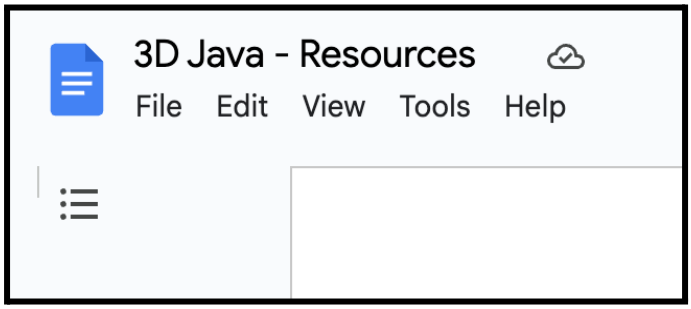
3D Java Resources

https://www.csteachingtips.org/3D
https://tinyurl.com/3DJavaResources (this file)
https://tinyurl.com/JavaAnchorCode

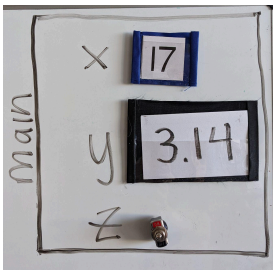
Click on the document outline in the top left corner to navigate between units:



Unit 1 - Primitive Types

(1.1) Types: Variables can hold different types (video, slides)

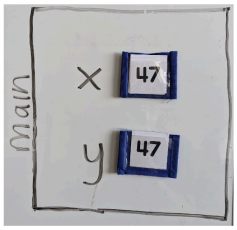
```
1 public class Examples {
2     public static void main(String[] args) {
3         int x = 17;
4         double y = 3.14;
5         boolean z = false;
6     }
7 }
```



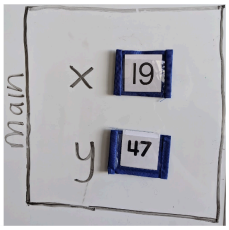
(1.2a) Assignment: Two int variables are independent and do not become linked (slides)

```
1 public class Examples {
2     public static void main(String[] args) {
3         int x = 47;
4         int y = x;
5         x = 19;
6     }
7 }
```

Before Line 5

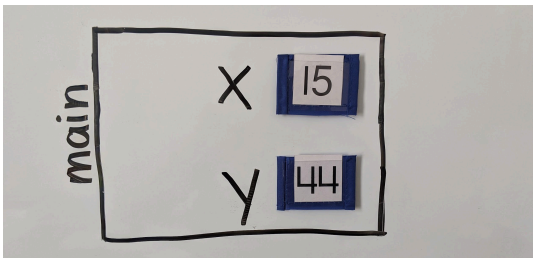


After Line 5



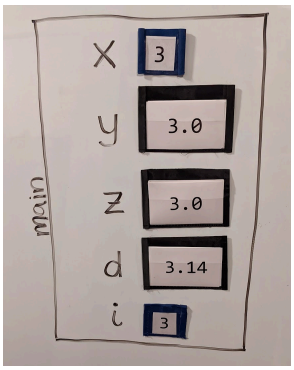
(1.2b) Assignment: Two int variables are independent and do not become linked (video)

```
public class Example1{
    public static void main(String[] args){
        int x = 44;
        int y = x;
        x = 15;
    }
}
```



(1.3) Casting: We can convert/cast between the types int and double by putting the new type in parentheses (slides, videos)

```java
public class Examples {
    public static void main(String[] args) {
        int x = 3;
        double y = (double) x;
        double z = x;

        double d = 3.14;
        int i = (int) d;
    }
}
```

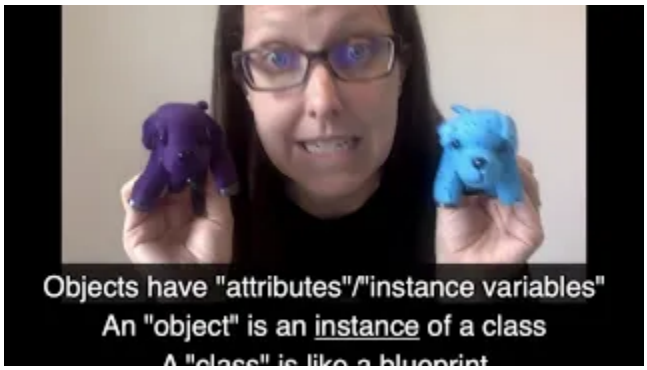


(1.4) Casting: Any calculation involving a double produces a double value (slides)

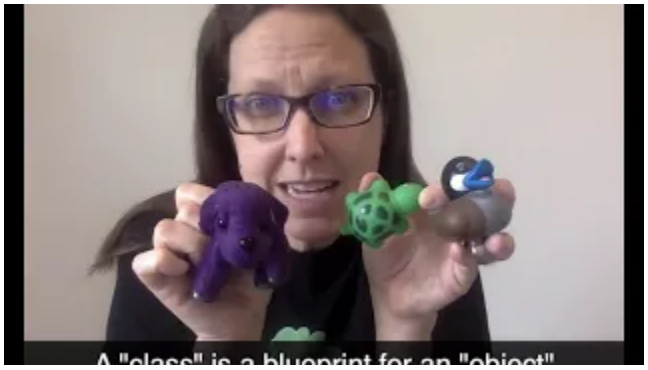
```java
public class Anchor01b {
    public static void main(String[] args) {
        int i = 7/2; // 3
        double v = 7/2; // 3.0
        double w = (double) (7/2); // 3.0
        double x = 7/2.0; // 3.5
        double y = 7.0/2; // 3.5
        double z = (double) 7/2; // 3.5
    }
}
```

Unit 2 - Using Objects

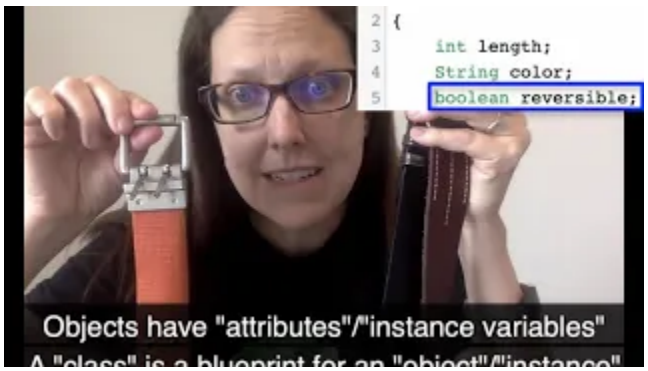
(2.1) Vocab: Class, Object/Instance, Attributes/Instance Variables (video)



