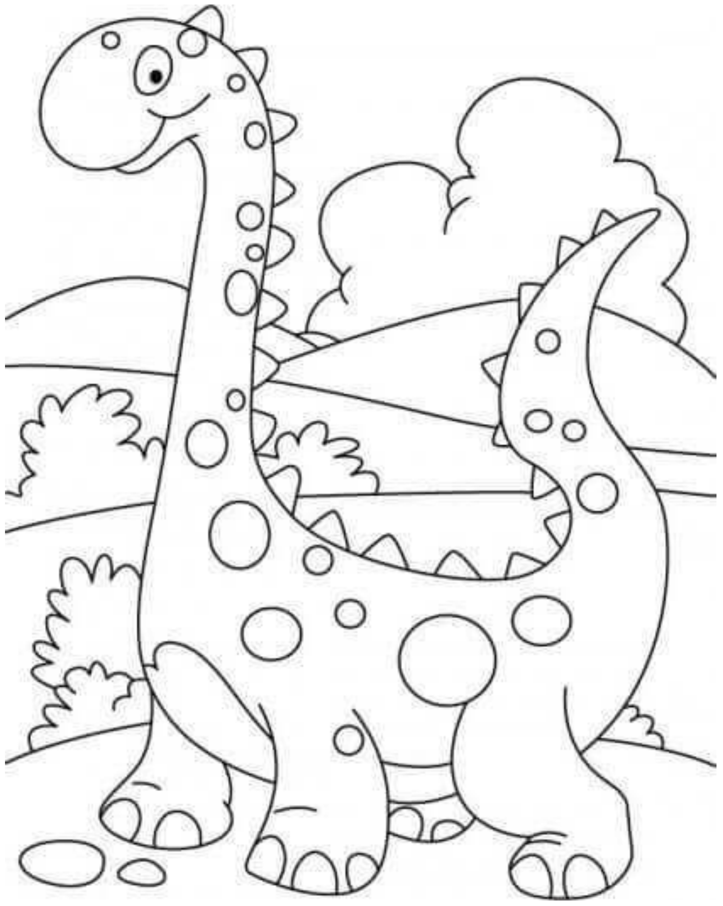
Discussion 04: Inheritance

We'll start at Berkeley time–till then, feel free to color in these friendly dinosaur using the annotate feature on Zoom! Even big kids like to color :)



Inheritance

Discussion 04

Announcements

Week 4

- ❏ Midterm 1 - **this Wednesday 2/10**
- ❏ Post-Midterm form - due **Friday 2/12**
- ❏ Lab 4 - due **next Tuesday 2/16**

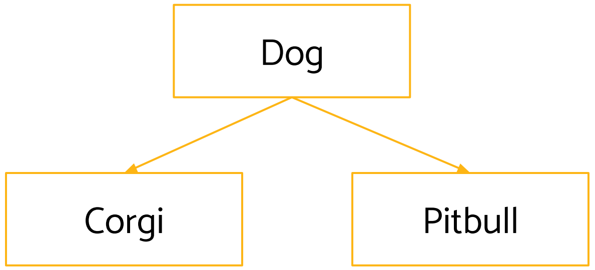
Content Review

Classes

Subclasses (or child classes) are classes that extend another class. This means that they have access to all of the functions and variables of their parent class in addition to any functions and variables defined in the child class.

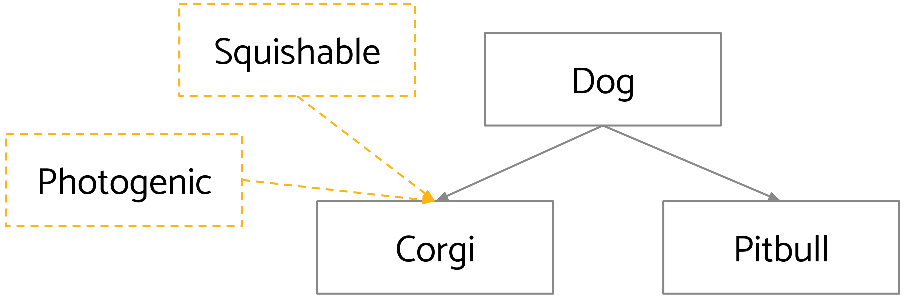
Superclasses or parent classes are classes that are extended by another class.

Classes can only extend one class but can be extended by many classes.



