Course: Object Based Modeling
Code: CS-33105
Branch: MCA-3

Lecture -3

Dr. J Sathish Kumar (JSK)
(Faculty & Coordinator)

Department of Computer Science and Engineering

Motilal Nehru National Institute of Technology Allahabad,
Prayagraj-211004

Class Exercise #1

A

```
class TapeDeck {

  boolean canRecord = false;

  void playTape() {
    System.out.println("tape playing");
  }

  void recordTape() {
    System.out.println("tape recording");
  }
}

class TapeDeckTestDrive {
  public static void main(String [] args) {

    t.canRecord = true;
    t.playTape();

    if (t.canRecord == true) {
      t.recordTape();
    }
  }
```

BE the compiler

B

```
class DVDPlayer {

  boolean canRecord = false;

  void recordDVD() {
    System.out.println("DVD recording");
  }
}

class DVDPlayerTestDrive {
  public static void main(String [] args) {

    DVDPlayer d = new DVDPlayer();
    d.canRecord = true;
    d.playDVD();

    if (d.canRecord == true) {
      d.recordDVD();
    }
  }
}
```

A

```
class TapeDeck {
  boolean canRecord = false;
  void playTape() {
    System.out.println("tape playing");
  }
  void recordTape() {
    System.out.println("tape recording");
  }
}

class TapeDeckTestDrive {
  public static void main(String [] args) {

    TapeDeck t = new TapeDeck( );
    t.canRecord = true;
    t.playTape();

    if (t.canRecord == true) {
      t.recordTape();
    }
  }
}
```

We've got the template, now we have to make an object !

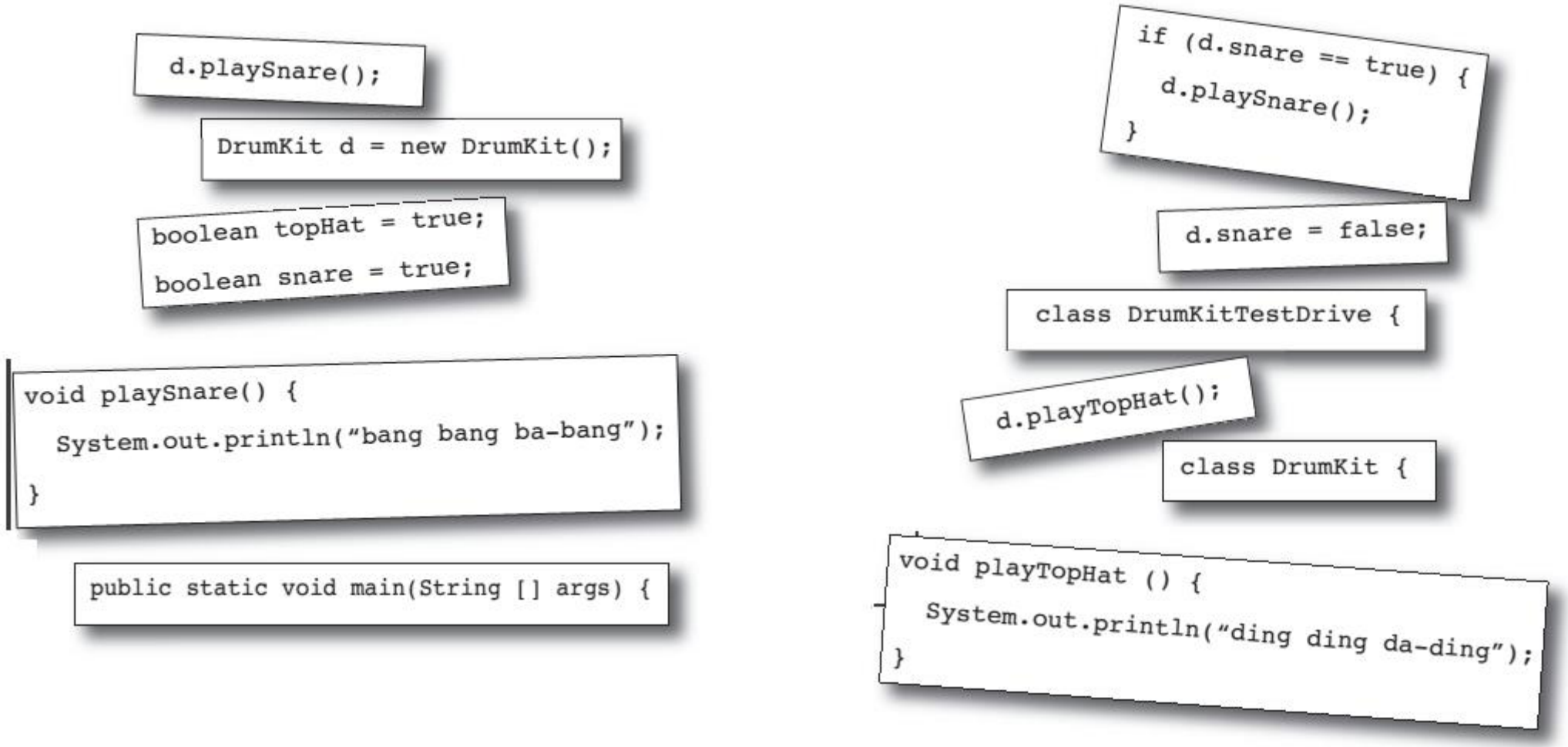
B

```
class DVDPlayer {
  boolean canRecord = false;
  void recordDVD() {
    System.out.println("DVD recording");
  }
  void playDVD ( ) {
    System.out.println("DVD playing");
  }
}

class DVDPlayerTestDrive {
  public static void main(String [] args) {
    DVDPlayer d = new DVDPlayer();
    d.canRecord = true;
    d.playDVD();
    if (d.canRecord == true) {
      d.recordDVD();
    }
  }
}
```

