CSCI 132:
Basic Data Structures and Algorithms

OOP Conclusion, UML, Java file I/O

Reese Pearsall

Fall 2023

Announcements

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

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

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.

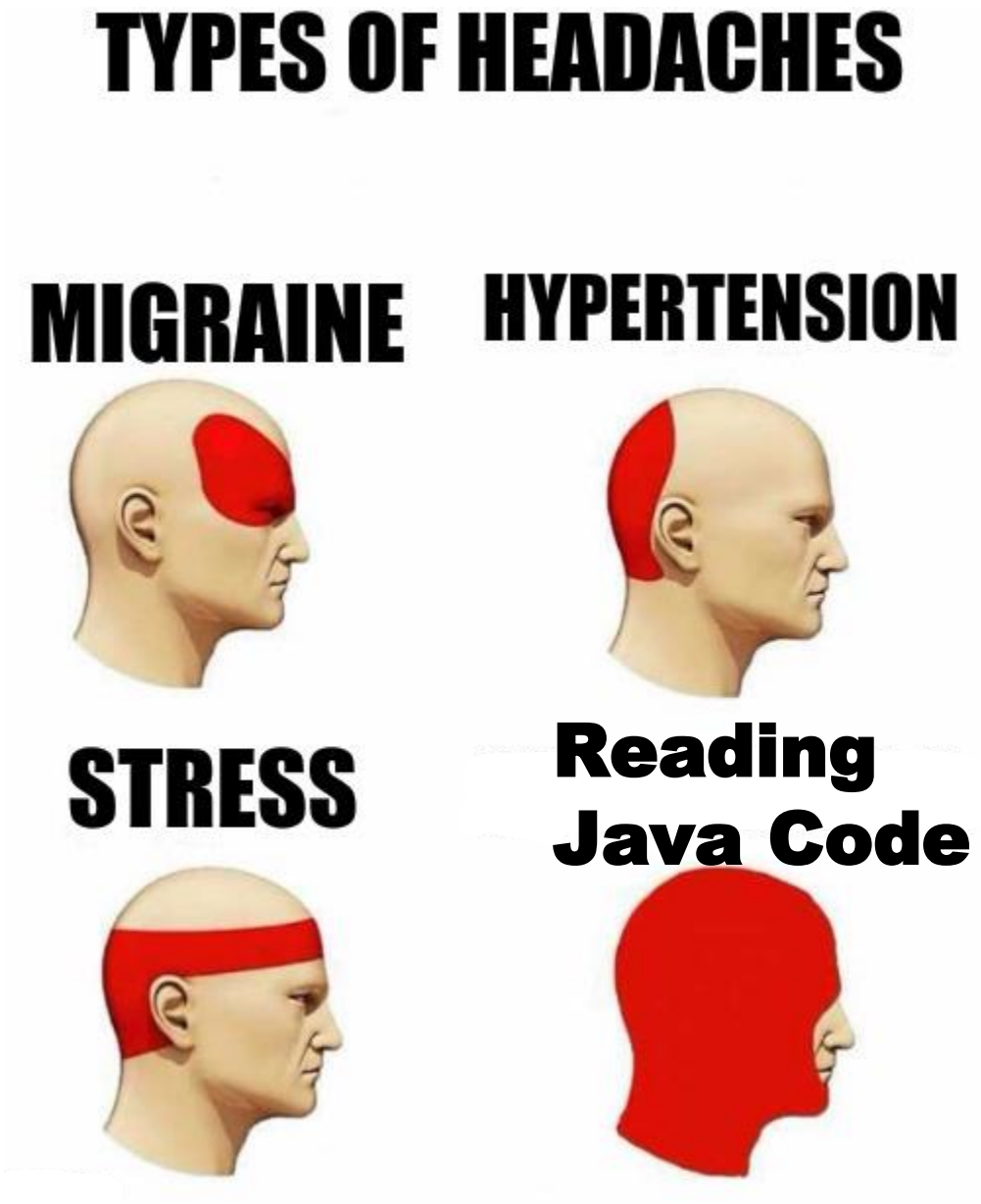
```
Bird a2 = new Bird("Puffin",27.0, "North America",7400000,21.5);
Wolf b2 = new Wolf("Arctic Wolf",120.0, "North America",200000, 16);

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

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

