Physical Java Memory Models - Introduction

Colleen Lewis
Associate Professor of Computer Science
University of Illinois at Urbana-Champaign
ColleenL@illinois.edu
@CSTeachingTips

Hi friends! You should have access to this paper for free that describes my instructional sequence: https://doi.org/10.1145/3408877.3432477
And my updated videos
And I have new Videos on Inheritance
Videos made for the APCSA: CS Awesome curriculum and others

Introduction to the Models: Borrowing from math education, I have created physical models of Java memory. These may help students understand the abstract representation of Java programs -- much like blocks or cuisenaire rods help young children reason about abstract things like quantity, addition, and subtraction. These physical models of Java represent method calls, primitives, arrays, objects, and polymorphism.

Structure of this Document: I included six example programs. This isn't how I'd introduce this to students. These examples are purely designed for other educators who are familiar with Java and likely familiar with the difficulties students have reasoning about Java programs. You might also want to see the Example Videos.

Notes about the memory model: It doesn't represent class (i.e., static) variables or return values from methods.

Amazon list and construction instructions:
When in doubt, glue a magnet to the back. Gorilla glue might be helpful.

- Pockets to hold references to objects or arrays:
  - Cutting on the blue dividing stripes works well (I thought it might fray or look bad)
  - Keep some together to show an array of references (not shown in examples - video)
- Pockets to hold ints:
  - I'd use 3 squares to make a single int variable. On the outer squares, I'd cut the plastic part out so that I could fold the blue backing over on itself for the edges. I'd hot glue it.
  - Keep multiple squares together for a 1D or 2D array.
- Pockets to hold doubles:
  - I cut them to the width of a 3x5 card.
- Binary variables:
  - Red/Green magnets:
    - I think these are more visible than the switches, so I'm updating the models to use these.
  - Switches to represent binary variables:
    - I've sometimes found these $1 or $2 cheaper.
- Objects (stuffed animals) - Buy two sets because you need two that are identical!
  - Option 1: Small objects with magnets built in
  - Option 2: Stuffed animals without magnets
    - I can't find the ones I bought, but I'd buy these next time b/c they seem like they seem smaller and don't have legs that make it hard to attach the dog to the white board with a

magnet. It is nearly impossible to get these ducks to stick to a magnet and then stick to the whiteboard.

- Ribbon
- References:
    - Bookmarks
        - I glue a few together so that they feel a little sturdier.
    - Necklace Boxes for References (remote controls)
        - I hot glued the ribbon into the center of this and then tied the clips (below) to the end.
        - I also hot glued the box closed.
        - I also glued a magnet to the outside of the box so that the magnetic clip (below) could make the ribbon less tangled.
- Clips for References: (at the end of the ribbon)
    - Option 1: Use clothes pins (lots are available, I try to get a slightly smaller version).
    - Option 2: Clips with magnets built in

Example 1: Primitives & References to Arrays

I want to demonstrate that:

- A variable can store a value.
- A variable's type determines the type of value that can be stored.
- An `int` is smaller than a double.
- A `double` has a decimal.
- A `boolean` has only two states.
- An `int` array is just int variables in a sequence.
- A variable can reference an `int` array.
- If we don't set a variable that can reference an array, that variable will be `null`.
- A variable can reference a two-dimensional `int` array.
- A variable defined within a method is within scope in the method.

Some hard things that this tries to hit:

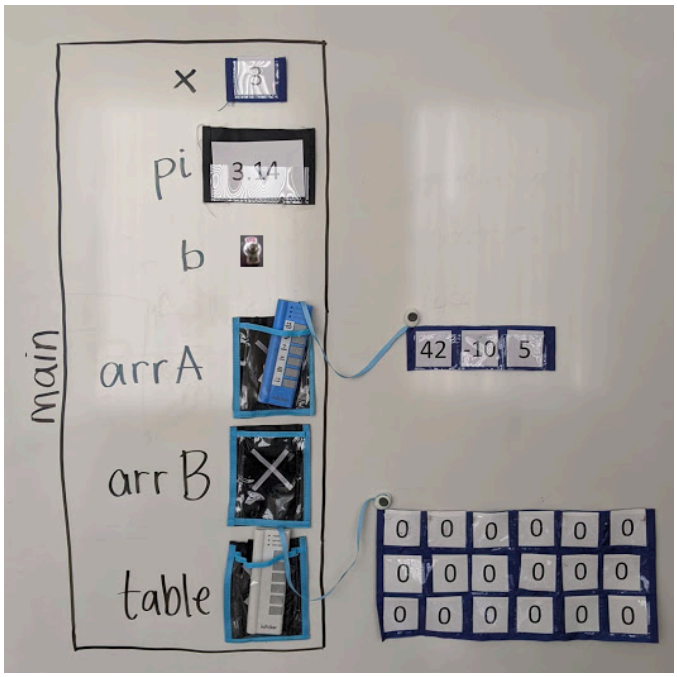
- Variables have types.
- `null`
- The difference between references and primitives.
- Preparation for seeing methods with different scope.

A few additional notes:

- When I start with these memory models, I don't represent the input args, but I do in the next example.
- References:
    - I represent references as a remote control with a ribbon to the referent. The remote control color doesn't matter.
    - I represent a null reference as a pocket without a remote control, which allows you to see the X glued to the back in the pocket.
    - I think it is really helpful to start with references to arrays because it gets around creating objects.

Example1.java

```
1  public class Example1 {
2      public static void main(String[] args){
3          int x = 3;
4          double pi = 3.14;
5          boolean b = false;
6          int[] arrA = {42, -10, 5};
7          int[] arrB;
8          int[][] table = new int[3][6];
9      }
10 }
```



Example 2: Calling a method with a reference to an array

I want to demonstrate that:

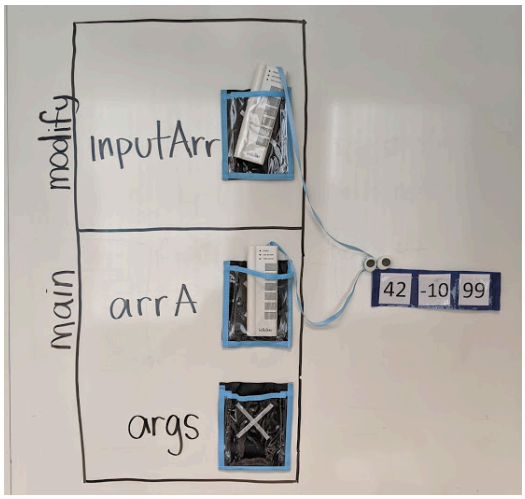
- Two references can reference the same thing.
- Any reference to something can modify it.
- That we get a copy of a reference when it is passed as an argument.
- A variable defined within a method is within scope in the method.
- We get a variable named args. We represent it as `null`, but it is likely a `0` length `String` array.

Some hard things that this tries to hit:

- All of this is hard!!!

Example2.java

```
1  public class Example2 {
2      public static void modify(int[] inputArr) {
3          inputArr[2] = 99;
4      }
5
6      public static void main(String[] args) {
7          int[] arrA = { 42, -10, 5 };
8          Example2.modify(arrA);
9      }
10 }
```



Example 3: References to Objects

I want to demonstrate that:

- A reference to an object is similar to a reference to an array.
- Two references can reference the same thing.
- Two references can reference *identical* things that are not the same thing.
- If we don't set a variable that can reference an object, that variable will be `null`.