The line: d.playDVD( ); wouldn't compile without a method !

Class Exercise #2



```
class DrumKit {

  boolean topHat = true;
  boolean snare = true;

  void playTopHat() {
    System.out.println("ding ding da-ding");
  }

  void playSnare() {
    System.out.println("bang bang ba-bang");
  }
}
```

```
class DrumKitTestDrive {
  public static void main(String [] args) {

    DrumKit d = new DrumKit();
    d.playSnare();
    d.snare = false;
    d.playTopHat();

    if (d.snare == true) {
      d.playSnare();
    }
  }
}
```

```
public class EchoTestDrive {
  public static void main(String [] args) {
    Echo e1 = new Echo();

    ____________________________

    int x = 0;

    while ( ____________ ) {
      e1.hello();

      ____________________________

      if ( _____________ ) {
        e2.count = e2.count + 1;
      }
      if ( _____________ ) {
        e2.count = e2.count + e1.count;
      }
      x = x + 1;
    }
    System.out.println(e2.count);
  }
}
```

Class Exercise #3

```
class _____________ {
  int _________ = 0;
  void ____________ {
    System.out.println("helloooo... ");
  }
}
```

```
public class EchoTestDrive {
  public static void main(String [] args) {
    Echo e1 = new Echo();
    Echo e2 = new Echo( );  // the correct answer
          - or -
    Echo e2 = e1;   // is the bonus answer!
    int x = 0;
    while ( x < 4 ) {
      e1.hello();
      e1.count = e1.count + 1;
      if ( x == 3 ) {
        e2.count = e2.count + 1;
      }
      if ( x > 0 ) {
        e2.count = e2.count + e1.count;
      }
      x = x + 1;
    }
    System.out.println(e2.count);
  }
}
```

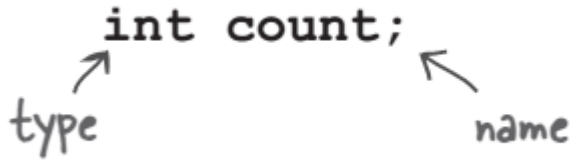
```
class Echo {
  int count = 0;
  void hello( ) {
    System.out.println("helloooo... ");
  }
}
```

Install Java in Ubuntu

- Update the repositories:
  - sudo apt-get update
- Install OpenJDK:
  - sudo apt-get install openjdk-8-jdk
- Verify the version of the JDK:
  - java –version
- Compile Java program
  - javac program.java
- Run/ Execute the program
  - java classname