We could have many classes with many kinds of relationships

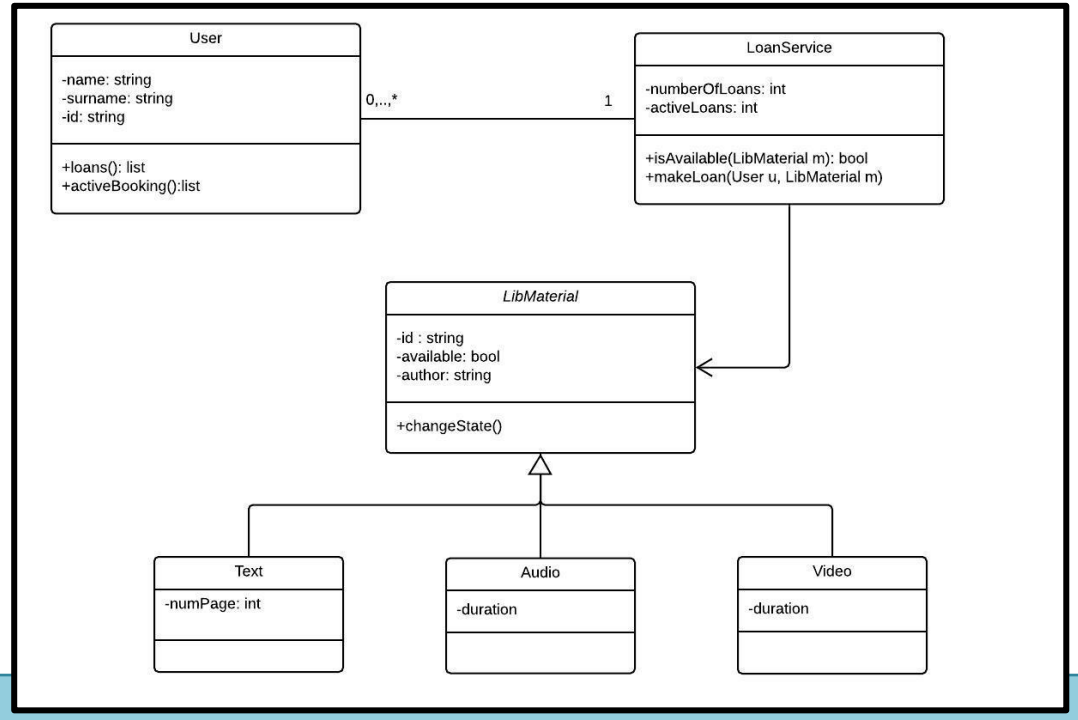
- Many levels of inheritance
- Multiple interfaces
- Some abstract classes, some not
- Method overloading



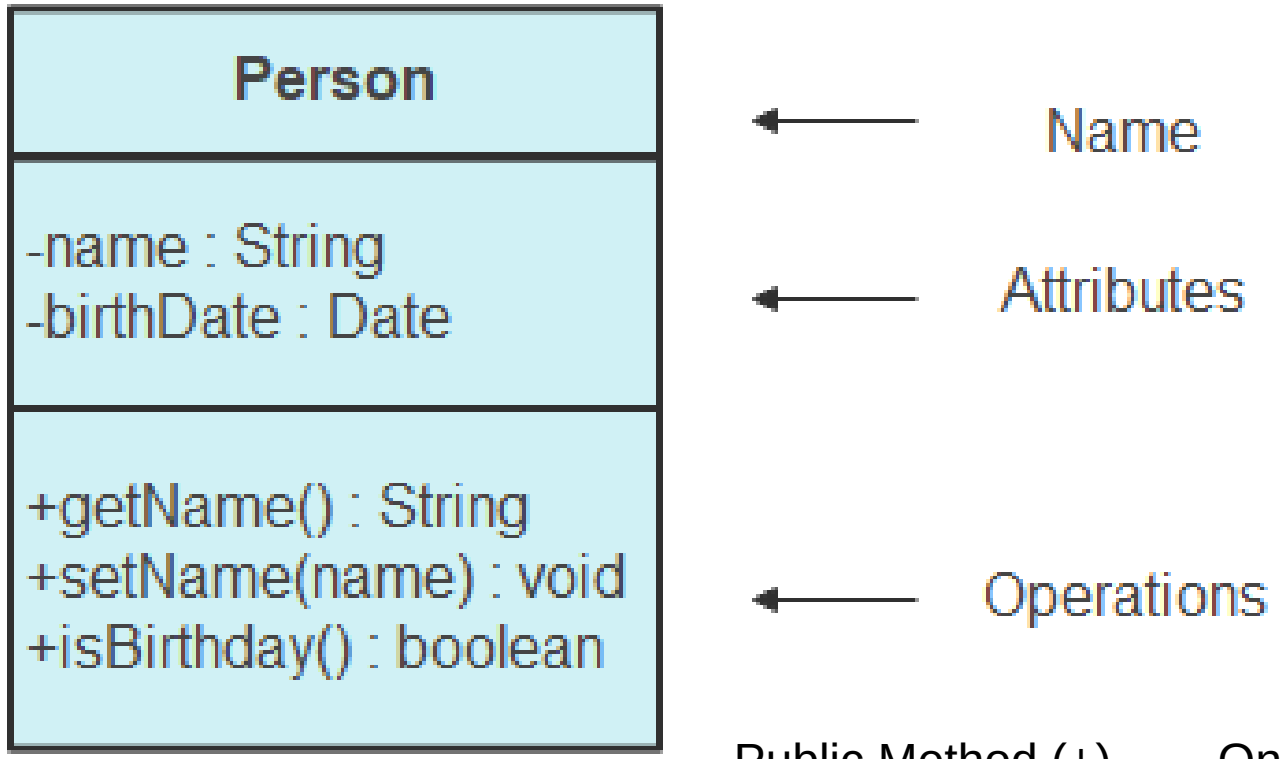