(2.2) Vocab: Class, Object, Methods, Instance Variables (video)

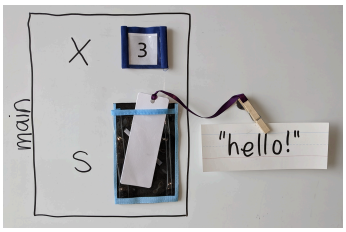


(2.3) Objects: Every Java object has their own copy of the attributes/instance variables (video)



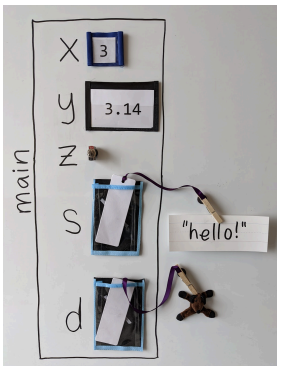
(2.4) References: A reference is like a remote control that keeps track of the object (slides, video)

```java
public class Examples {
    public static void main(String[] args) {
        int x = 3;
        String s = "hello!";
    }
}
```



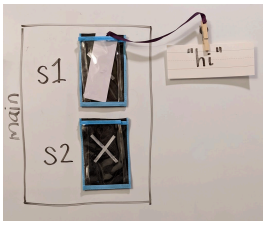
(2.5) Types: Non-primitive variable types always store a reference (slides, video)

```java
public class Examples {
    public static void main(String[] args) {
        int x = 3;
        double y = 3.14;
        boolean z = false;

        String s = "hello!";
        Dog d = new Dog();
    }
}
```



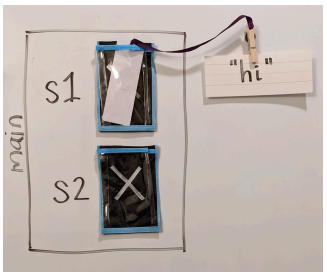
(2.6) `null` String reference: If we don't give a reference variable a value, it will be `null` (slides, video)

```
1 public class Examples {
2     public static void main(String[] args) {
3         String s1 = "hi";
4         String s2;
5     }
6 }
```



(2.7) Null Pointer Exception: We get a Null Pointer Exception when we call a method on a variable that is null (slides)

```
1 public class Examples {
2     public static void main(String[] args) {
3         String s1 = "hi";
4         System.out.println(s1.length());
5         String s2;
6         System.out.println(s2.length());
7     }
8 }
```

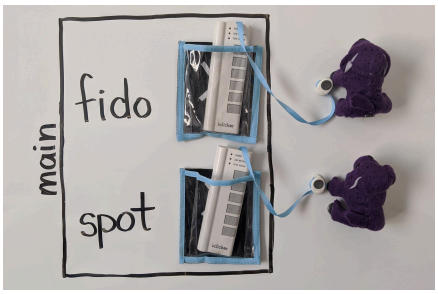


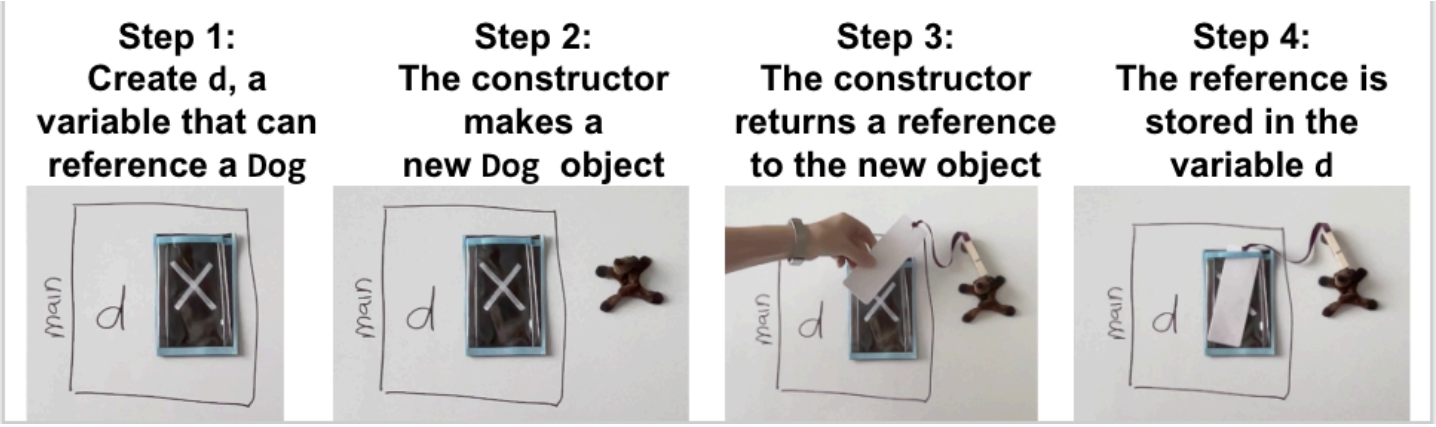
(2.8a) Constructors: Calling a constructor makes a new object (slides, video)

```
1 public class Examples {
2     public static void main(String[] args) {
3         Dog d = new Dog();
4     }
5 }
```

(2.8b) Constructors: Calling a constructor makes an object (Video)

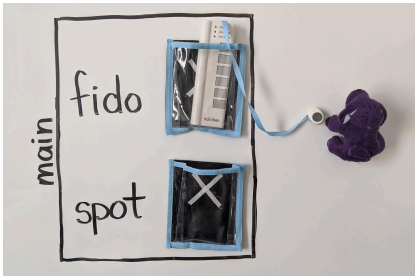
```
public class Example7{
    public static void main(String[] args){
        Dog fido = new Dog();
        Dog spot = new Dog();
    }
}
```



