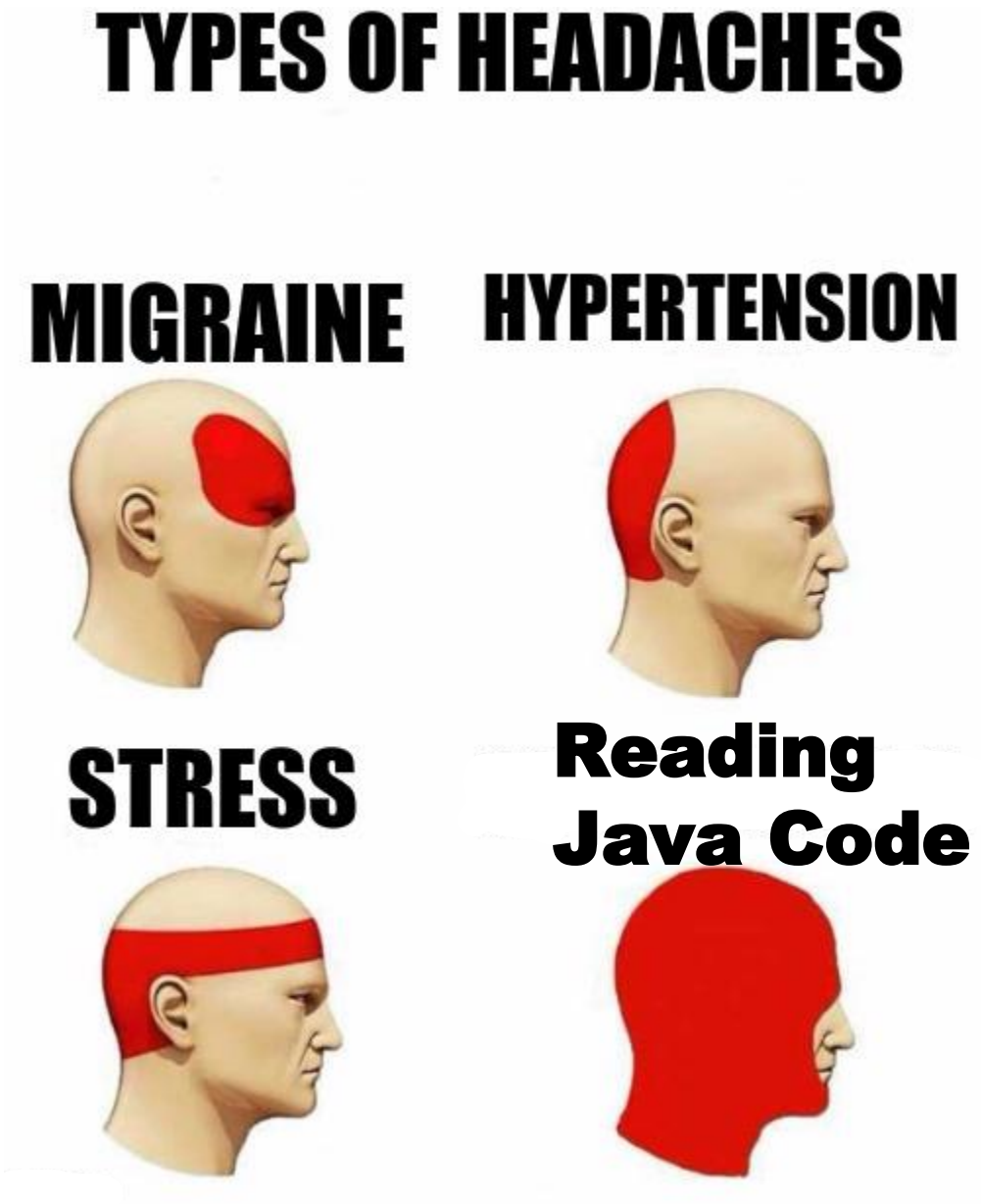
```
Bird a2 = new Bird("Puffin",27.0, "North America",7400000,21.5);
Wolf b2 = new Wolf("Arctic Wolf",120.0, "North America",200000, 16);

a2.makeSound();
b2.makeSound();
```