It would be nice to have a way to visualize the architecture of Java classes without needing to dive into complex source code

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

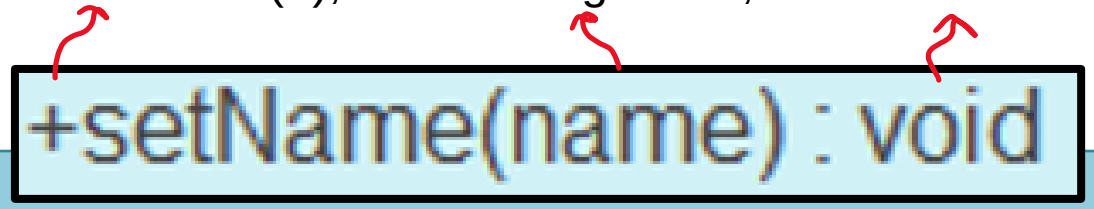


+ = public
- = private
 # = protected

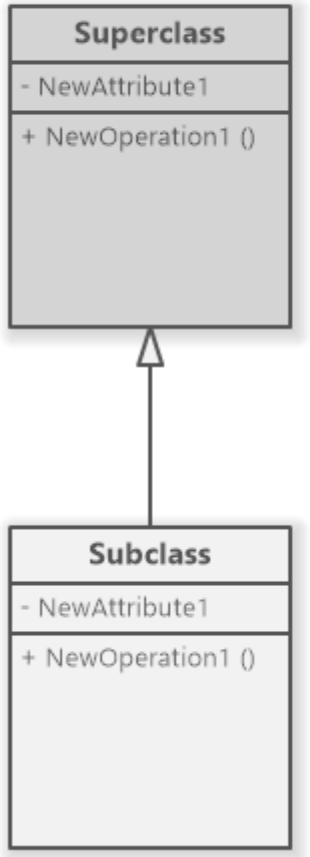