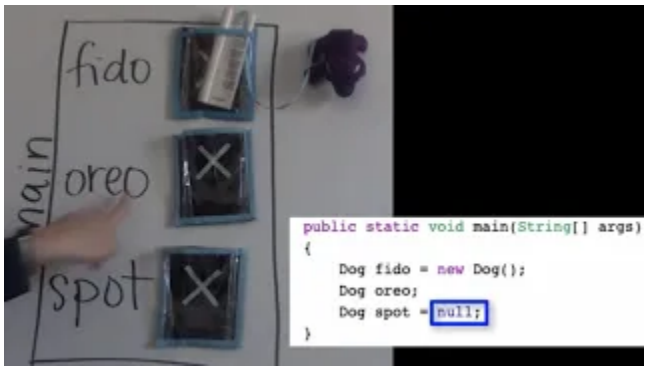


(2.9a) Null: A reference that isn't set is a `null` reference (video)

```
public class Example8{
    public static void main(String[] args){
        Dog fido = new Dog();
        Dog spot;
    }
}
```

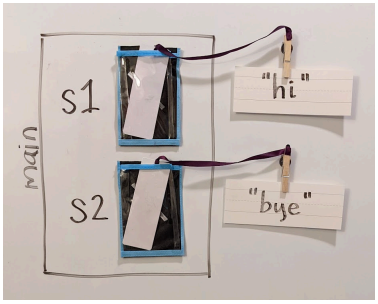


(2.9b) `null`: If we don't give a reference variable a value, it will be `null` (video)



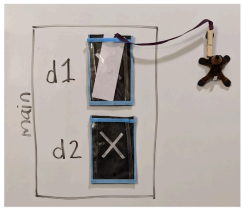
(2.10) Objects: Objects each have their own data (slides)

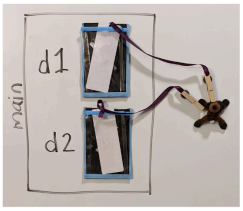
```
1 public class Examples {
2     public static void main(String[] args) {
3         String s1 = "hi";
4         String s2 = "bye";
5         System.out.println(s1.length());
6         System.out.println(s2.length());
7     }
8 }
```



(2.11a) Aliasing: (Dog Alias) In an assignment statement, we always copy the value in the variable (slides, video)

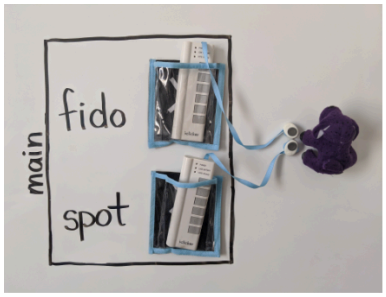
```
1 public class Examples {
2     public static void main(String[] args) {
3         Dog d1 = new Dog();
4         Dog d2 = d1;
5     }
6 }
```



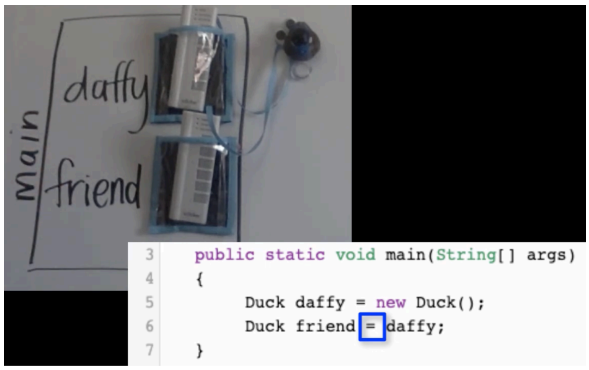


(2.11b) Aliasing: Two variables can reference the same object and both of them can modify it (video)

```
public class Example9{
    public static void main(String[] args){
        Dog fido = new Dog();
        Dog spot = fido;
    }
}
```

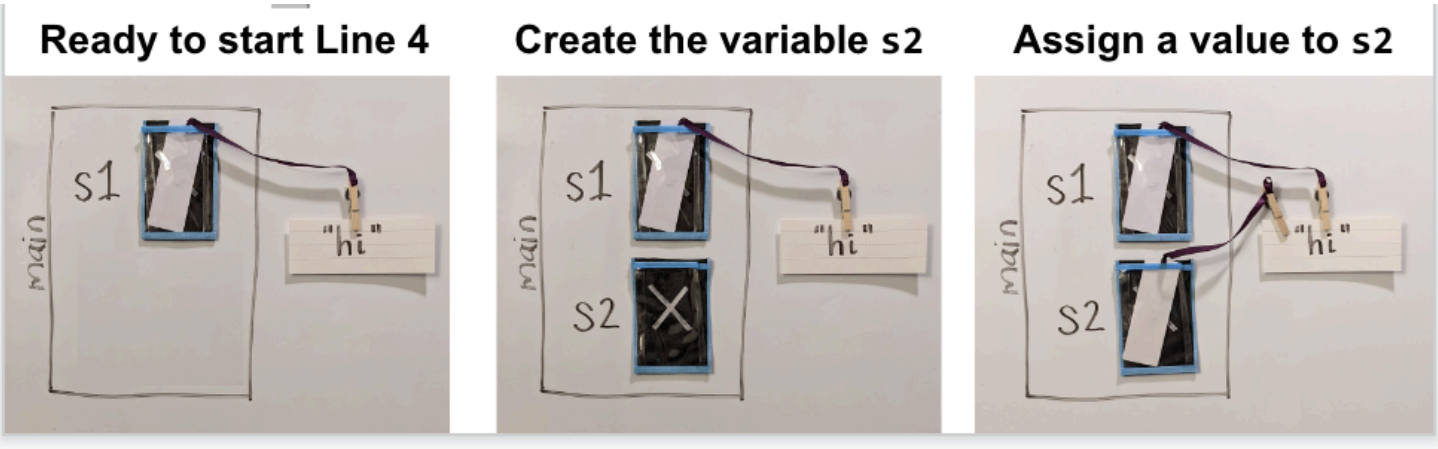


(2.11c) Aliasing: Two variables can reference the same object (video)



(2.12) Aliasing: (String Alias) In an assignment statement, we always copy the value in the variable (slides, video)

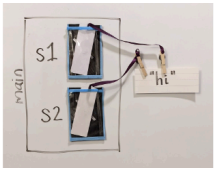
```
1 public class Examples {
2     public static void main(String[] args) {
3         String s1 = "hi";
4         String s2 = s1;
5     }
6 }
```