The `makeSound()` method does something different for each object

We could have many classes with many kinds of relationships

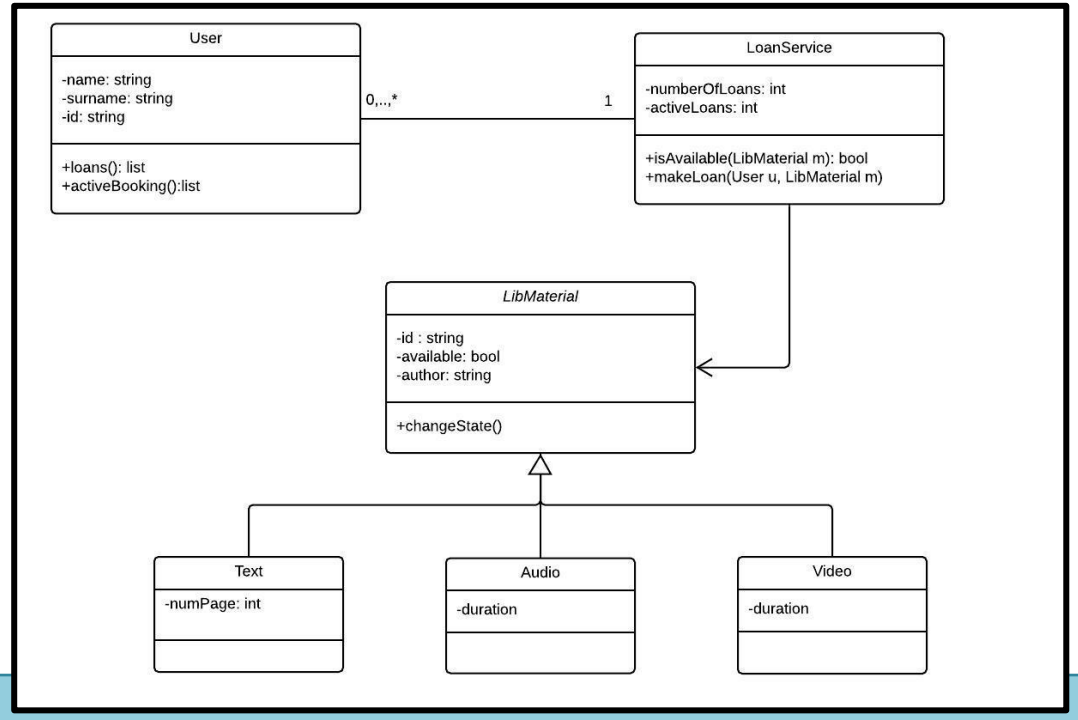
- Many levels of inheritance
- Multiple interfaces
- Some abstract classes, some not
- Method overloading



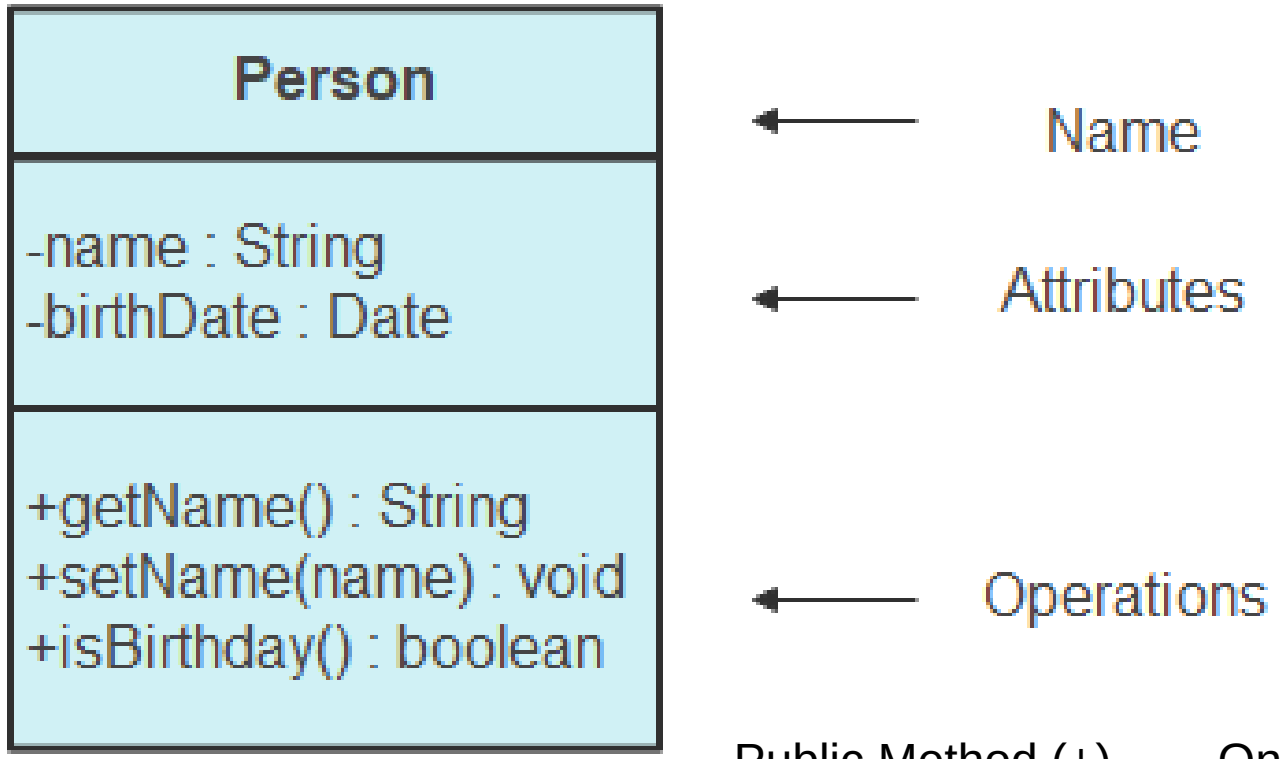
It would be nice to have a way to visualize the architecture of Java classes without needing to dive into complex source code