Each Java class is a box.

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

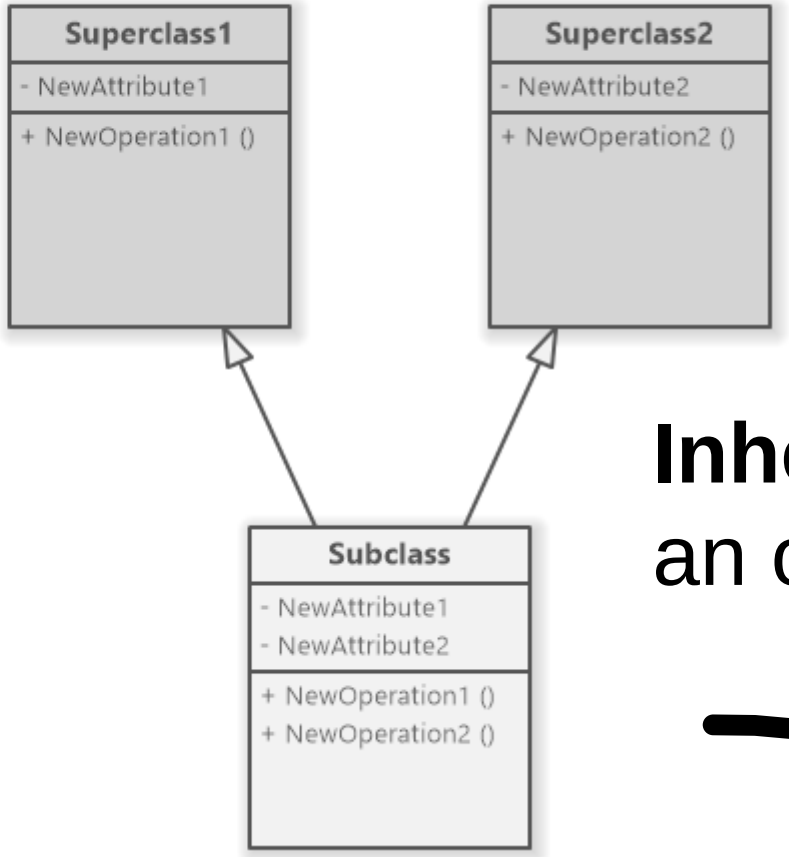
Public Method (+), One argument, Returns nothing



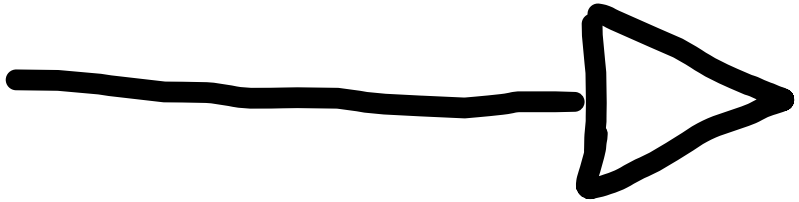
Single Inheritance

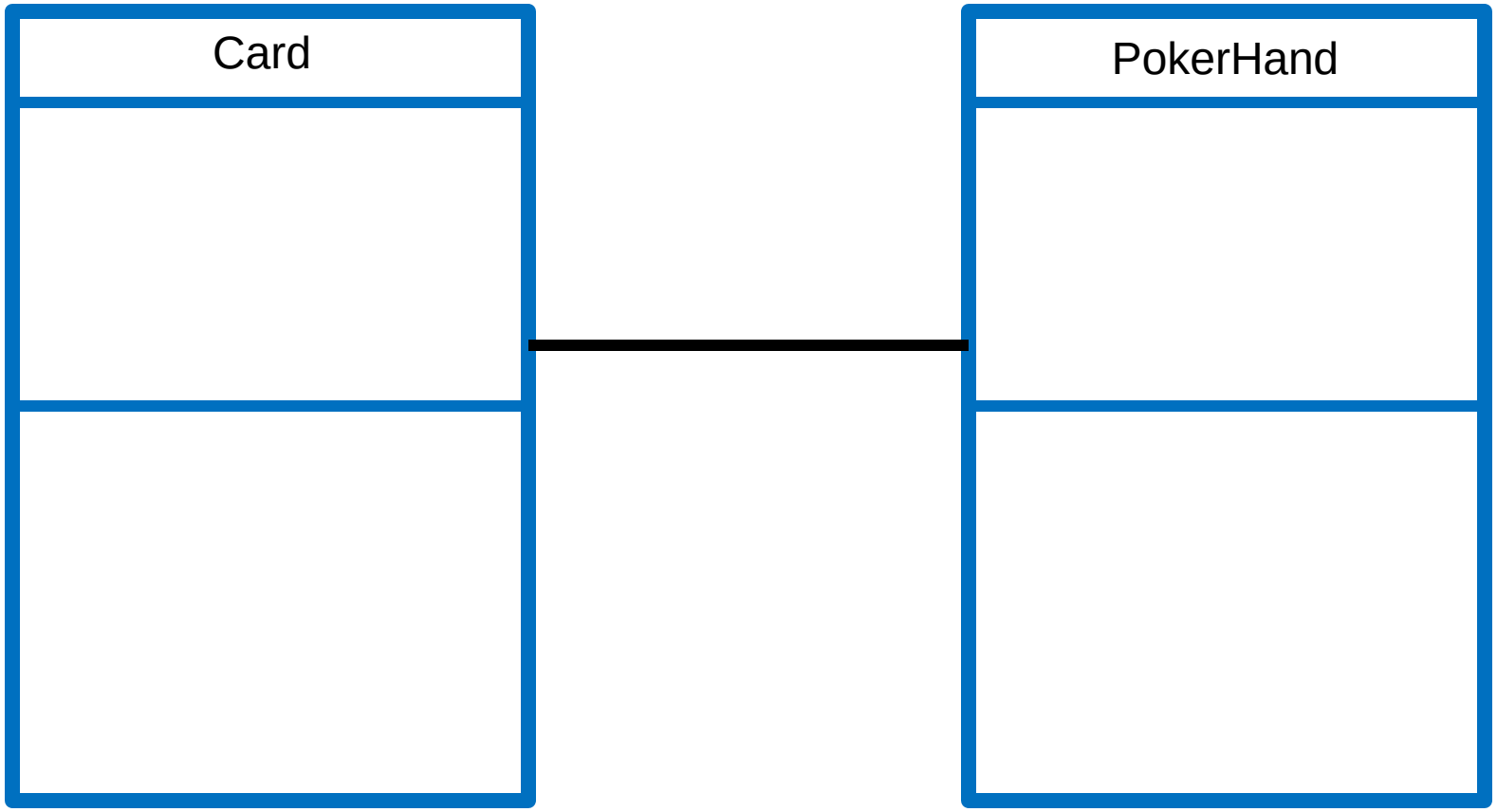


