# super

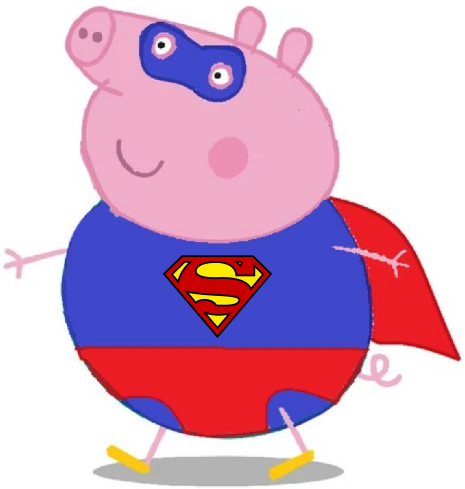
Mr. Poole
Java

# Review keyword: **super**

**super** is a reference to the superclass.

**super** helps call constructors and methods from the inherited class within the subclass.

**super** is similar to **this**.



# We've used **super** for our constructors!

```
public class Dog{
      private String name;
      private int age;

      public Dog() {
            name = "Toto";
            age = 3;
      }
      public Dog(String n, int a){
            name = n;
            age = a;
      }
}
```

```
public class Corgi extends Dog{
      private String color;
      public Corgi() {
            super();
            color = "Brown";
      }
      public Corgi(String n, int a, String c){
            super(n, a);
            color = c;
      }
}
```

The following code uses **super** to use the correct constructor when creating the Corgi.

```
Corgi joey = new Corgi("Joey", 5, "Blue");
```

**super** calls methods, constructors, and values from the superclass to use in the subclass.

# Using **super**

For constructors we've used the following

```
super();
super(n, a);
```

in place of

```
Dog();
Dog(n, a);
```

So how would we call
the Dog's bark() method inside of
the Greyhound's bark() method?

```
public class Greyhound extends Dog{
      private String color;

      public Greyhound () {...}
      public Greyhound (String n, int a, String c){...}

      public boolean isFast(){
            return true;
      }
      public void bark(){
            System.out.println("LOUD BARK!");
      }
}
```

# Using **super**

For constructors we've used the following

```
super();
super(n, a);
```

in place of

```
Dog();
Dog(n, a);
```

A good guess would be this:

But this would call THIS method.

```
public class Greyhound extends Dog{
      private String color;

      public Greyhound () {...}
      public Greyhound (String n, int a, String c){...}

      public boolean isFast(){
            return true;
      }
      public void bark(){
            bark();
            System.out.println("LOUD BARK!");
      }
}
```

# Using **super**

For constructors we've used the following

```
super();
super(n, a);
```

in place of

```
Dog();
Dog(n, a);
```

The correct way to do this is:

```
super.bark();
```

**super** calls the superclass and tells it to use the **bark** method.

```
public class Greyhound extends Dog{
      private String color;

      public Greyhound () {...}
      public Greyhound (String n, int a, String c){...}

      public boolean isFast(){
            return true;
      }
      public void bark(){
            super.bark();
            System.out.println("LOUD BARK!");
      }
}
```

# Using **super**

```
public class Dog{
      private String name;
      private int age;

      public Dog() {...}
      public Dog(String n, int a){...}

      public void bark(){
            System.out.println("Bark!");
      }
}
```

```
public class Greyhound extends Dog{
      private String color;

      public Greyhound () {...}
      public Greyhound (String n, int a, String c){...}

      public boolean isFast(){
            return true;
      }
      public void bark(){
            super.bark();
            System.out.println("LOUD BARK!");
      }
}
```

What's the output of the following now?

```
Greyhound rapid = new Greyhound("Rapid", 7, "grey");
rapid.bark();
```

# Using **super**

```
public class Dog{
      private String name;
      private int age;

      public Dog() {...}
      public Dog(String n, int a){...}

      public void bark(){
            System.out.println("Bark!");
      }
}
```

```
public class Greyhound extends Dog{
      private String color;

      public Greyhound () {...}
      public Greyhound (String n, int a, String c){...}

      public boolean isFast(){
            return true;
      }
      public void bark(){
            super.bark();
            System.out.println("LOUD BARK!");
      }
}
```

1

2

3

Our new output for the code below is:

```
Greyhound rapid = new Greyhound("Rapid", 7, "grey");
rapid.bark();
```

```
Bark!
LOUD BARK!
```

This is because Greyhound's bark method calls Dog's bark method first.

# Using **super**

```
public class Dog{
      private String name;
      private int age;

      public Dog() {...}
      public Dog(String n, int a){...}

      public void bark(){
            System.out.println("Bark!");
      }
}
```

```
public class Greyhound extends Dog{
      private String color;

      public Greyhound () {...}
      public Greyhound (String n, int a, String c){...}

      public boolean isFast(){
            super.bark();
            return true;
      }
      public void bark(){
            System.out.println("LOUD BARK!");
      }
}
```

You can also call superclass methods wherever you want!

Example: call Dog's **bark()** method in Greyhound's isFast()

This works perfectly fine too! It just barks before!

# Lab Part 1: super

1. Create an Apprentice class
    a. This class **inherits Musician**.
    b. Global Variables - **String** school, **int** yearsOfExperience
    c. Constructors
        i. **Empty** - default school = “CVHS”, yearsOfExperience = 0
        ii. **String** school, **int** yearsOfExperience
        iii. String **instrument, school, yearsOfExperience**
        **iv. Name, age, instrument, school, yearsOfExperience**
    d. Methods
        i. **Override** playInstrument
        ii. **Override** old practice + new practice (say how yearsOfExperience of practice)
        iii. **Override** old perform + new perform
        iv. **New Method** practiceAtUniversity, old practice + at school

# Lab Part 2: super

2. In Main

a. Create 4 Apprentices

| | | |
|---|---|---|
| i. | Empty Apprentice | - Call playInstruments() |
| ii. | String, int | - Call practice() |
| iii. | String, String, int | - Call perform() |
| iv. | String, int, String, String, int | - Call practiceAtUniversity() |