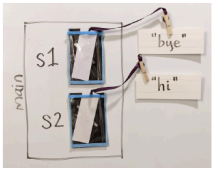
(2.13) Assignment: Variables do not become linked (slides)

```
public class Examples {
    public static void main(String[] args) {
        String s1 = "hi";
        String s2 = s1;
        s1 = "bye";
    }
}
```

Before Line 5



After Line 5



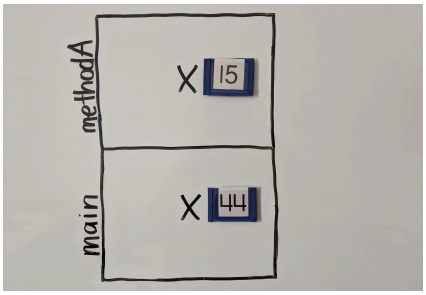
(2.14) Aliasing: When two variables reference the same object, they can both call methods on it (DRAFT; slides)

```
public class Anchor02b {
    public static void main(String[] args){
        Dog d1 = new Dog();
        Dog d2 = d1;
        Dog d3;
        d1.getOlder();
        d2.getOlder();
    }
}
```

Unit 2 Bonus - Writing and calling static methods

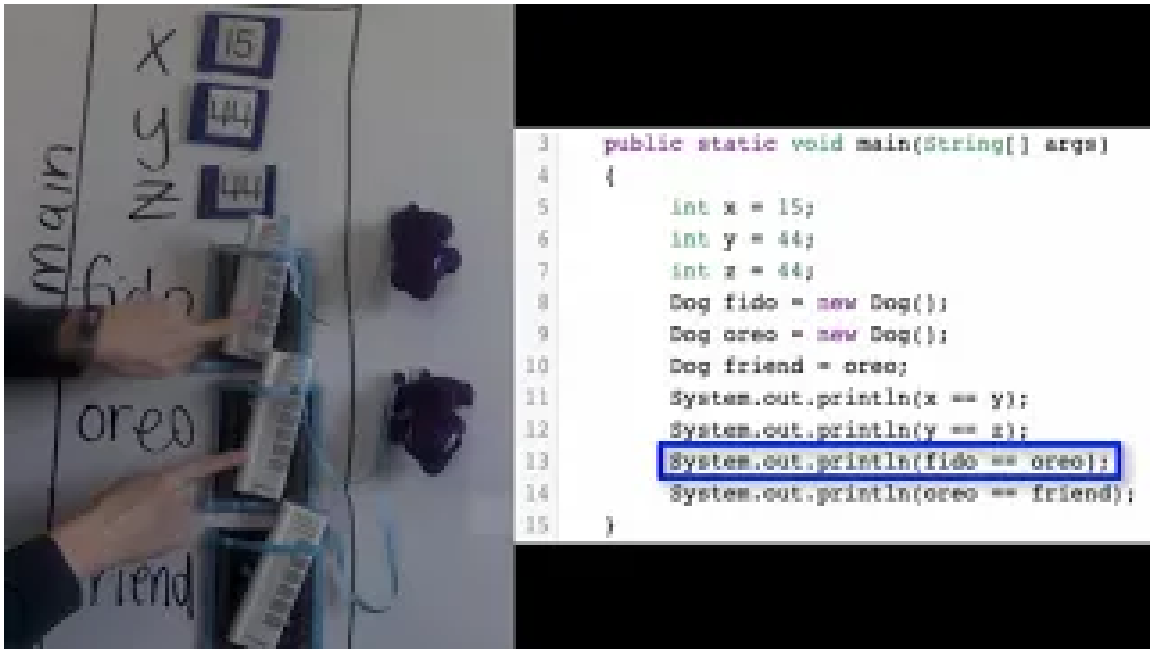
(2.15) Assignment: Calling a method with an argument is another way to assign a value to a variable (video)

```
public class Example2{
    public static void methodA(int x){
        x = 15;
    }
    public static void main(String[] args){
        int x = 44;
        methodA(x);
    }
}
```



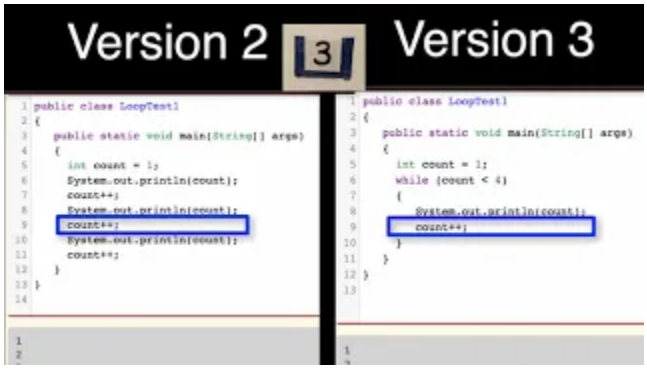
Unit 3 - Booleans and Conditionals

(3.1) Equality: == with `int`s and object references check if the variable's contents are identical (video)

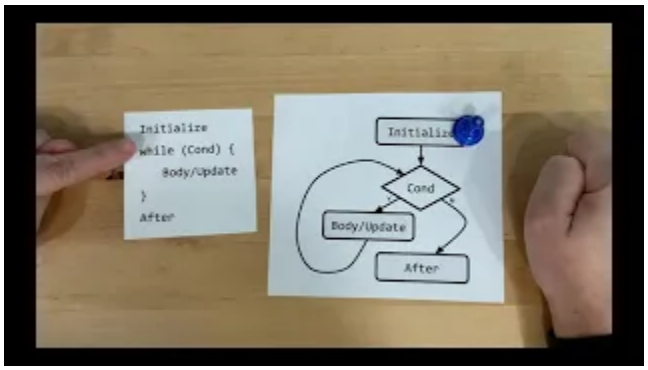


Unit 4 - Loops/Iteration

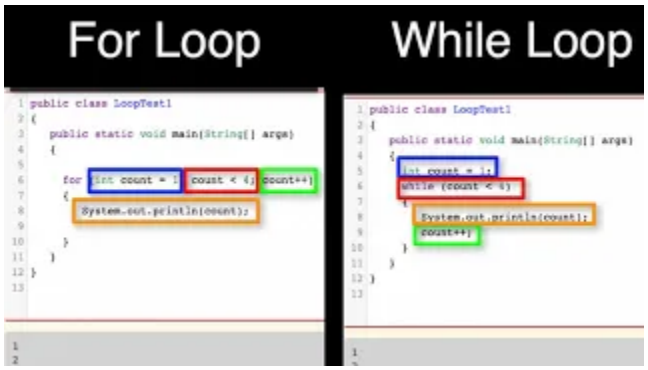
(4.1) `while`: `while` loops allow us to repeat actions until a condition is not met (video)



