


Object Controlling


SAM




How to declare an Obj?

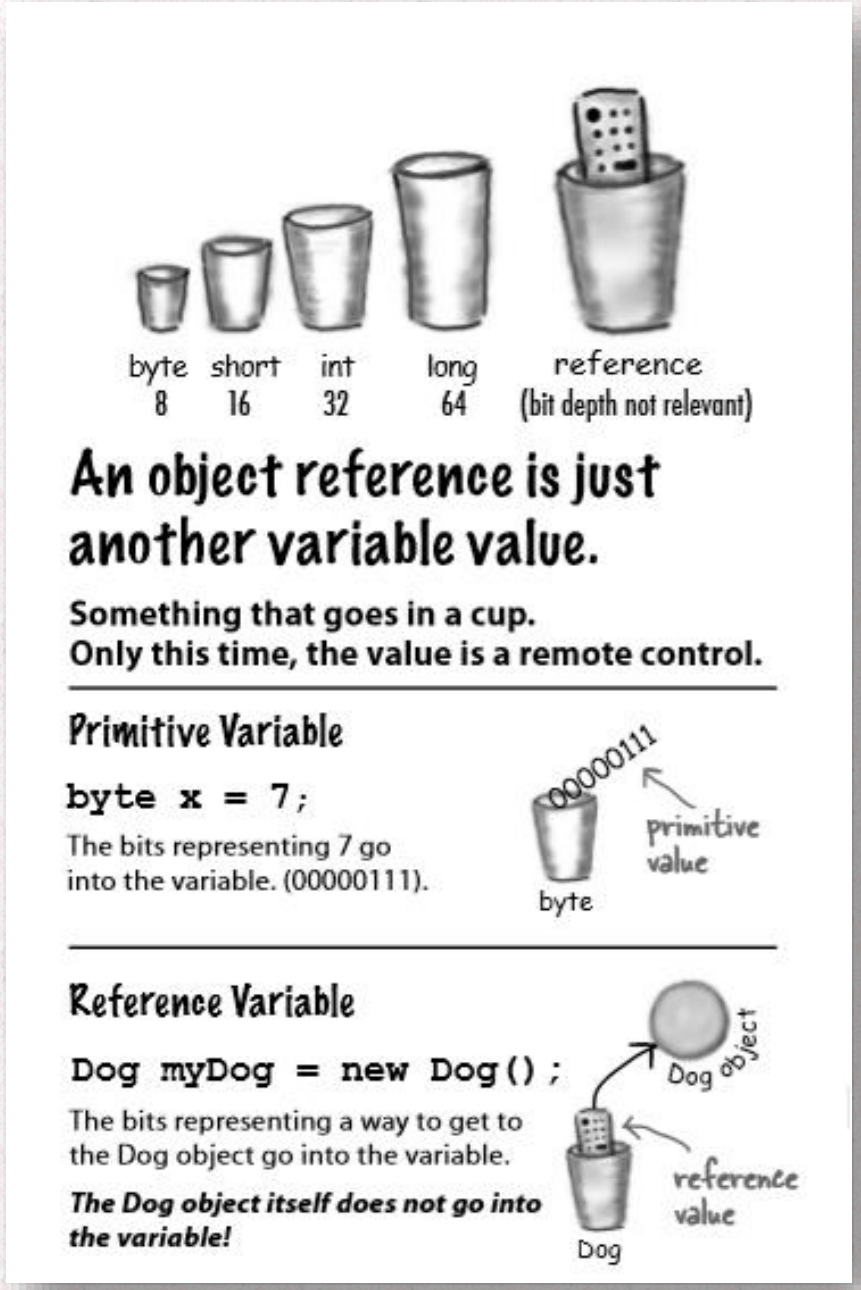
- There is actually no such thing as an object variable.
- There’s only an object reference variable.
- An object reference variable holds bits that represent a way to access an object.
- It doesn’t hold the object itself, but it holds something like a pointer. Or an address. Except, in Java we don’t really know what is inside a reference variable.
- We do know that whatever it is, it represents one and only one object. And the JVM knows how to use the reference to get to the object.

```
Dog d = new Dog();
d.bark();
```

think of this

like this

Think of a Dog reference variable as a Dog remote control. You use it to get the object to do something (invoke methods).



- With primitive variables, the value of the variable is... the value (5, -26.7, ‘a’).
- With reference variables, the value of the variable is... bits representing a way to get to a specific object.
- You don’t know (or care) how any particular JVM implements object references. Sure, they might be a pointer to a pointer to... but even if you know, you still can’t use the bits for anything other than accessing an object.

The 3 steps of object declaration, creation and assignment

```
1        3   2
Dog myDog = new Dog();
```

1 Declare a reference variable

```
Dog myDog = new Dog();
```

Tells the JVM to allocate space for a reference variable, and names that variable *myDog*. The reference variable is, forever, of type Dog. In other words, a remote control that has buttons to control a Dog, but not a Cat or a Button or a Socket.