Some hard things that this tries to hit:

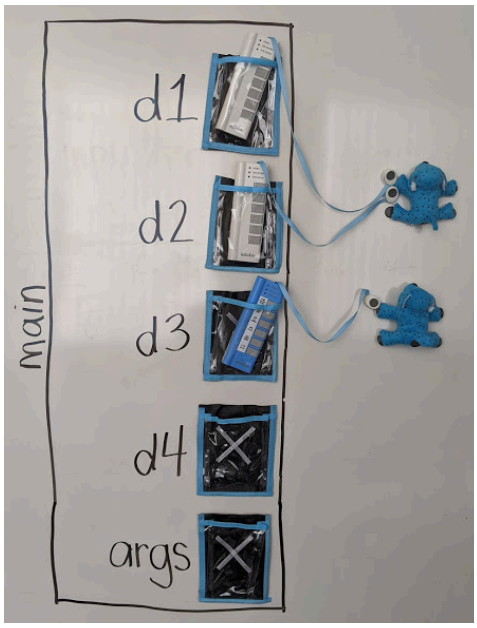
- `null`!
- I often use this to introduce the difference between == and `.equals`.

Example3.java

```
1 public class Example3 {
2     public static void main(String[] args) {
3         Dog d1 = new Dog();
4         Dog d2 = d1;
5         Dog d3 = new Dog();
6         Dog d4;
7     }
8 }
```

Dog.java

```
1 public class Dog {
2
3 }
```



Example 4: Non-static method calls

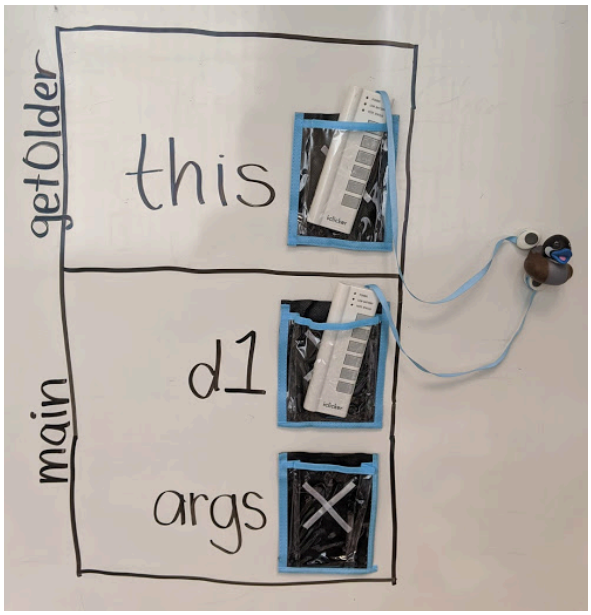
I want to demonstrate that:

- The variable `this` that we get in a non-static method call and references the thing the method was called on.

Some hard things that this tries to hit:

- The variable `this`

```
Duck.java
 1 public class Duck {
 2     private int myAge;
 3
 4     public void getOlder() {
 5         this.myAge++;
 6     }
 7
 8     public static void main(String[] args) {
 9         Duck d1 = new Duck();
10         d1.getOlder();
11     }
12 }
```



Example 5: Instance variables

I want to demonstrate that:

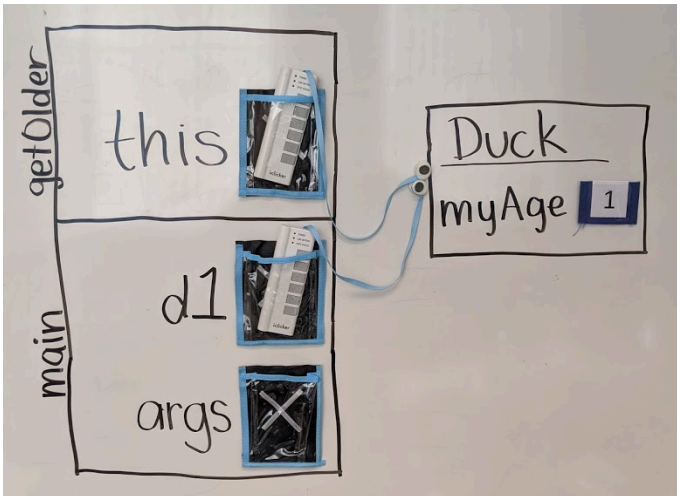
- That each object of a class gets a copy of the instance variables.

Some hard things that this tries to hit:

- The variable `this`
- The non-static method can modify the variable that the method was called on.