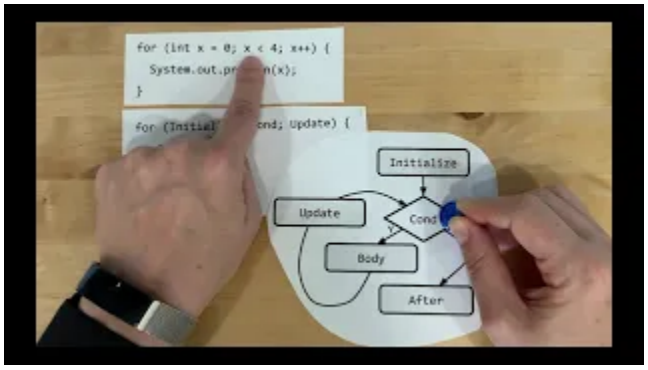
(4.2) `while`: `while` loops check the condition before executing the body of the loop (video)



(4.3) `for` loops: `while` loops can be rewritten as a `for` loop (video)



(4.4) `for` loops: `for` loops separate the body of the loop from the update of the loop variable (video)

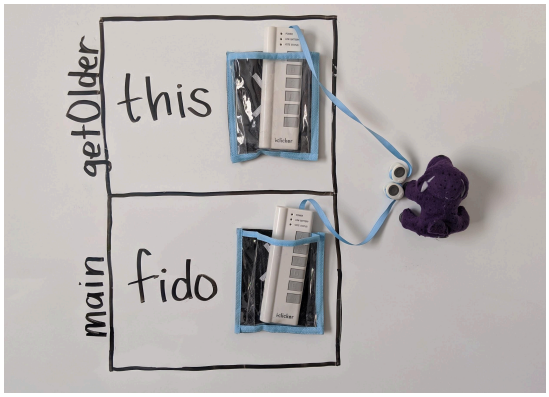


Unit 5 - Writing Classes

Most of Unit 2 is relevant!

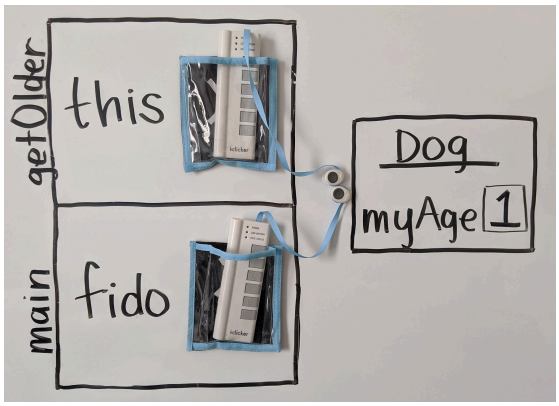
(5.1) `this`: The variable `this` is created when we call a method on an object (video)

```
public class Dog{
    private int myAge;
    public void getOlder(){
        this.myAge++;
    }
    public static void main(String[] args){
        Dog fido = new Dog();
        fido.getOlder();
    }
}
```



(5.2) Instance Variables: An object contains the values of each of the instance variables in the class (video)

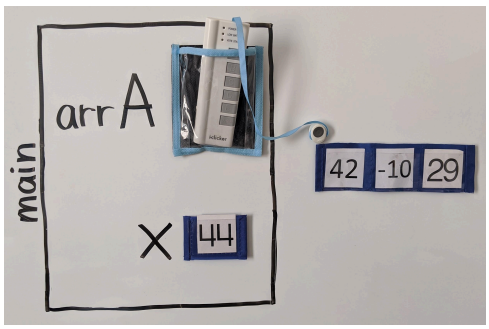
```
public class Dog{
    private int myAge;
    public void getOlder(){
        this.myAge++;
    }
    public static void main(String[] args){
        Dog fido = new Dog();
        fido.getOlder();
    }
}
```



Unit 6 - Arrays

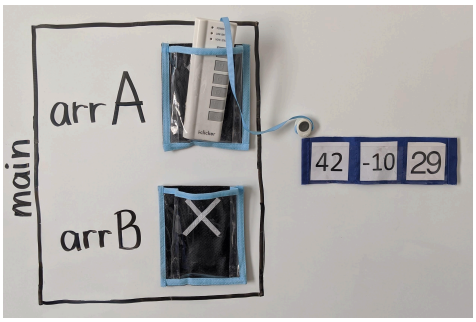
(6.1) `int` array: A variable can hold a reference to an int array (video)

```
public class Example3{
    public static void main(String[] args){
        int[] arrA = {42, -10, 29};
        int x = 44;
    }
}
```



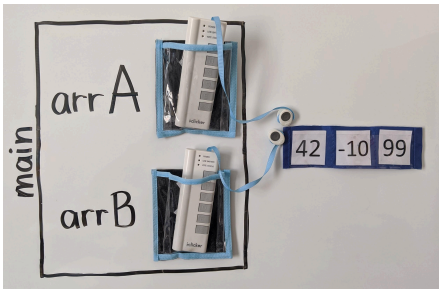
(6.2) `null`: A reference that isn't set is a `null` reference (video)

```
public class Example4{
    public static void main(String[] args){
        int[] arrA = {42, -10, 29};
        int[] arrB;
    }
}
```



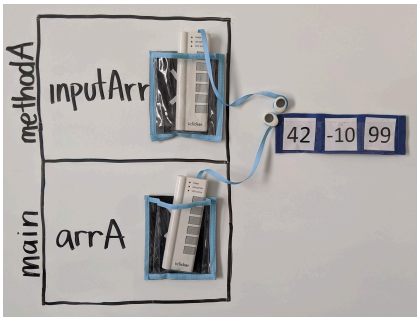
(6.3) Aliasing: Two variables can reference the same `int` array and both of them can modify it (video)

```
public class Example5{
    public static void main(String[] args){
        int[] arrA = {42, -10, 29};
        int[] arrB = arrA;
        arrA[2] = 99;
    }
}
```