Declaring a variable

- Variables come in two flavors: *primitive* and *object reference*.
- Primitives hold fundamental values (think: simple bit patterns) including integers, booleans, and floating point numbers.
- Object references hold, well, *references* to *objects.*



boolean and char

boolean	(JVM-specific) ***true*** or ***false***	
char	16 bits	0 to 65535

numeric (all are signed)

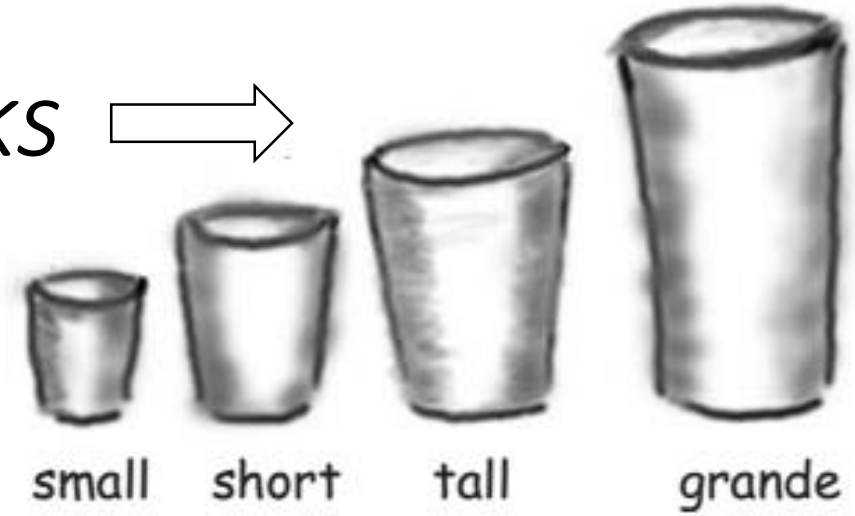
integer

byte	8 bits	-128 to 127
short	16 bits	-32768 to 32767
int	32 bits	-2147483648 to 2147483647
long	64 bits	-huge to huge

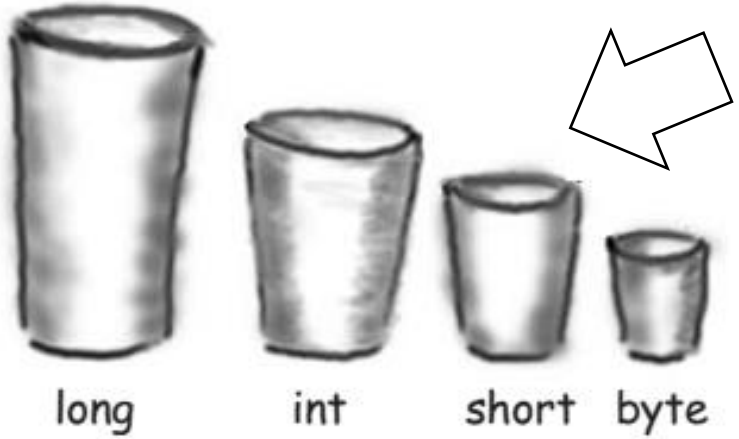
floating point

float	32 bits	varies
double	64 bits	varies

An Illustrative Example: Starbucks



Java



The eight primitive types are:

boolean **c**har **b**yte **s**hort **i**nt **l**ong **f**loat **d**ouble

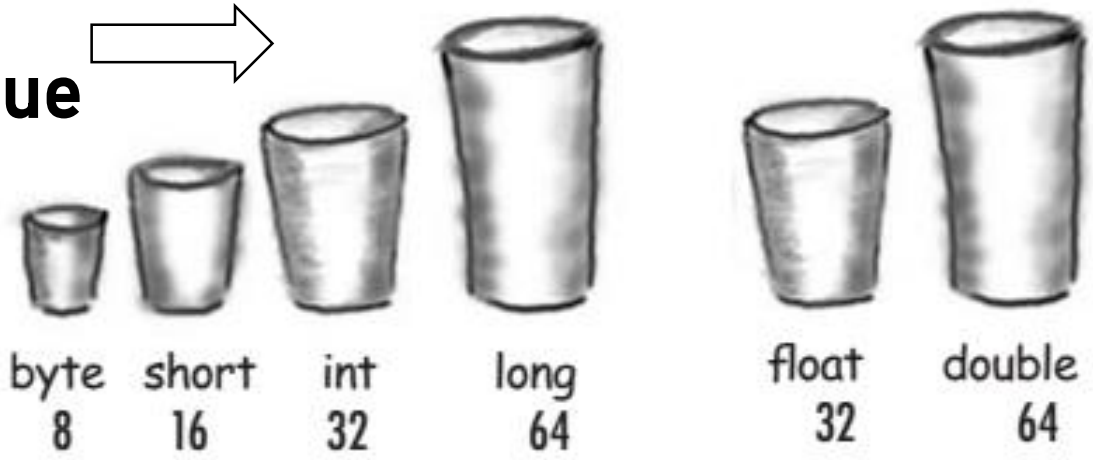
And here's a mnemonic for remembering them:

Be **C**areful! **B**ears **S**houldn't **I**ngest **L**arge **F**urry **D**ogs

If you make up your own, it'll stick even better.

B_ C_ B_ S_ I_ L_ F_ D_

Each Cup holds Value



Primitive declarations with assignments:

```
int x;
x = 234;
byte b = 89;
boolean isFun = true;
double d = 3456.98;
char c = 'f';
int z = x;
```

```
boolean isPunkRock;
isPunkRock = false;
boolean powerOn;
powerOn = isFun;
long big = 3456789;
float f = 32.5f;
```

Note the 'f'. Gotta have that with a float, because Java thinks anything with a floating point is a double, unless you use 'f'.

Be sure the value can fit into the variable.

You can't put a large value into a small cup! Well, OK, you can, but you'll lose some.

For example, you can't pour an int-full of stuff into a byte-sized container, as follows:

```
int x = 24;
byte b = x;
//won't work!!
```

The compiler cares about is that you're trying to put a big thing into a small thing, and there's the *possibility* of spilling.

The compiler always errs on the side of safety.

Keyword!

- It must start with a letter, underscore (_), or dollar sign ($).
- You can’t start a name with a number.
- After the first character, you can use numbers as well.
- It can be anything you like, subject to those two rules,
  - just so long as it isn’t one of Java’s reserved words.

This table reserved.

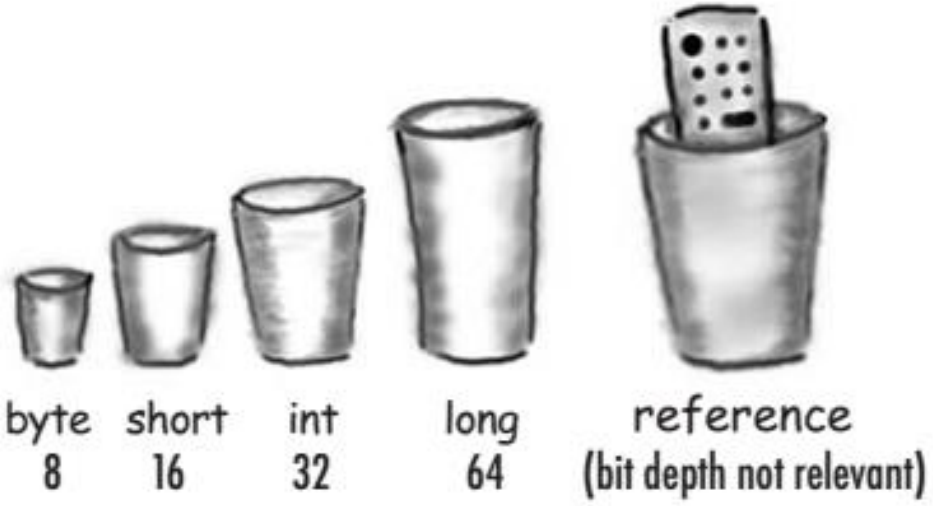
boolean	byte	char	double	float	int	long	short	public	private
protected	abstract	final	native	static	strictfp	synchronized	transient	volatile	if
else	do	while	switch	case	default	for	break	continue	assert
class	extends	implements	import	instanceof	interface	new	package	super	this
catch	finally	try	throw	throws	return	void	const	goto	enum

Java’s keywords and other reserved words (in no useful order). If you use these for names, the compiler will be very, *very* upset.

