CSCI 132:
Basic Data Structures & Algorithms

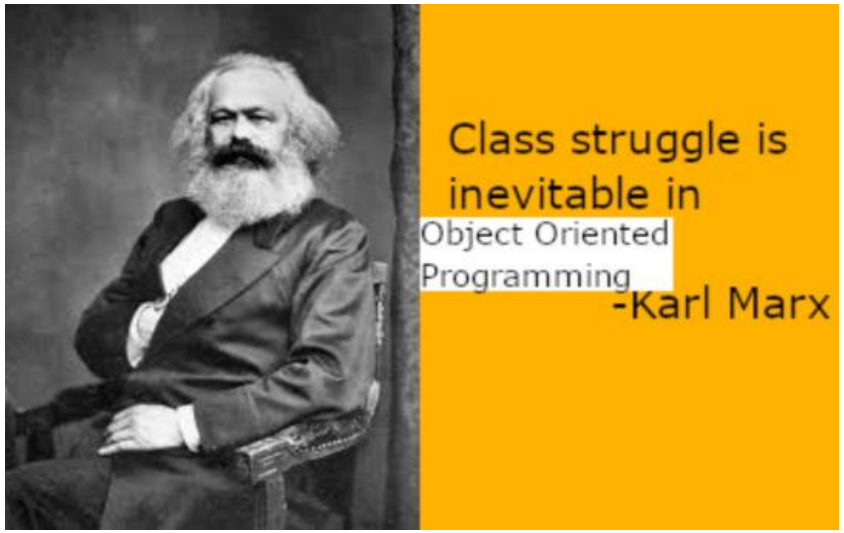
Abstract, Static, & OOP conclusion

Reese Pearsall & Iliana Castillon
Fall 24

MONTANA
STATE UNIVERSITY

Announcements

- Don’t have to download all the .java files from the website
- Can open a java project by downloading .zip
  - “workspace” from the website



Abstract classes are restricted classes that cannot be used to create objects (to access it, it must be inherited from another class)

```
public abstract class Food {

    protected String name;
    protected int calories;
    protected double price;

    public Food(String name, int calories, double price) {
        this.name = name;
        this.calories = calories;
        this.price = price;
    }
```

```
public class FoodDemo {


public static void main(String[] args) {
    Food cheetos = new Food("Cheetos", 100, 5.99);
```

Error:

Cannot instantiate the type Food

Abstract classes cannot be used to create objects

Must be inherited by another class if you want to access

Abstract methods are defined but don't provide implementation (can only be implemented using subclasses)

```
public abstract class Animal {
        public abstract void makeSound();
}
```

```
public class Bird extends Animal {
        public void makeSound() {
                System.out.println("The " + species + " goes chirp chirp chirp!");
}
```

Very similar to interfaces

abstract methods can only be used in abstract classes

Static methods are methods in Java that can be called without creating an object (instance) of a class

```
public class StaticDemo {
	public static void fun1(String arg1) {
		System.out.println(arg1);
	}
	public static void main(String[] args) {
		fun1("Hello");
	}
}
```

We do **not** need to create a StaticDemo object to call the fun1() method

Static methods are methods in Java that can be called without creating an object (instance) of a class

```java
public class StaticDemo {
        public static void main(String[] args) {
                AnotherClass.funMethod("Hello");
        }
}
```

StaticDemo.java

```java
public class AnotherClass {
    public static void funMethod(String arg)
    {
        System.out.println(arg);
    }
}
```

AnotherClass.java

If the static method is in another class, we can access it by giving the class name (AnotherClass)

Once again, I do not need to create an AnotherClass object to call this static method

However, now objects are no longer an implicit argument to this method (cant use `this` anymore)

Static methods are methods in Java that can be called without creating an object (instance) of a class

Cannot make a static reference to the non-static method funMethod() from the type AnotherClass

```
funMethod("Hello");
```

This is a very common error/warning to see in Java.

- You can turn the method static by adding the static keyword in the method definition *(Easy & quick fix)*
- Or you use OOP and call the method on an instance of the class
  - ```
    AnotherClass obj = new AnotherClass();
    Obj.funMethod("Hello");
    ```
    (Usually this is the better solution most of the time)
    ```

Static variables are shared by all instances of a class. This means that if one instance modifies the variable, the change is visible to all other instances.

