Constructors for Subclasses

Mr. Poole
Java

First: Constructors are **not** inherited.

```
public class Dog{
     public Dog() {...}
     public Dog(String name, int age){...}
}
```

```
public class Corgi extends Dog{
     public Corgi() {...}
     public Corgi(String color){...}
}
```

```
Dog toto = new Dog();
Corgi joey = new Corgi();
Dog toto = new Dog("Toto", 3);
Corgi joey = new Corgi ("brown");
Corgi joey = new Corgi ("Joey", 5);
```

The String, int constructor isn't declared in Corgi, it can't be used.
Constructors are **not** inherited.

Now how do subclass constructors actually work?

```
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() {
            color = "Brown";
      }
      public Corgi(String c){
            color = c;
      }
}
```

If we call the following, what happens?

```
Corgi joey = new Corgi("Blue");
```

Now how do subclass constructors actually work?

```
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() {
            color = "Brown";
      }
      public Corgi(String c){
            color = c;
      }
}
```

1

If we call the following, what happens?

```
Corgi joey = new Corgi("Blue");
```

It starts by calling the String constructor in Corgi!

BUTTTTT, it doesn't start by setting color = c;

Now how do subclass constructors actually work?

```
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;
      }
}
```

2

1

```
public class Corgi extends Dog{
      private String color;
      public Corgi() {
            color = "Brown";
      }
      public Corgi(String c){
            color = c;
      }
}
```

If we call the following, what happens?

```
Corgi joey = new Corgi("Blue");
```

The first thing that happens when calling the Corgi String constructor is that **it calls the empty Dog constructor**.

Since Corgi extends Dog, it inherits Dog's name and age through the constructor.

Now how do subclass constructors actually work?

```
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;
      }
}
```

2

1

```
public class Corgi extends Dog{
      private String color;
      public Corgi() {
            color = "Brown";
      }
      public Corgi(String c){
            color = c;
      }
}
```

If we call the following, what happens?

```
Corgi joey = new Corgi("Blue");
```

So imagining this so far our Corgi has the following:

Corgi
name = "Toto" age = 3

Now how do subclass constructors actually work?

```
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;
      }
}
```

2

1

3

```
public class Corgi extends Dog{
      private String color;
      public Corgi() {
            color = "Brown";
      }
      public Corgi(String c){
            color = c;
      }
}
```

If we call the following, what happens?

```
Corgi joey = new Corgi("Blue");
```

After finishing the Dog empty constructor, it goes back to the String Constructor.

Now it sets color.

Corgi
name = "Toto" age = 3
color = "Blue"

Now how do subclass constructors actually work?

```
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() {
            color = "Brown";
      }
      public Corgi(String c){
            color = c;
      }
}
```

1

2

3

If we call the following, what happens?

```
Corgi joey = new Corgi("Blue");
```

Lastly, the Corgi will inherit the **bark**() method and Corgi has the **hasSmallLegs()** method.

Corgi
name = "Toto" age = 3
color = "Blue"
bark()
hasSmallLegs()

In short: Java calls the inherited constructor before executing the subclass constructor.

How does Java actually do that though?

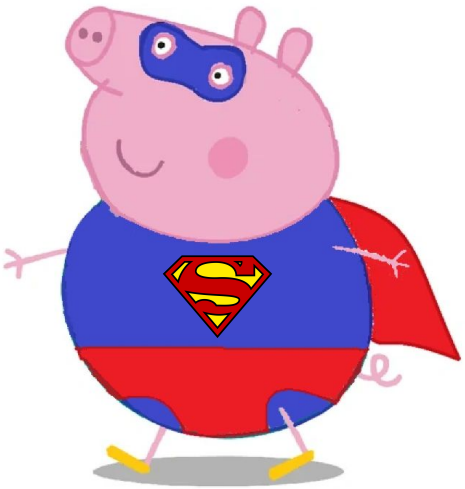
New keyword: **super**

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**.

Now let’s use it!



Let's show how **super** is used here.

```
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 c){
            super();
            color = c;
      }
}
```

```
Corgi joey = new Corgi("Blue");
```

When not written,
Java inserts **super** into the subclass constructors.

NOTE: super() looks exactly like the empty constructor. That's how we know what we're calling in the superclass.

Corgi
name = "Toto" age = 3
color = "Blue"
bark()
hasSmallLegs()

Let's show how **super** is used here.

```
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 c){
            super();
            color = c;
      }
}
```

1

2

3

```
Corgi joey = new Corgi("Blue");
```

This code does the exact same actions as before.

But now it's just defined by calling the default constructor

Corgi
name = "Toto" age = 3
color = "Blue"
bark()
hasSmallLegs()

Now we know that Java calls the empty/default constructor inherently.

What if we want to use a different constructor from the superclass?

We can control which constructor Java uses!

```
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();
            color = c;
      }
}
```

Let's change our Corgi constructor!

We want it to give a **name, age, and color**!

Now we're, not using the parameter **name and age** yet…

Let's use the superclass constructor to use them!

Guess what we're going to change here to do so.

We can control which constructor Java uses!

```
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;
      }
}
```

That's right! We'll change **super()**!

super() looks like our default Dog() constructor

super(n, a) looks like our **String, int constructor**.

Since **n** is a parameter String and **a** is an integer. **n** and **a** are passed up to the Dog constructor as parameters to be used there.

Now this calls our **String, int constructor** in Dog

We can control which constructor Java uses!

```
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;
      }
}
```

Now what does the following code do?

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

We can control which constructor Java uses!

```
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;
      }
}
```

2

```
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;
      }
}
```

3

1

Now what does the following code do?

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

We follow this path to give this output:

Corgi
name = "Joey" age = 5
color = "Blue"
bark()
hasSmallLegs()

Using **super** correctly, can help us choose which constructor of the superclass to call.

Lab: Constructors Subclasses

1. Built on top of your Performer and Musician files
2. In Performer
   a. Add a new constructor that is just String name. Make sure you default age.
3. In Musician
   a. Create a new constructor that takes **String name, String instrument**
      i. Call the correct superclass constructor
   b. Create a new constructor that takes **String name, int age, String instrument**
      i. Call the correct superclass constructor
4. Create 2 new Musicians in main.
   a. A String, String musician - Call getName(), practice() and getInstrument()
   b. A String, int, String musician - Call getName(), perform() and playInstrument