UML (Unified Modeling Language) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems.

We can use a **UML Class Diagram** to visualize the architecture of our Java classes



We can use a **UML Class Diagram** to visualize the architecture of our Java classes

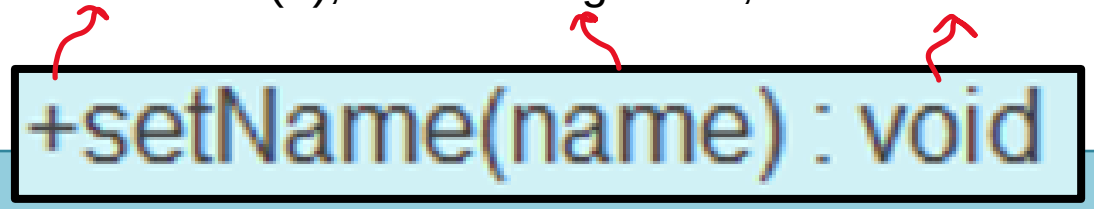


+ = public
- = private
 # = protected

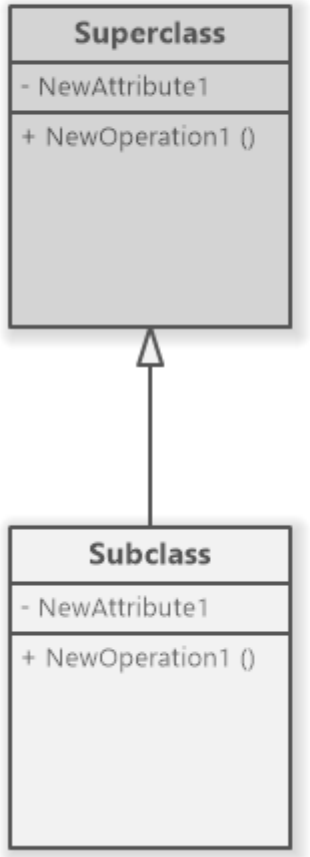
Each Java class is a box.

Each box has the
- name of the class
- Attributes
- operations/methods of the class