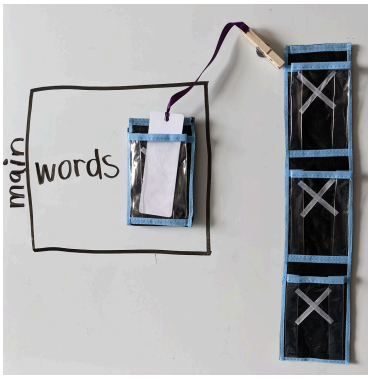
(6.4) Aliasing: An array passed to a method can be modified within that method (video)

```
public class Example6{
    public static void methodA(int[] inputArr){
        inputArr[2] = 99;
    }
    public static void main(String[] args){
        int[] arrA = {42, -10, 29};
        methodA(arrA);
    }
}
```



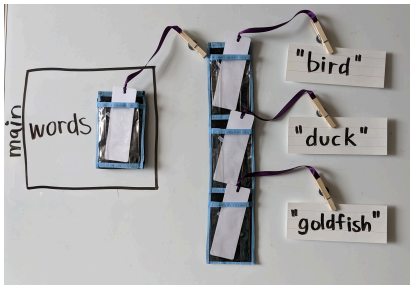
(6.5) Creating Arrays: When we create an array of `Strings`, we create `String` references. If we don't give a reference variable a value, it will be `null`.

```
1 public class Example {
2     public static void main(String[] args) {
3         String[] words = new String[3];
4     }
5 }
```



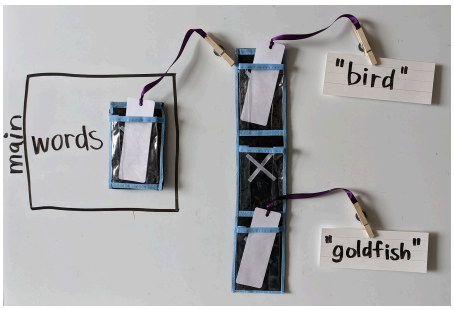
(6.6) Creating Arrays: When we first create an array variable, we can give it a value by putting the contents of the array in curly braces. Inside the curly braces, each element of the array is separated by commas.

```
1 public class Example {
2     public static void main(String[] args) {
3         String[] words = {"bird", "duck", "goldfish"};
4     }
5 }
```



(6.7) Array Assignment: We refer to an element in an array by using the array name followed by an index in square brackets.

```
1 public class Example {
2     public static void main(String[] args) {
3         String[] words = new String[3];
4         words[0] = "bird";
5         words[2] = "goldfish";
6     }
7 }
```

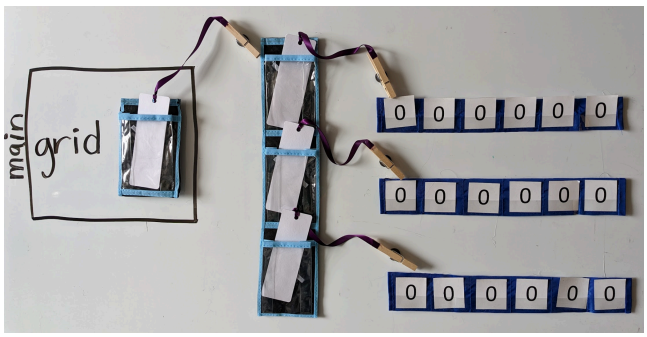


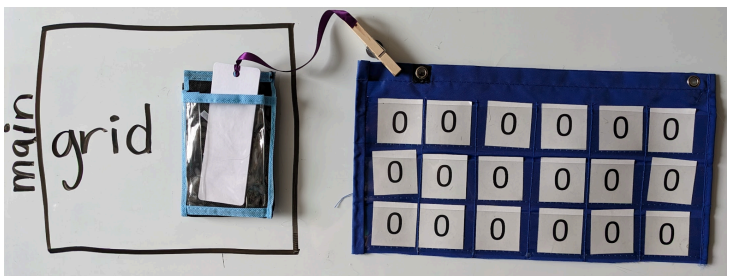
Unit 7 - ArrayLists

Unit 8 - 2D Arrays

(8.1) Creating 2D Arrays: An `int` 2D array is really an array of `int[]` references, but we think about it like a table of `int` values.

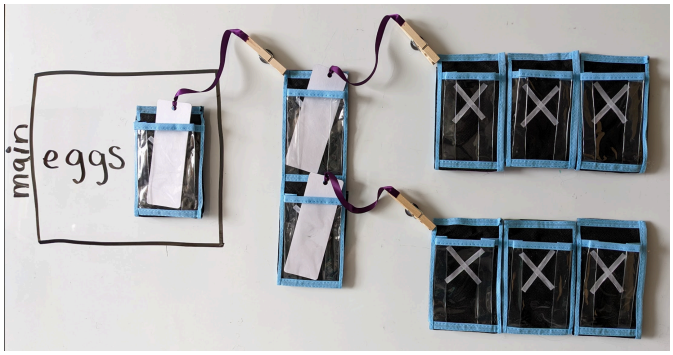
```
1 public class Example {
2     public static void main(String[] args) {
3         int[][] grid = new int[3][6];
4     }
5 }
```

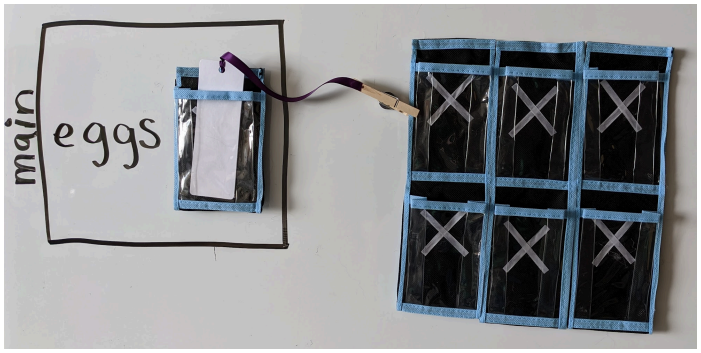




(8.2) Creating 2D Arrays: A `String` 2D array is really an array of `String[]` references, but we think about it like a table of `String` values.