2 Create an object

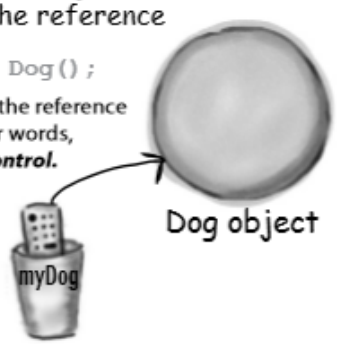
```
Dog myDog = new Dog();
```

Tells the JVM to allocate space for a new Dog object on the heap (we'll learn a lot more about that process, especially in chapter 9.)

3 Link the object and the reference

```
Dog myDog = new Dog();
```

Assigns the new Dog to the reference variable myDog. In other words, ***programs the remote control.***



How big is a reference variable?

We don't know.

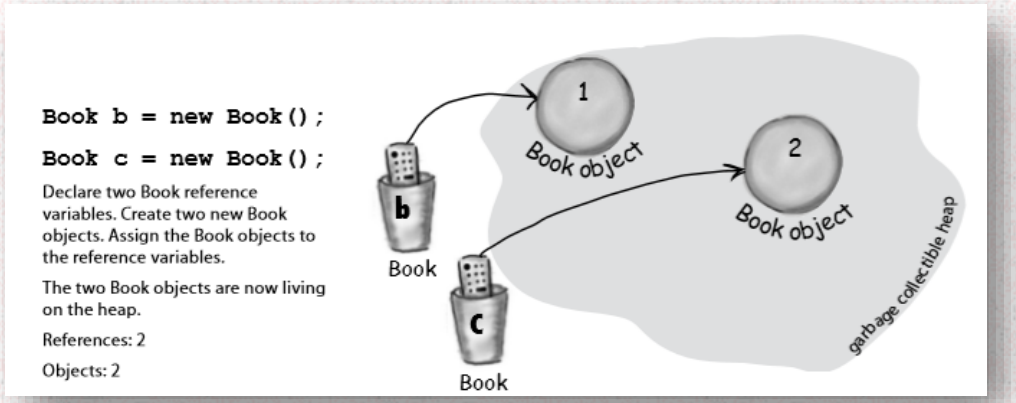
So, does that mean that all object references are the same size, regardless of the size of the actual objects to which they refer?

Yep. All references for a given JVM will be the same size regardless of the objects they reference, but each JVM might have a different way of representing references, so references on one JVM may be smaller or larger than references on another JVM.

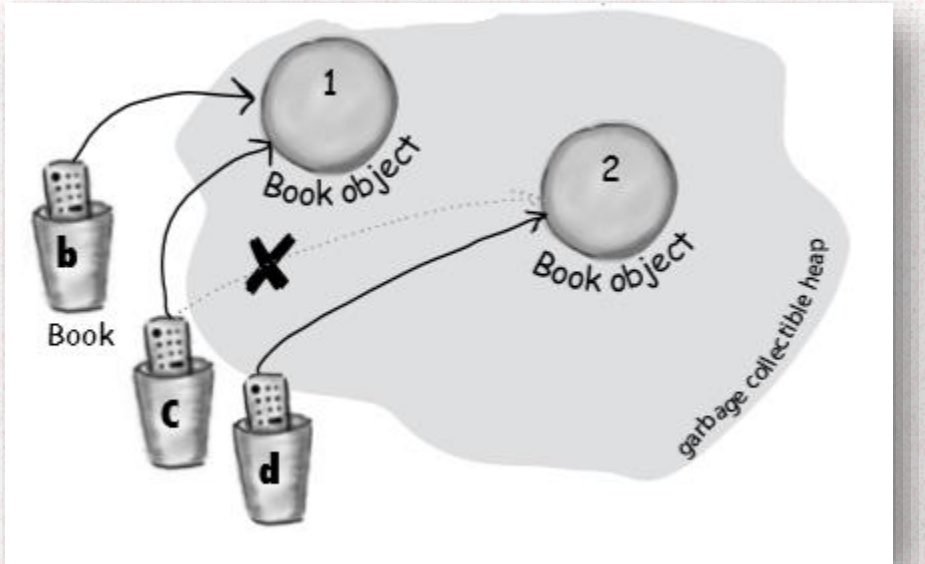
Can I do arithmetic on a reference variable, increment it, you know – C stuff?

Nope. "Java is not C."