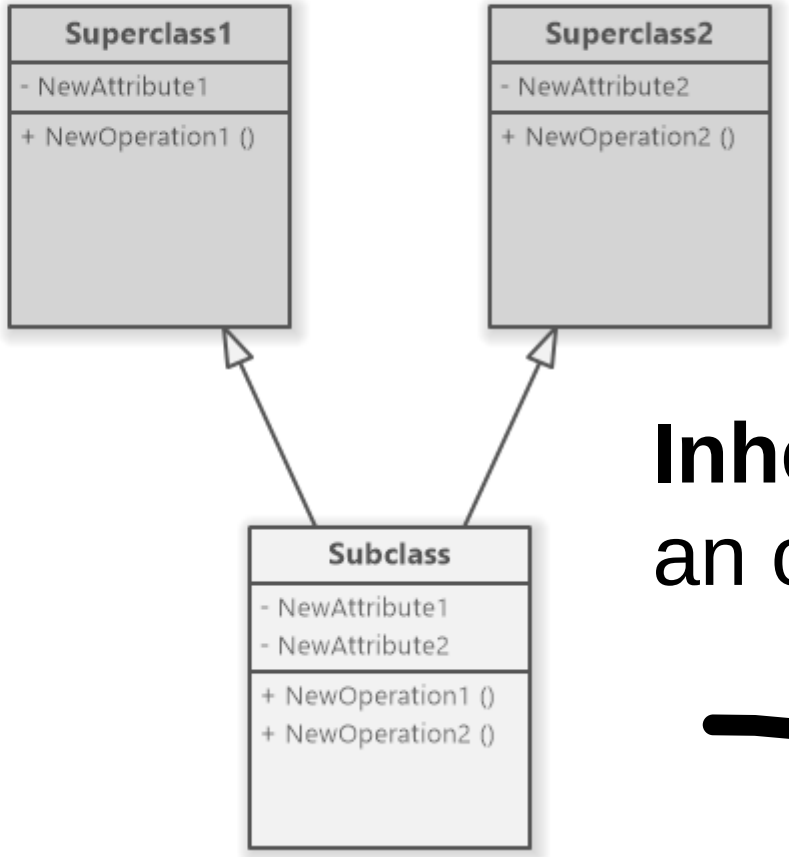
Public Method (+), One argument, Returns nothing