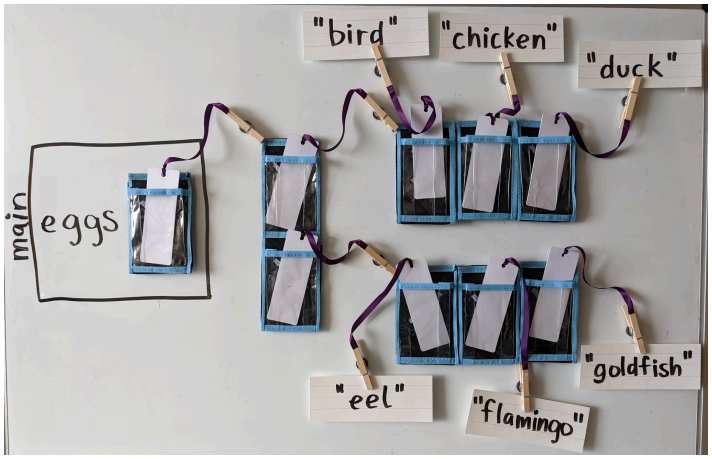
```
1 public class Example {
2     public static void main(String[] args) {
3         String[][] eggs = new String[2][3];
4     }
5 }
```





(8.3) Creating 2D Arrays: When we first create an array variable, we can give it a value by putting the contents of the array in curly braces. Inside the curly braces, each element of the array is separated by commas.

```
1 public class Example {
2     public static void main(String[] args) {
3         String[][] eggs = {{"bird", "chicken", "duck"}, {"eel", "flamingo", "goldfish"}};
4     }
5 }
```

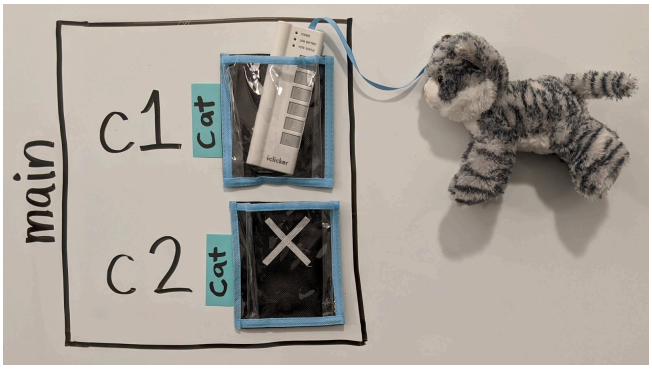


Unit 9 - Inheritance

All Inheritance Videos - YouTube Playlist
A paper about our approach: https://doi.org/10.1145/3641554.3701871

(9.1) Variable Type vs Object Type: There is a difference between the type of a variable and the type of an object (video)

```
public class Example1 {
    public static void main (String [] args) {
        Cat c1 = new Cat();
        Cat c2;
    }
}
```



(9.2) Method Execution - No Inheritance: When we call a method on an object, we look in the object's class to find the code for the method (video)

```
public class Example2 {
    public static void main (String [] args) {
        Cat c = new Cat();
        Tiger t = new Tiger();
        c.sleep();
        c.sayHello();
        t.swim();
        t.sayHello();
        t.sleep();
    }
}
```

(9.3) Syntax to Extend a Class: A class can extend another class to reduce redundant code, and an extending class can add methods, inherit methods, or override methods (video)

```
public class Cat {
    public void sleep() {
        System.out.println("Zzzzzzz");
    }
    public void sayHello() {
        System.out.println("Purr");
    }
}
```

```
public class Tiger extends Cat{

    public void sayHello() {
        System.out.println("Roar");
    }
    public void swim() {
        System.out.println("Splash Splash");
    }
}
```

(9.4) Method Execution with Inheritance: If we don't find a method in a class, we look in the parent's class (video)

```
public class Example2 {
    public static void main (String [] args) {
        Cat c = new Cat();
        Tiger t = new Tiger();
        c.sleep();
        c.sayHello();
        t.swim();
        t.sayHello();
        t.sleep();
    }
}
```

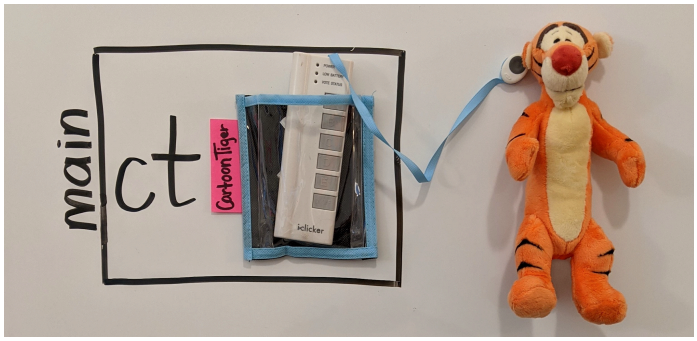
(9.5) Inheriting a Grandparent's Method: If we don't find a method in a class, we look in the parent's class (video)

```
public class Cat {
    public void sleep() {
        System.out.println("Zzzzzzz");
    }
    public void sayHello() {
        System.out.println("Purr");
    }
}
```

```
public class Tiger extends Cat{

    public void sayHello() {
        System.out.println("Roar");
    }
    public void swim() {
        System.out.println("Splash Splash");
    }
}
```

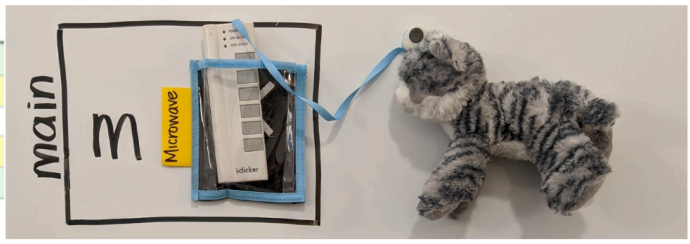
```
public class CartoonTiger extends Tiger {

    public void sayHello() {
        System.out.println("Hoo-hoo-hoo-hoo");
    }

}
```

```
public class Example3 {
    public static void main (String [] args) {
        CartoonTiger ct = new CartoonTiger();
        ct.sayHello();
        ct.swim();
        ct.sleep();
    }
}
```