Object reference

- 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.
- A primitive variable is full of bits representing the actual ***value*** of the variable, an object reference variable is full of bits representing ***a way to get to the object.***
- You use the dot operator (.)
  - **myDog.bark();**

An Illustrative Example

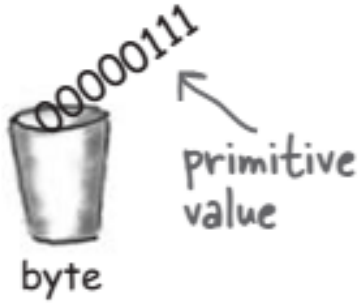


- An object reference is just another variable value.
- Something that goes in a cup. Only this time, the value is a remote control.

Primitive Variable

```
byte x = 7;
```

The bits representing 7 go into the variable. (00000111).

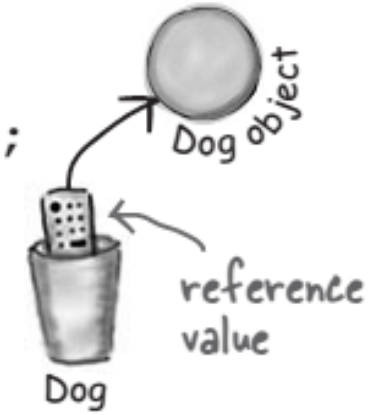


Reference Variable

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

The bits representing a way to get to the Dog object go into the variable.

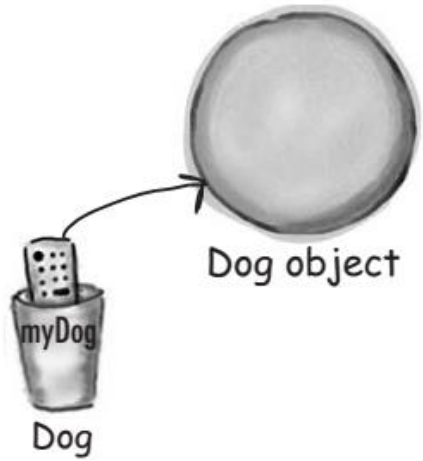
The Dog object itself does not go into the variable!



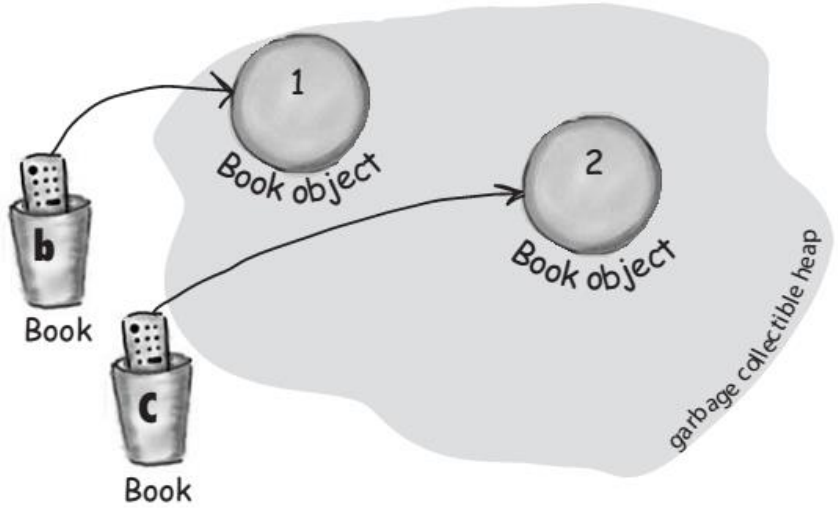
The 3 steps of object declaration, creation and assignment