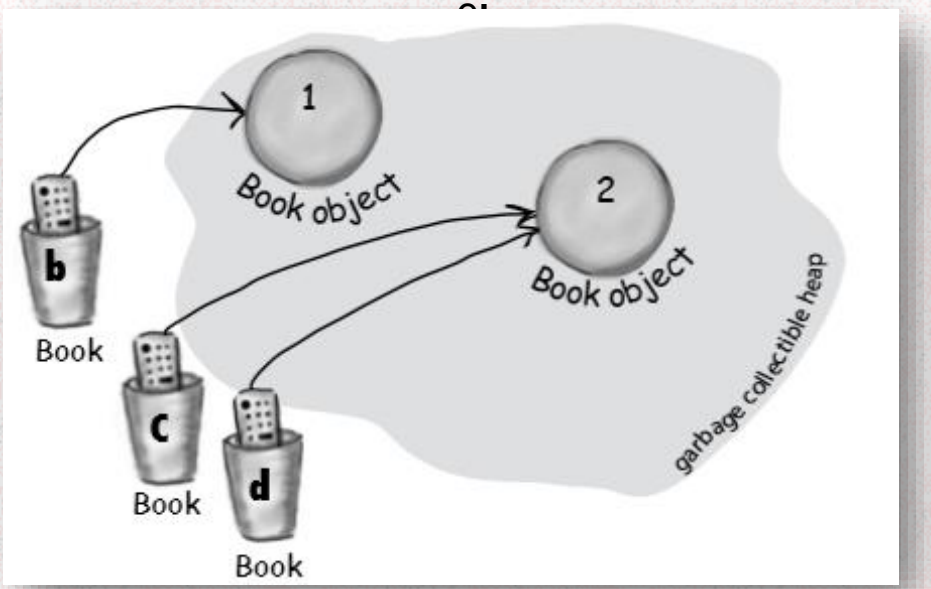
Life on the garbage-collectible heap



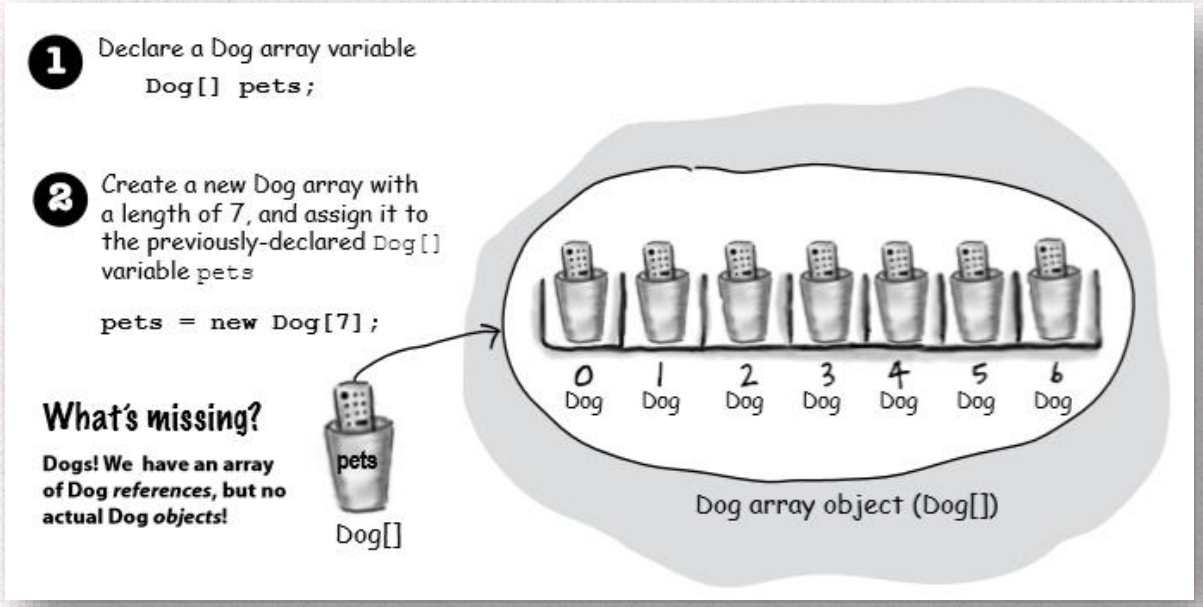
c = b;