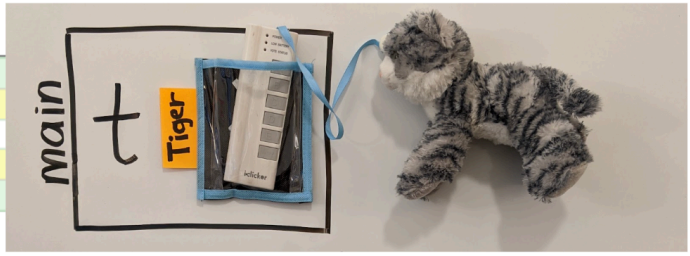


(9.6) Polymorphism Introduction: The type of the object must be the same class or a subclass of the type of the variable. If not, the type of the variable would imply that the object would have methods that it might not have (video)

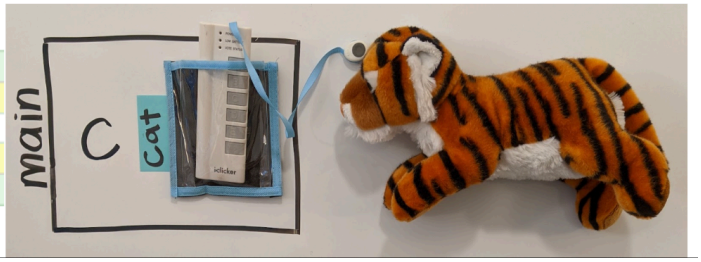
```
public class Example4 {
    public static void main (String [] args) {
        Microwave m = new Cat(); // Does not compile
    }
}
```




```
public class Example5 {
    public static void main (String [] args) {
        Tiger t = new Cat(); // Does not compile
    }
}
```

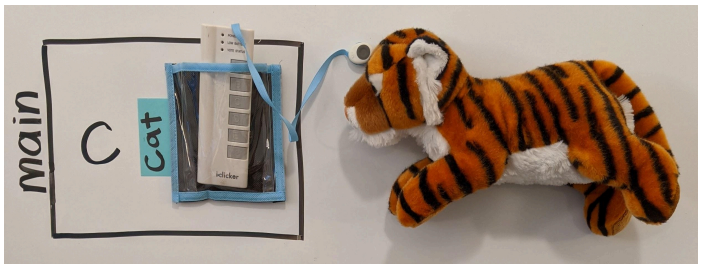



```
public class Example6 {
    public static void main (String [] args) {
        Cat c = new Tiger();
    }
}
```



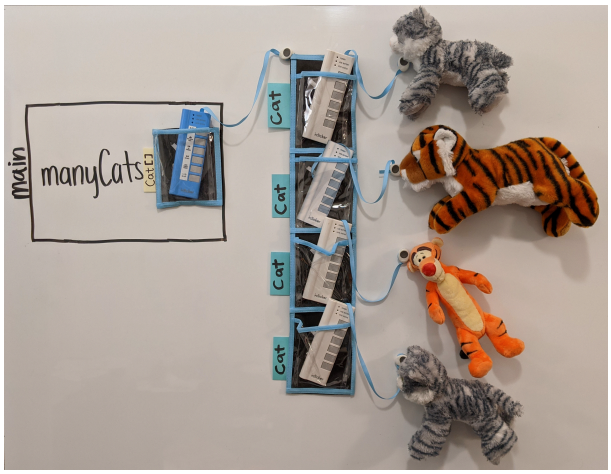
(9.7) Method Calls with Polymorphism: When we call a method on an object, we look in the object's class to find the code for the method (video).

```
public class Example7 {
    public static void main (String [] args) {
        Cat c = new Tiger();
        c.sayHello();
        c.sleep();
    }
}
```



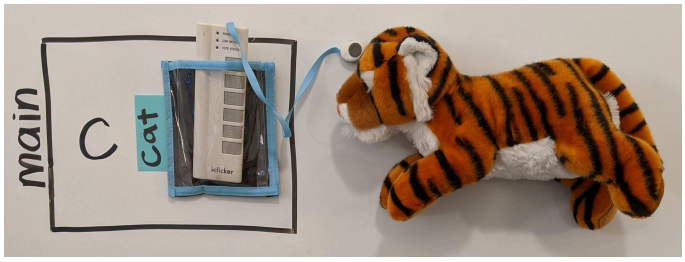
[9.8] Motivation for Polymorphism: When we call a method on an object, we look in the object's class to find the code for the method (video)

```
public class Example8 {
    public static void main (String [] args) {
        Cat[] manyCats = new Cat[4];
        manyCats[0] = new Cat();
        manyCats[1] = new Tiger();
        manyCats[2] = new CartoonTiger();
        manyCats[3] = new Cat();
        for (Cat c : manyCats) {
            c.sayHello();
        }
    }
}
```



[9.9] Compilation Rules for Methods: The type of the variable determines what methods can be called. The type of the object determines what method is called (video)

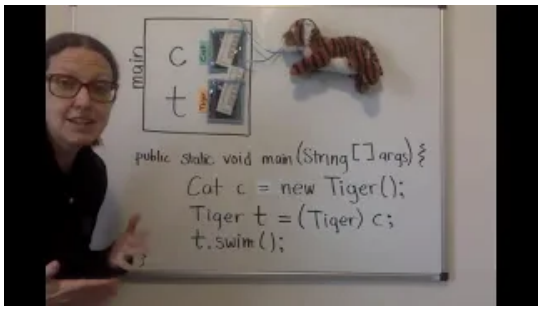
```
public class Example8 {
    public static void main (String [] args) {
        Cat c = new Tiger();
        c.swim(); // does not compile
    }
```



[9.10] Casting: Casting creates an additional reference to an object with the promised type (Video)

Note: Casting of reference variables is not tested on the AP CS A exam.

```
public class Example10 {
    public static void main (String [] args) {
        Cat c = new Tiger();
        Tiger t = (Tiger) c;
        t.swim();
    }
}
```



Unit 10 - Recursion