Multiple Inheritance



Inheritance is indicated by an open arrow

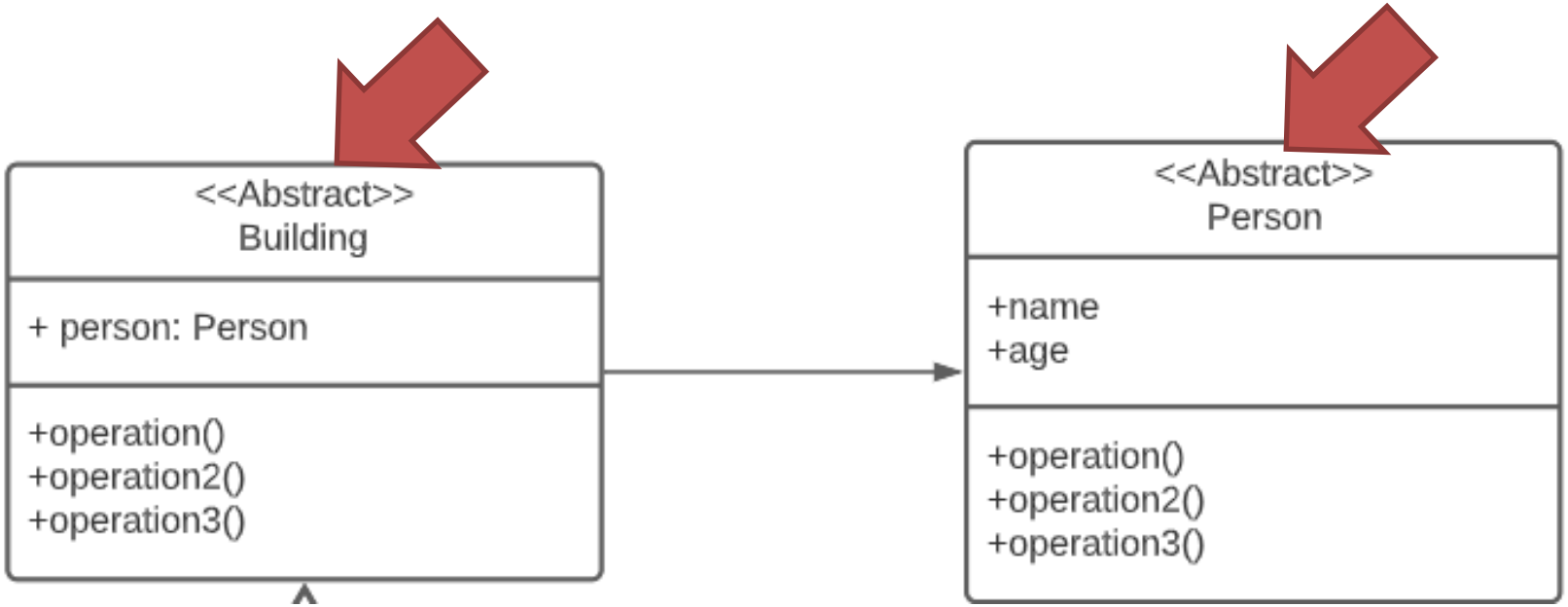




Indicated by a normal line

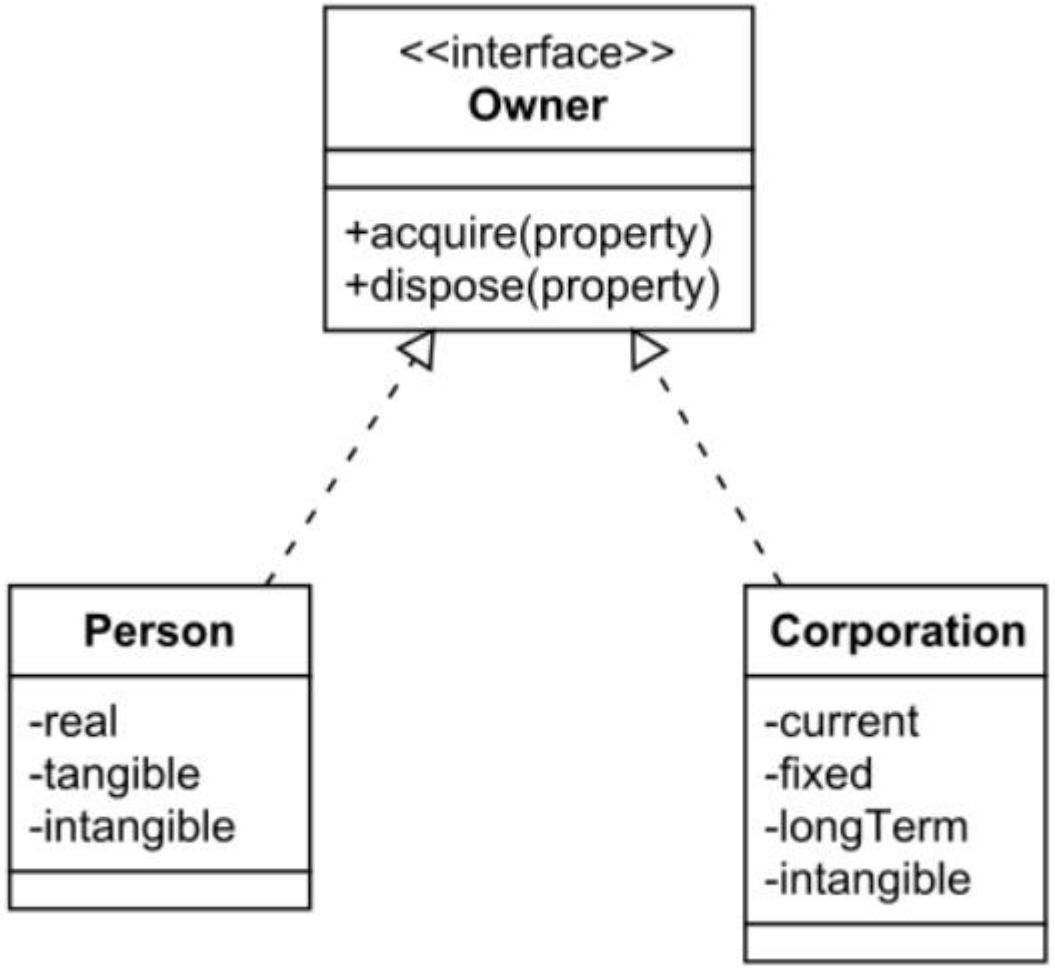
Or by a normal arrow

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