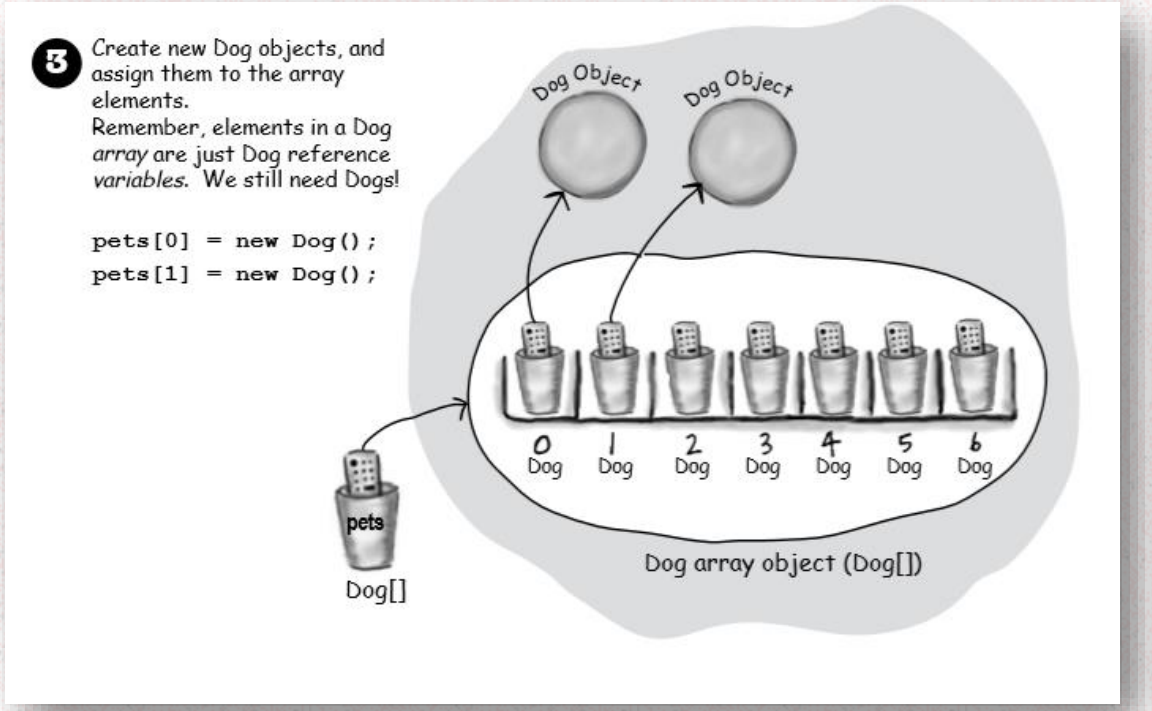


Book d =



Object array





Let do programming


```
package sam.myclassproject;


public class Person {
    String name;
    int age;

    Person()
    {
    }

    Person(String name, int age)
    {
        this.name=name;
        this.age=age;
    }
}
```

```
public class MyclassProject {


    public static void main(String[] args) {
        //System.out.println("Hello World! with me");
        Person[] p = new Person[4];


        p[0]=new Person("shamim", 45);
        p[1]=new Person("Karim", 45);
        p[2]=new Person("Rahim", 45);
        p[3]=new Person();


        for(int i=0;i<p.length;i++)
        {
            System.out.println(p[i].name);
            System.out.println(p[i].hashCode());
            System.out.println(p[i].toString());
            System.out.println(Integer.toHexString(p[i].hashCode()));



        }


        System.out.println(p[3].name);
        System.out.println(p[3]);


        for(Person obj : p){
            System.out.println(obj.name);
        }
    }
}
```

```
shamim
1523554304
sam.myclassproject.Person@5acf9800
5acf9800
Karim
1175962212
sam.myclassproject.Person@4617c264
4617c264
Rahim
918221580
sam.myclassproject.Person@36baf30c
36baf30c
null
2055281021
sam.myclassproject.Person@7a81197d
7a81197d
null
sam.myclassproject.Person@7a81197d
shamim
Karim
Rahim
null
```

BULLET POINTS

- Variables come in two flavors: primitive and reference.
- Variables must always be declared with a name and a type.
- A primitive variable value is the bits representing the value (5, 'a', true, 3.1416, etc.).
- A reference variable value is the bits representing a way to get to an object on the heap.
- A reference variable is like a remote control. Using the dot operator (.) on a reference variable is like pressing a button on the remote control to access a method or instance variable.
- A reference variable has a value of `null` when it is not referencing any object.
- An array is always an object, even if the array is declared to hold primitives. There is no such thing as a primitive array, only an array that *holds* primitives.

Class Task: A short Java program is listed to the right. When '// do stuff' is reached, some objects and some reference variables will have been created. Your task is to determine which of the reference variables refer to which objects. Not all the reference variables will be used, and some objects might be referred to more than once. Draw lines connecting the reference variables with their matching objects.

```
class HeapQuiz {
  int id = 0;
  public static void main(String [] args) {
    int x = 0;
    HeapQuiz [ ] hq = new HeapQuiz[5];
    while ( x < 3 ) {
      hq[x] = new HeapQuiz();
      hq[x].id = x;
      x = x + 1;
    }
    hq[3] = hq[1];
    hq[4] = hq[1];
    hq[3] = null;
    hq[4] = hq[0];
    hq[0] = hq[3];
    hq[3] = hq[2];
    hq[2] = hq[0];
    // do stuff
  }
}
```