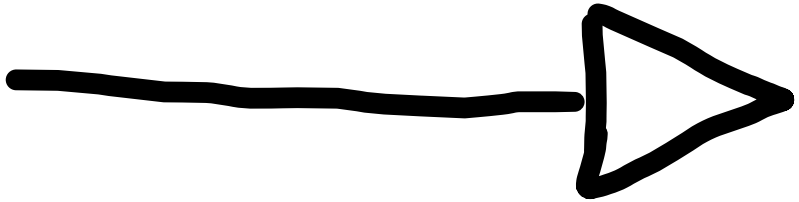
Single Inheritance

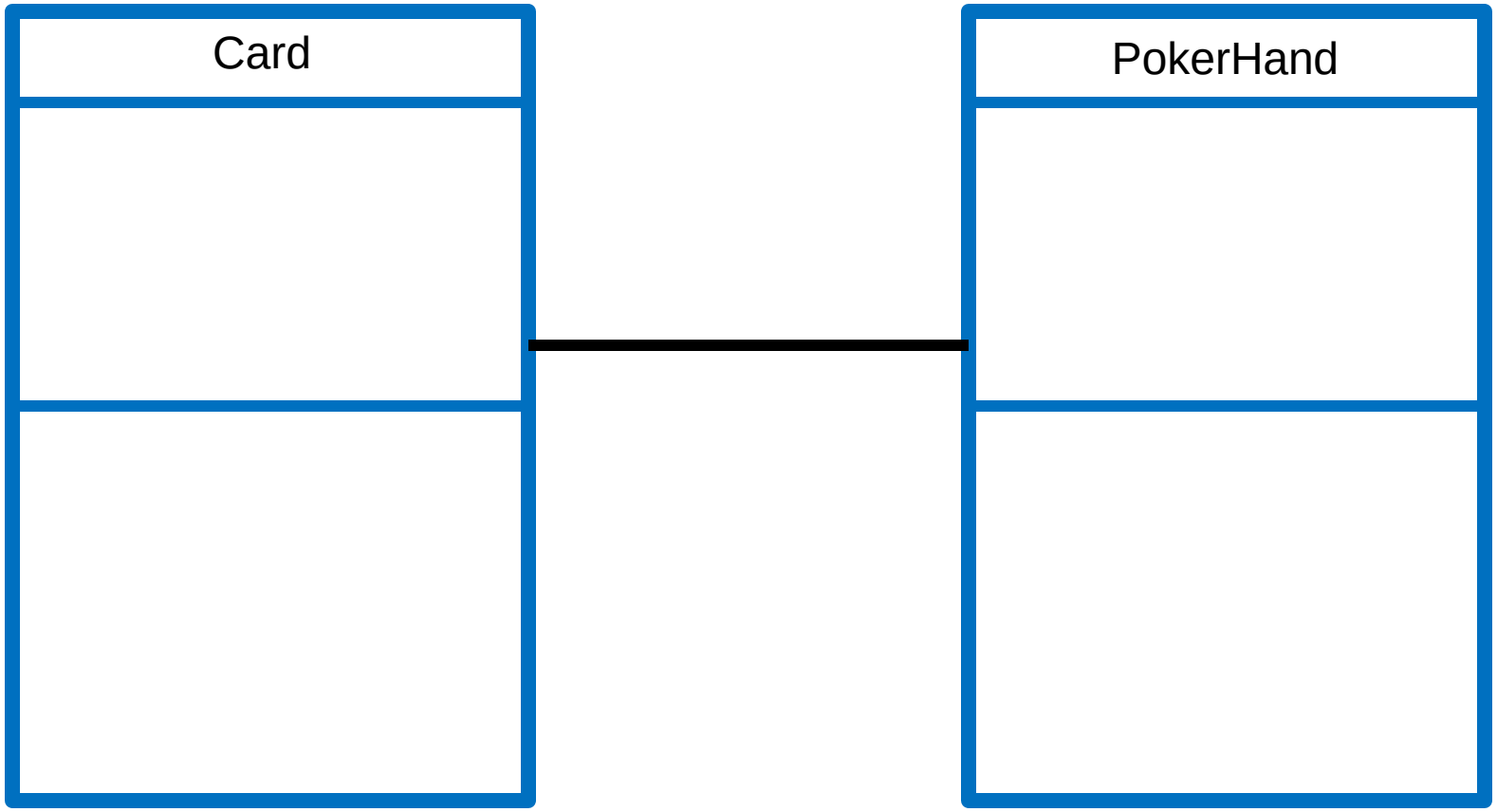


Multiple Inheritance



Inheritance is indicated by an open arrow

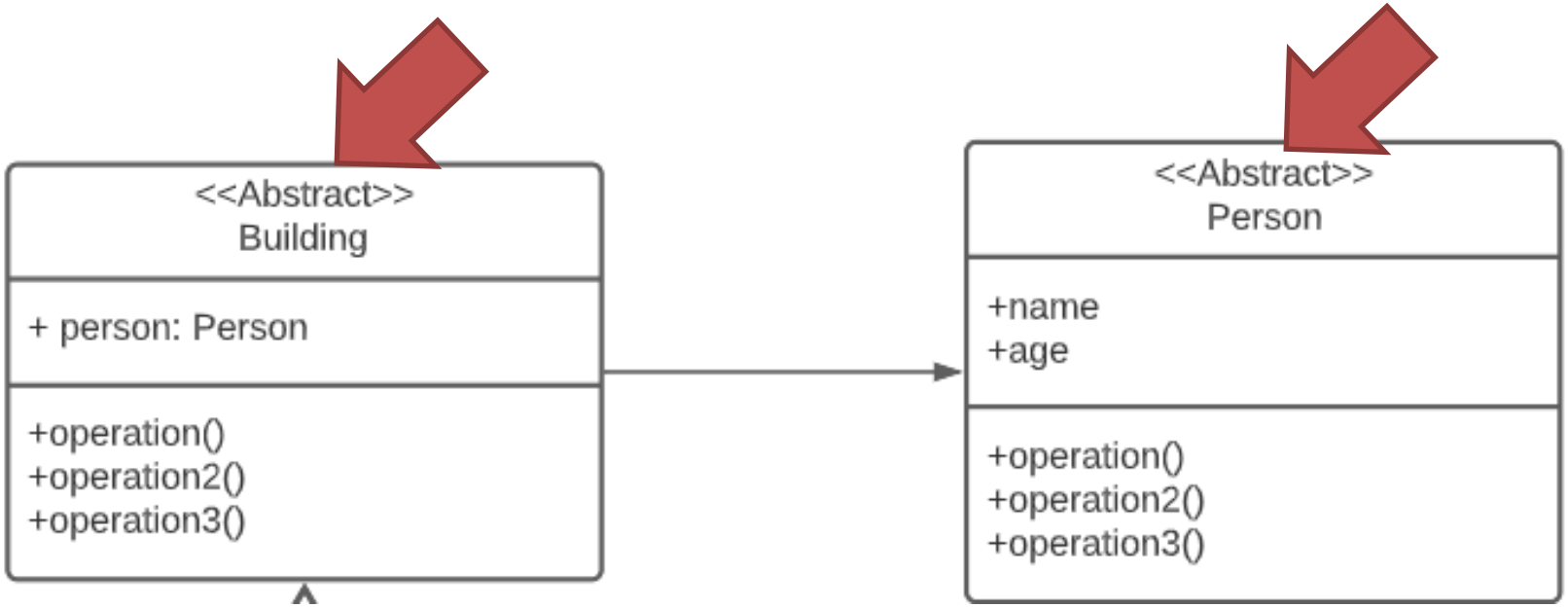




Indicated by a normal line

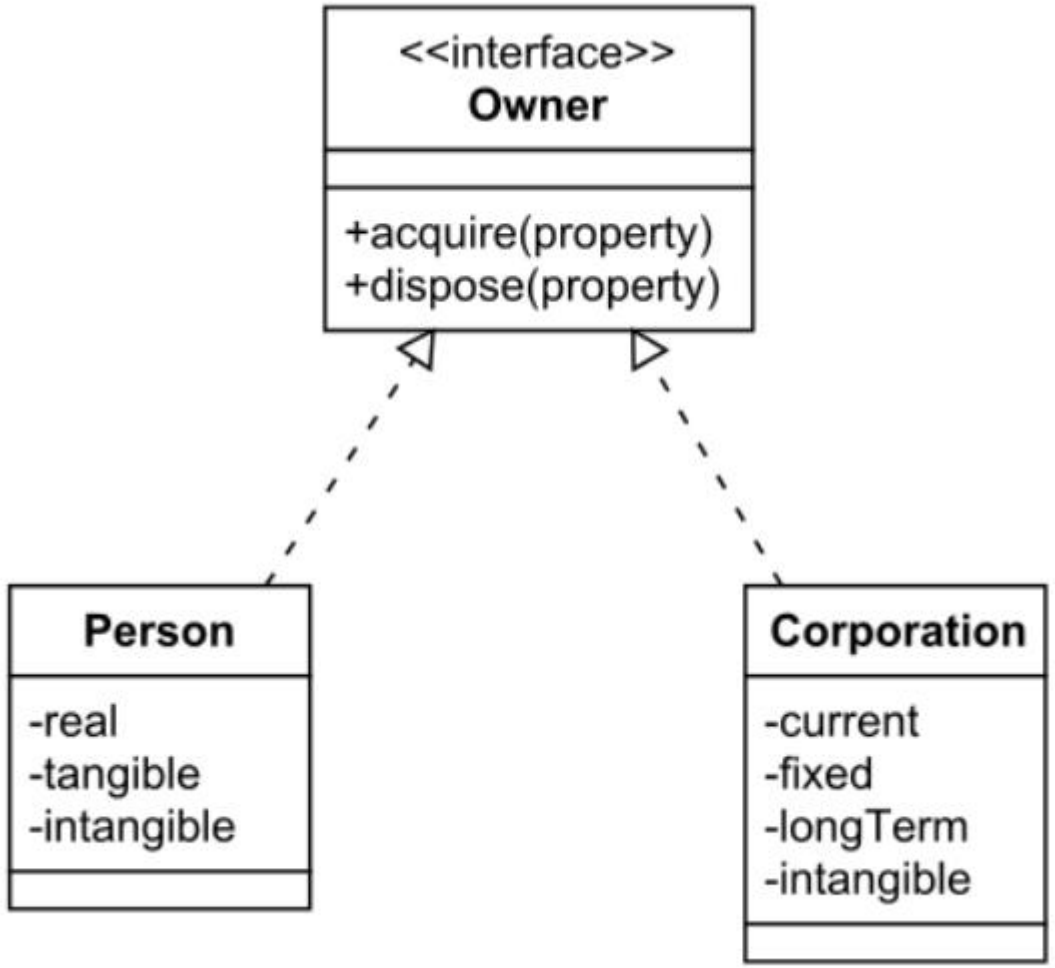
Or by a normal arrow

An **association** is created when a class is referenced from another



Abstract classes are indicated by

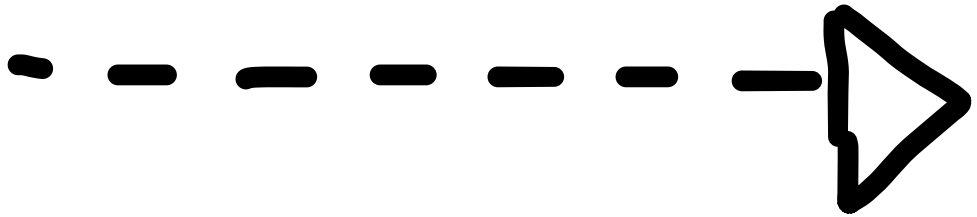
<< Abstract >>



Interfaces are indicated by

<<interface>>

Implementing an interface is a **realization** relationship



Basic UML Example