Abstract classes are indicated by

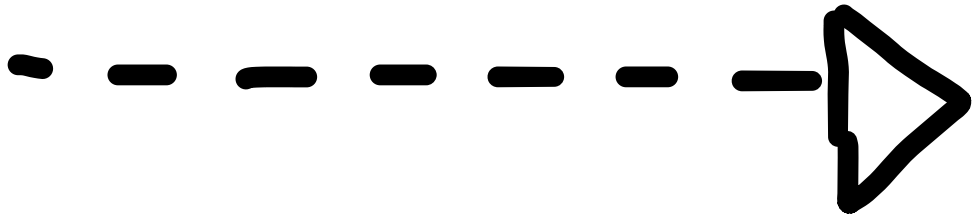
<< Abstract >>



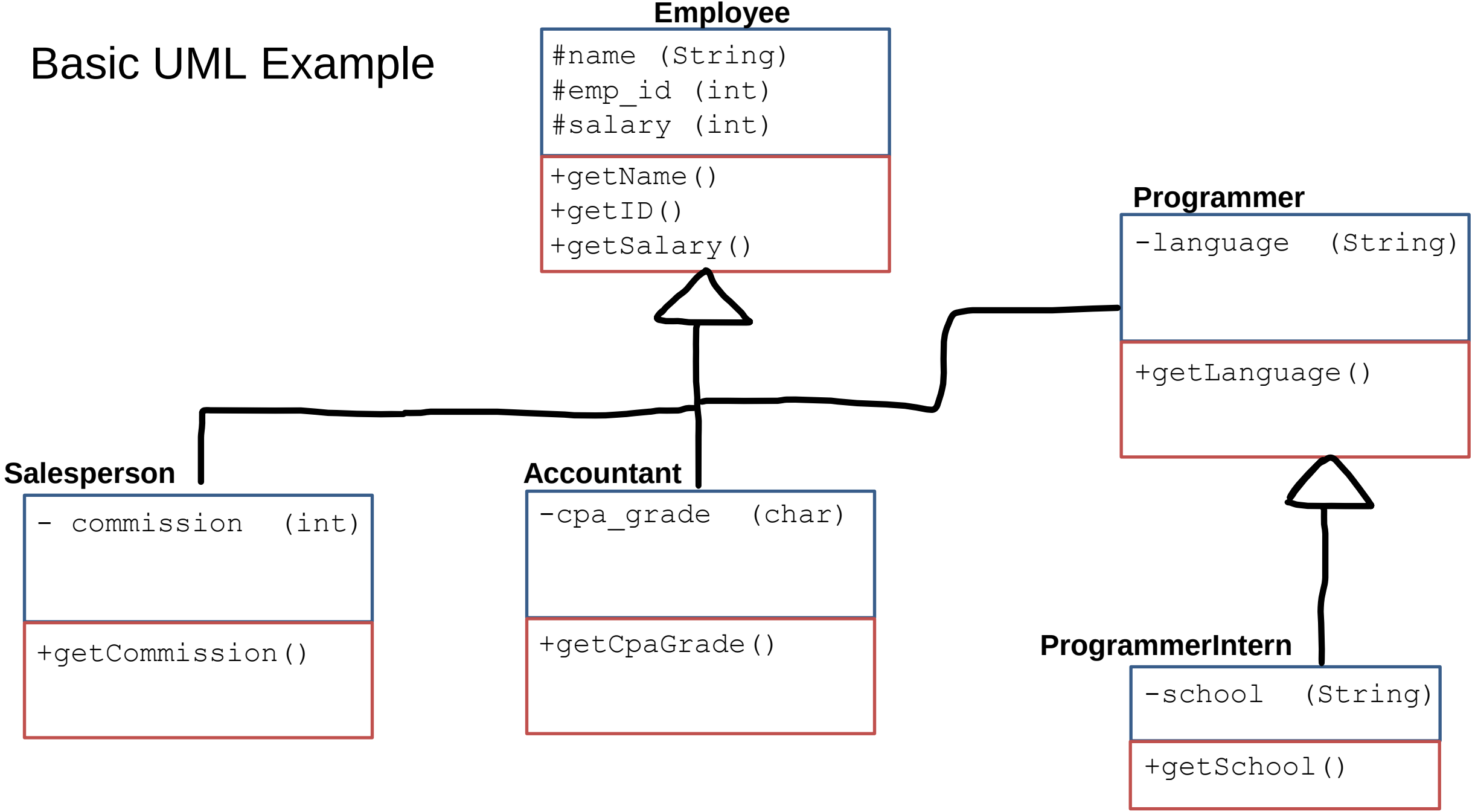
Interfaces are indicated by

<<interface>>

Implementing an interface is a **realization** relationship



Basic UML Example



Reading in a file in Java