```
public class Example {
        public static int count = 0; // Static var

        public Example() {
                count++;
        }

        public static void printCount() {
            System.out.println("Count: " + ExampleDemo.count);
        }
```

```
public class ExampleDemo {

        public static void main(String[] args) {
            Example e1 = new Example();
            Example e2 = new Example();
            Example.printCount(); // Prints "Count: 2"
        }
```

Access static vars by calling class name:

ExampleDemo.*count*

4 pillars of Object Oriented Programming (OOP)

Polymorphism, Abstraction, Encapsulation, Inheritance

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

```
Lion simba = new Lion(”African Lion", 400.0, "Africa", 25000, 25);
Bird private = new Bird(”Adelie penguin", 10, "Antarctica", 10000000, 15);

simba.makeSound();
private.makeSound();
```

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

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

Polymorphism also refers to the ability for an object to take many forms

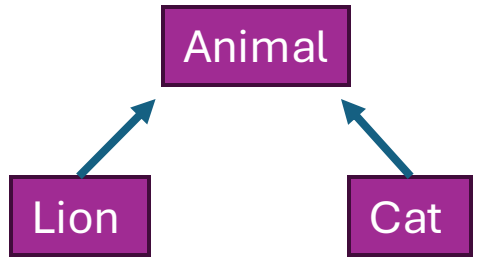
```
Lion simba = new Lion(”African Lion", 400.0, "Africa", 25000, 25);

Animal simba = new Lion(”African Lion", 400.0, "Africa", 25000, 25);
```

We can also treat the `simba` reference variable as an Animal, since Lion inherits from Animal

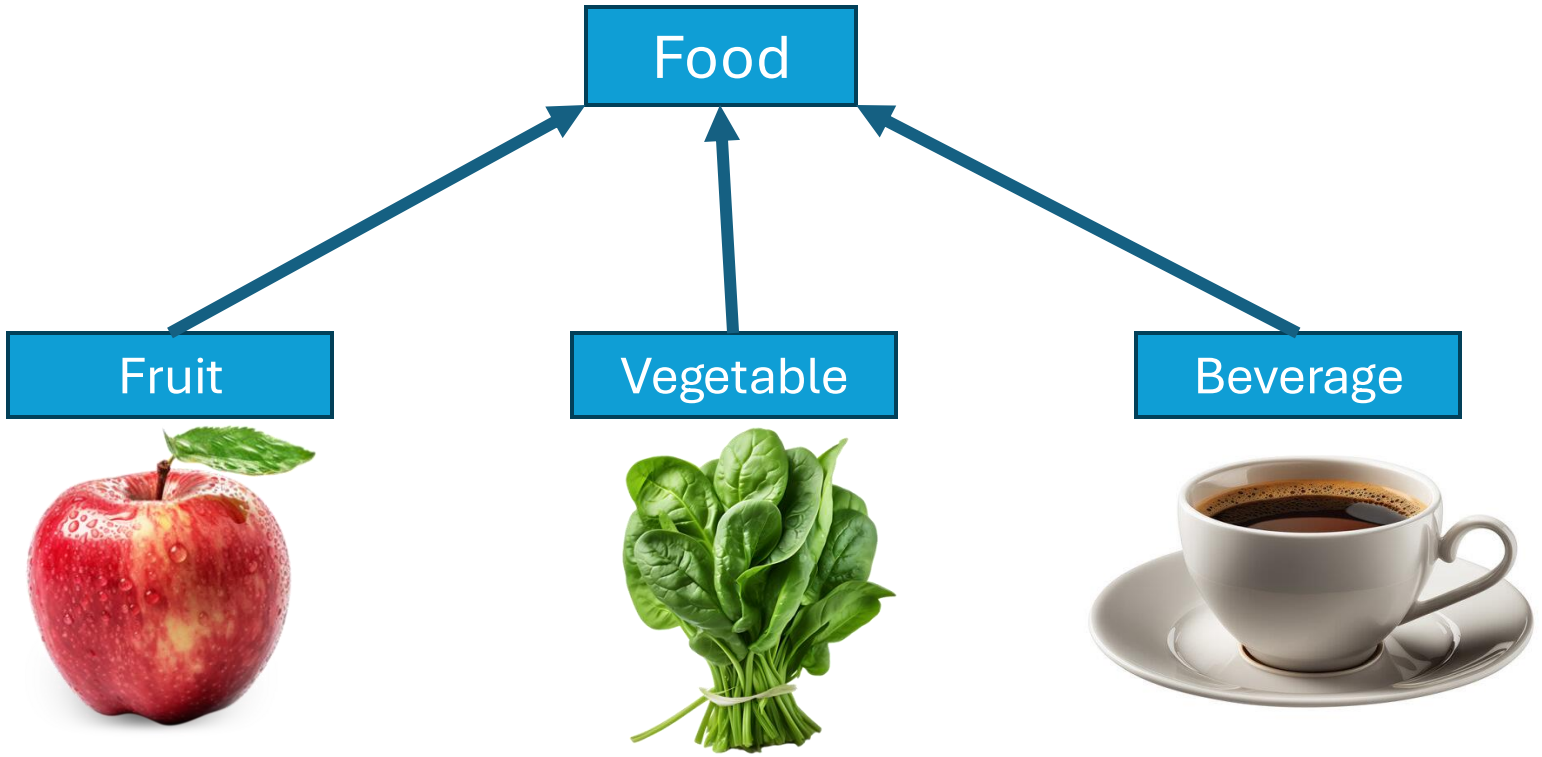
```
Animal simba = new Lion();
Animal meatball = new Cat();