Interfaces

Interfaces are implemented by classes. They describe a narrow ability that can apply to many classes that may or may not be related to one another. They do not usually implement the methods they specify. One class can implement many interfaces.
Ex. Comparable



Implementation

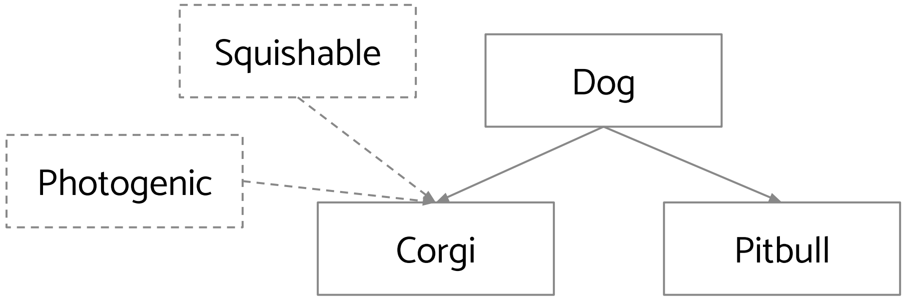
```
interface Squishable {...}

interface Photogenic {...}

class Dog {...}

class Pitbull extends Dog {...}

class Corgi extends Dog implements Squishable, Photogenic {...}
```



Fun with Methods

Method Overloading is done when there are multiple methods with the same name, but different parameters.

```
public void barkAt(Dog d) { System.out.print("Woof, it's another dog!"); }
public void barkAt(Animal a) { System.out.print("Woof, what is this?"); }
```

Method Overriding is done when a subclass has a method with the exact same function signature as a method in its superclass.

In Dog class:

```
public void speak() { System.out.print("Woof, I'm a dog!"); }
```

In Corgi Class:

```
public void speak() { System.out.print("Woof, I'm a corgi!"); }
```

Casting (not in scope for MT1, in scope eventually)

Casting allows us to tell the compiler to treat the static type of some variable as whatever we want it to be (within reason). If the cast is valid, for that line only we will treat the static type of the casted variable to be whatever we casted it to.

```
Animal a = new Dog();
Dog d = a;    // Compiler error: an animal is not a dog
Dog d = (Dog) a;    // Valid cast: an animal could reasonably be a dog
d = new Dog();
a = (Animal) d   // Valid cast: a dog definitely is-a animal
Cat c = new Cat();
d = (Dog) c;     // Compiler error: a cat is definitely not a dog
a = c;
d = (Dog) a;   // Cast compiles because an animal could reasonably be a dog. Runtime error.
```

Dynamic Method Selection

Your computer. . .

@ Compile Time:

1. Check for valid variable assignments
2. Check for valid method calls (only considering static type and static types superclass(es))

@ Run Time:

1. Check for overridden methods
2. Ensure casted objects can be assigned to their variables

Worksheet

1 Raining Cats and Dogs

```java
public class Animal {
    public String name, noise;
    public int age;

    public Animal(String name, int age) {
        this.name = name;
        this.age = age;
        this.noise = "Huh?";
    }

    public void greet() {print("Animal " + name + " says: " + this.noise);}

    public void play() {System.out.println("Woo it's fun being an animal!")}
}

public class Cat extends Animal {
    public Cat(String name, int age) {
        super(name, age);
        this.noise = "Meow!";
    }

    @Override
    public void greet() {System.out.println("Cat " + name + " says: " + this.noise);}

    public void play(String expr) {System.out.println("Woo it's fun being a cat!" + expr)}
}

public class Dog extends Animal {
    public Dog(String name, int age) {
        super(name, age);
        noise = "Woof!";
    }

    @Override
    public void greet() {System.out.println("Dog " + name + " says: " + this.noise);}

    public void play(int happiness) {
        if (happiness > 10) {
            System.out.println("Woo it's fun being a dog!")
        }
    }
}

public class TestAnimal {
    public static void main(String[] args) {
        Animal a = new Animal("Pluto", 10);
        Cat c = new Cat("Garfield", 6);
        Dog d = new Dog("Fido", 4);
        a.greet();
        c.greet();
        d.greet();
        c.play();
        c.play(":)")
        a = c;
        ((Cat) a).greet();
        ((Cat) a).play(":D");
        a.play(14);
        ((Dog) a).play(12);
        a.greet();
        c = a;
    }
}
```

4 An Exercise in Inheritance Misery

```
public class A {
    public int x = 5;
    public void m1() { System.out.println(“Am1-> ” + x); }
    public void m2() { System.out.println(“Am2-> “ + this.x); }
    public void update() { x = 99; }
}
public class B extends A {
    public void m2() { System.out.println(“Bm2-> “ + x); }
    public void m2(int y) { System.out.println(“Bm2y-> “ + y); }
    public void m3() { System.out.println(“Bm3-> “ + “called”); }
}
public class C extends B {
    public int y = x + 1;
    public void m2() { System.out.println(“Cm2-> “ + super.x); }
    public void m4() { System.out.println(“Cm4-> “ + super.super.x); }
    public void m5() { System.out.println(“Cm5-> “ + y); }
}
```

```
public class D {
public static void main (String[] args){
        B a0 = new A();
        a0.m1();
        a0.m2(16);
        A b0 = new B();
        System.out.println(b0.x);
        b0.m1();
        b0.m2();
        b0.m2(61);
        B b1 = new B();
        b1.m2(61);
        b1.m3();
        A c0 = new C();
        c0.m2();
        C c1 = (A) new C();
        A a1 = (A) c0;
        C c2 = (C) a1;
        c2.m3();
        c2.m4();
        c2.m5();
        ((C) c0).m3();
        (C) c0.m2();
        b0.update();
        b0.m1();
    }
}
```

Open Q&A Session (in preparation for MT1)