```
FileReader file;
try {
    file = new FileReader("video_games.csv");

    Scanner inFile = new Scanner(file);

    inFile.nextLine();

    while(inFile.hasNext()) {
            String line = inFile.nextLine();
            System.out.println(line);

            String[] splitted_line = line.split(",");
            System.out.println(splitted_line[0]);

} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Opening a reading a file must be done inside of a try/catch statement

Reading in a file in Java

Open a file with the `FileReader` library

```
FileReader file;
try {
    file = new FileReader("video_games.csv");

    Scanner inFile = new Scanner(file);

    inFile.nextLine();

    while(inFile.hasNext()) {
            String line = inFile.nextLine();
            System.out.println(line);

            String[] splitted_line = line.split(",");
            System.out.println(splitted_line[0]);

} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Reading in a file in Java

```
FileReader file;
try {
    file = new FileReader("video_games.csv");

    Scanner inFile = new Scanner(file);

    inFile.nextLine();

    while(inFile.hasNext()) {
            String line = inFile.nextLine();
            System.out.println(line);

            String[] splitted_line = line.split(",");
            System.out.println(splitted_line[0]);

} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Open a file with the `FileReader` library

Read contents of file with Scanner

Reading in a file in Java

```
FileReader file;
try {
    file = new FileReader("video_games.csv");

    Scanner inFile = new Scanner(file);

    inFile.nextLine();

    while(inFile.hasNext()) {
            String line = inFile.nextLine();
            System.out.println(line);

            String[] splitted_line = line.split(",");
            System.out.println(splitted_line[0]);

} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Open a file with the `FileReader` library

Read contents of file with Scanner

Read headers of csv file (discard them)

Reading in a file in Java

```
FileReader file;
try {
    file = new FileReader("video_games.csv");

    Scanner inFile = new Scanner(file);

    inFile.nextLine();

    while(inFile.hasNext()) {
            String line = inFile.nextLine();
            System.out.println(line);

            String[] splitted_line = line.split(",");
            System.out.println(splitted_line[0]);

} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Open a file with the `FileReader` library

Read contents of file with Scanner

Read headers of csv file (discard them)

Iterate through each line of file

Reading in a file in Java

```
FileReader file;
try {
    file = new FileReader("video_games.csv");

    Scanner inFile = new Scanner(file);

    inFile.nextLine();

    while(inFile.hasNext()) {
            String line = inFile.nextLine();
            System.out.println(line);

            String[] splitted_line = line.split(",");
            System.out.println(splitted_line[0]);

} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Open a file with the `FileReader` library

Read contents of file with Scanner

Read headers of csv file (discard them)

Iterate through each line of file

Get each line and print it out

Reading in a file in Java

```
FileReader file;
try {
    file = new FileReader("video_games.csv");

    Scanner inFile = new Scanner(file);

    inFile.nextLine();

    while(inFile.hasNext()) {
            String line = inFile.nextLine();
            System.out.println(line);

            String[] splitted_line = line.split(",");
            System.out.println(splitted_line[0]);
```

Open a file with the `FileReader` library

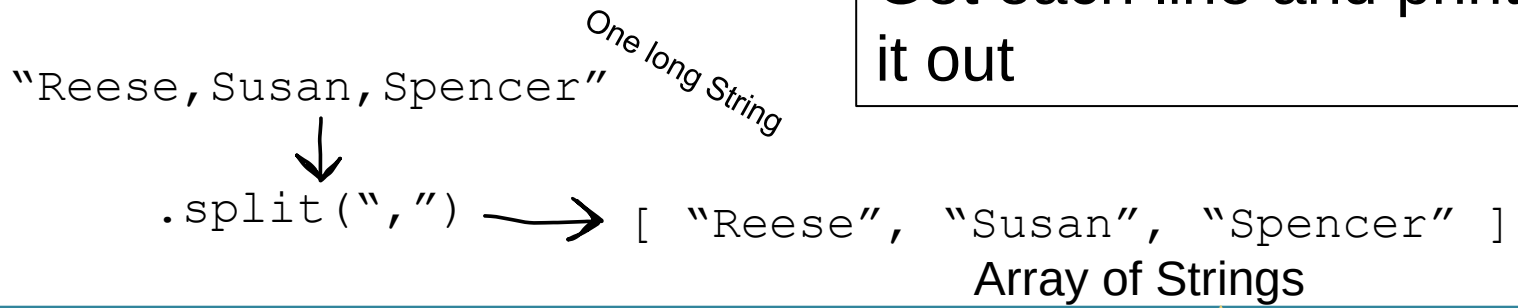
Read contents of file with Scanner

Read headers of csv file (discard them)

Iterate through each line of file

Get each line and print it out

To print out the first column of the file, we can split each line, and print out the first column



Writing to a file in Java

```
try {
    BufferedWriter writer = new BufferedWriter(new FileWriter("output.txt"));
    writer.write("Hello World! \n");
    writer.write("----------------------------\n");
    writer.close();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Writing to a file in Java

```
try {
    BufferedWriter writer = new BufferedWriter(new FileWriter("output.txt"));
    writer.write("Hello World! \n");
    writer.write("----------------------------\n");
    writer.close();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Name of output file

Writing to a file in Java

```
try {
    BufferedWriter writer = new BufferedWriter(new FileWriter("output.txt"));
    writer.write("Hello World! \n");
    writer.write("----------------------------\n");
    writer.close();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

We write out contents to file using the `write()` method

Writing to a file in Java

```
try {
    BufferedWriter writer = new BufferedWriter(new FileWriter("output.txt"));
    writer.write("Hello World! \n");
    writer.write("----------------------------\n");
    writer.close();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

When writing to a file, it is very important to close the file when you are finished