Same code as Example 4

```
Duck.java
 1 public class Duck {
 2     private int myAge;
 3
 4     public void getOlder() {
 5         this.myAge++;
 6     }
 7
 8     public static void main(String[] args) {
 9         Duck d1 = new Duck();
10         d1.getOlder();
11     }
12 }
```



Example 6: Polymorphism

I want to demonstrate that:

- A parent reference can reference an object of a child class.
- The type of the variable determines what methods *can be* called (the parent remote control only has methods in the parent class).
- The type of the object determines what method *is* called (the actual object looks in its class first to find the method! If it doesn't find it there - it looks in its parent's class).

Some hard things that this tries to hit:

- I use this to build intuition about what method will be called.
- Note - it might be confusing that we don't have a constructor, but Java provides a constructor with no arguments if you don't provide one.

Animal.java

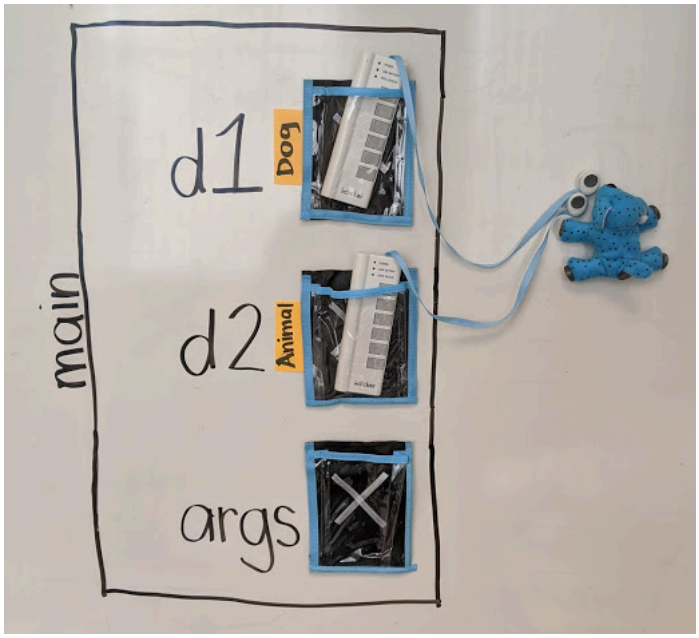
```
1 public class Animal {
2
3 }
```

Dog.java

```
1 public class Dog extends Animal {
2
3 }
```

Example5.java

```
1 public class Example5 {
2     public static void main(String[] args) {
3         Dog d1 = new Dog();
4         Animal d2 = d1;
5     }
6 }
```



Appendix

This is just a copy of the code in text form. :-)

Example 1 - Primitives & References to Arrays

```
public class Example1 {
	public static void main(String[] args){
		int x = 3;
		double pi = 3.14;
		boolean b = false;
		int[] arrA = {42, -10, 5};
		int[] arrB;
		int[][] table = new int[3][6];
	}
}
```

Example 2: Calling a method with a reference to an array

```
public class Example2 {
	public static void modify(int[] inputArr) {
		inputArr[2] = 99;
	}

	public static void main(String[] args) {
		int[] arrA = { 42, -10, 5 };
		Example2.modify(arrA);
	}
}
```

Example 3: References to Objects

```
public class Example3 {
@SuppressWarnings("unused")
	public static void main(String[] args) {
		Dog d1 = new Dog();
		Dog d2 = d1;
		Dog d3 = new Dog();
		Dog d4;
	}
}
```

Example 4: Non-static method calls & Example 5: Instance Variables

```
public class Duck {
@SuppressWarnings("unused")
	private int myAge;

	public void getOlder() {
		this.myAge++;
	}

	public static void main(String[] args) {
		Duck d1 = new Duck();
		d1.getOlder();
	}
```

```
}
```

Example 5: Polymorphism

```
public class Example5 {
      public static void main(String[] args) {
            Dog d1 = new Dog();
            Animal d2 = d1;
      }
}
public class Animal {
}
public class Dog extends Animal {
}
```