Animal[] myAnimalArray = {simba, meatball};
```



Abstraction is the process of hiding certain details and showing only essential information to the user.

Abstraction can be achieved with either *abstract classes* or *interfaces*



simplifying complex systems by breaking them into more manageable parts.

`Food.java`:
Took on name, calories, and price so we didn't have to worry about them in subclasses

Encapsulation is the process of wrapping code and data together into a single unit.

Bundling of data and methods that operate on that data within a single unit, which is called a *class* (ie getters and setters)

<u>Helpful for code organization:</u>

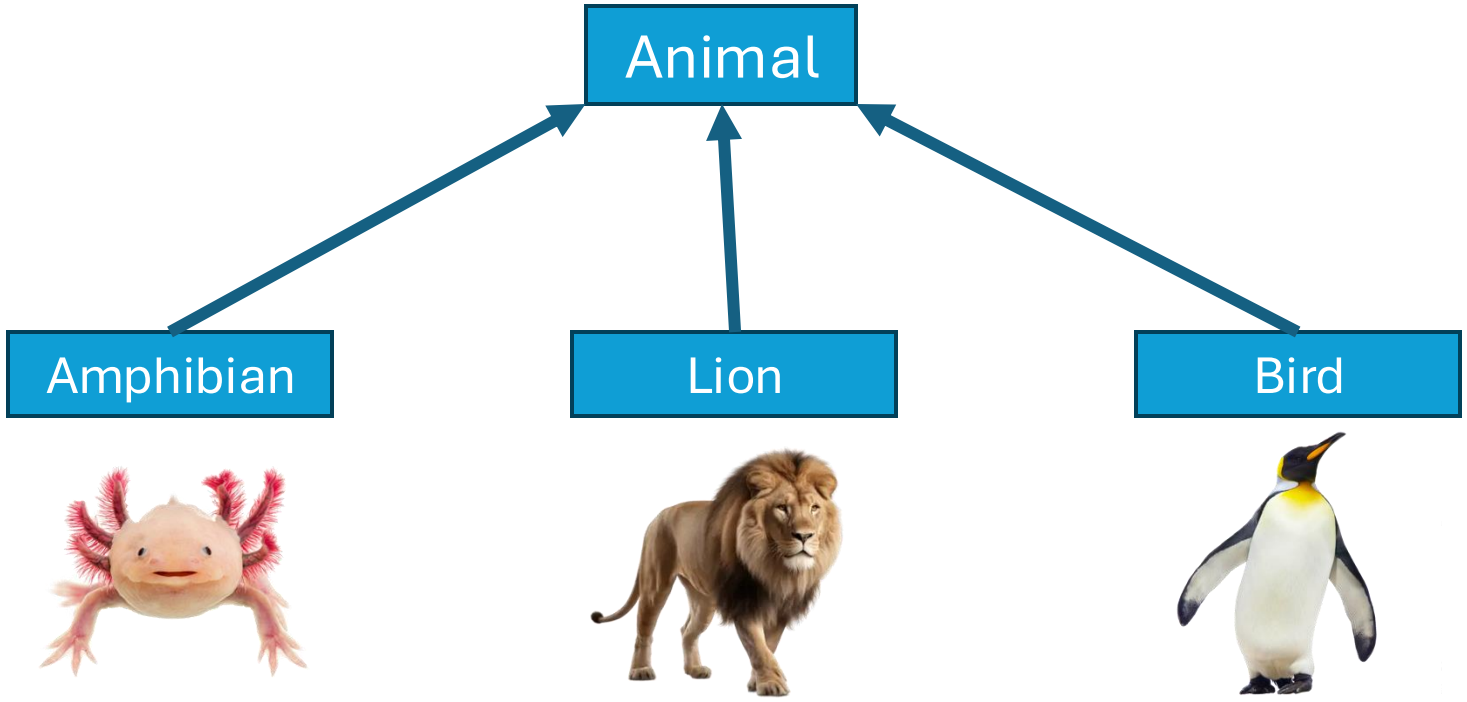
Where does it make sense to keep things together, and where are responsibilities separate?

Inheritance is the process of one class inheriting properties and methods from another class

Inheritance can be achieved with the *`extends`* keyword

Useful for code reusability:

reuse attributes and methods of an existing class when you create a new class



`final` keyword:
Creates a class that **can not** be extended

```
final class className{
}
```