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

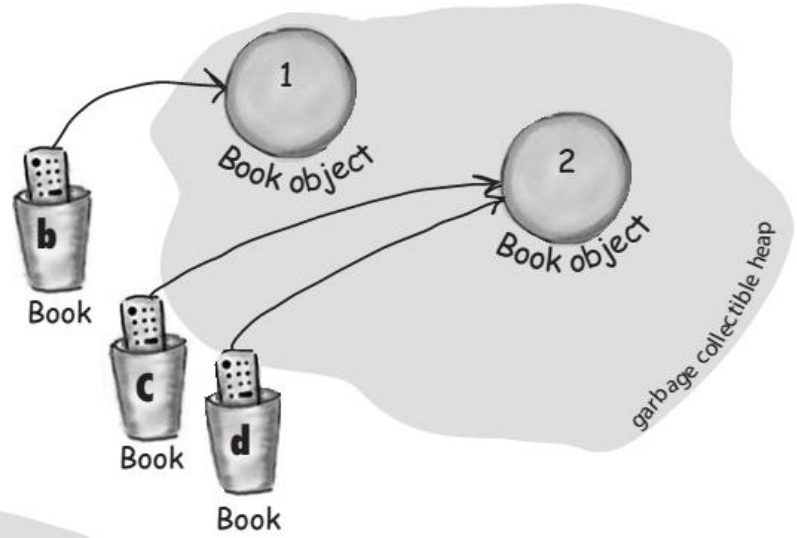
1. Declare a reference variable
   - **Dog myDog** = new Dog();
   - Tells the JVM to allocate space for a reference variable, and names that variable *myDog*.
2. Create an object
   - Dog myDog = **new Dog();**
   - Tells the JVM to allocate space for a new Dog object on the heap
3. Link the object and the reference
   - Dog myDog **=** new Dog();
   - Assigns the new Dog to the reference variable myDog. In other words,



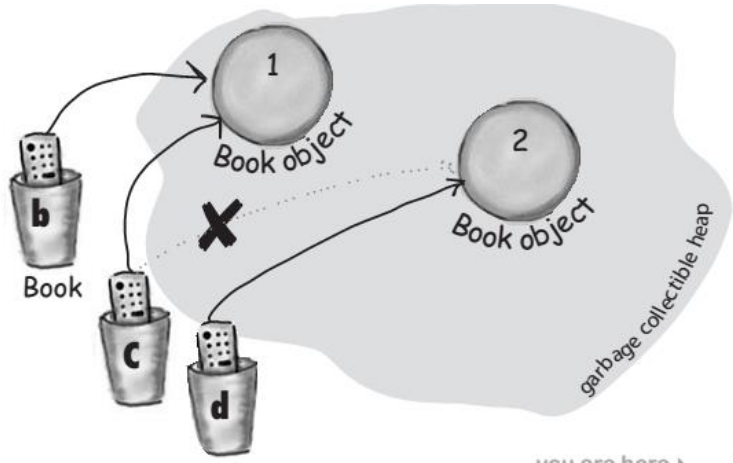
Objects!!



- **Book b = new Book();**
- **Book c = new Book();**
  - References: 2, Objects: 2



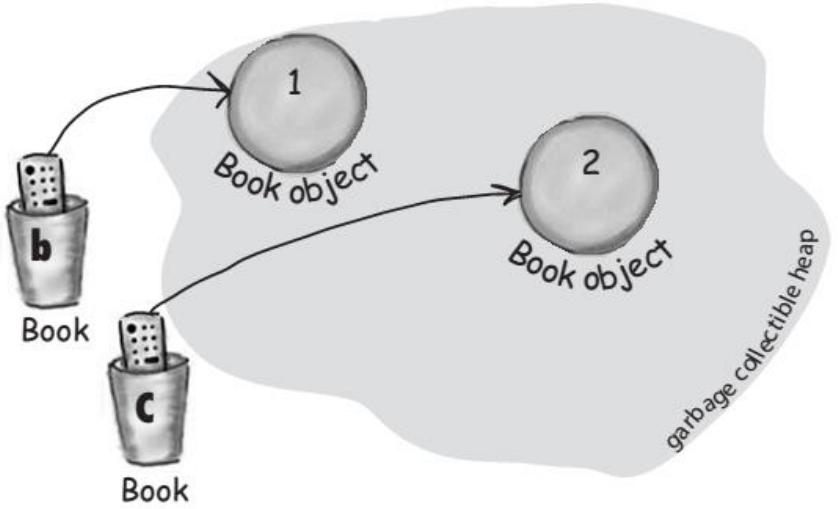
- **Book d = c;**
  - Both c and d refer to the same object.
  - The c and d variables hold two different copies of the same value. Two remotes programmed to one TV.
  - References: 3, Objects: 2



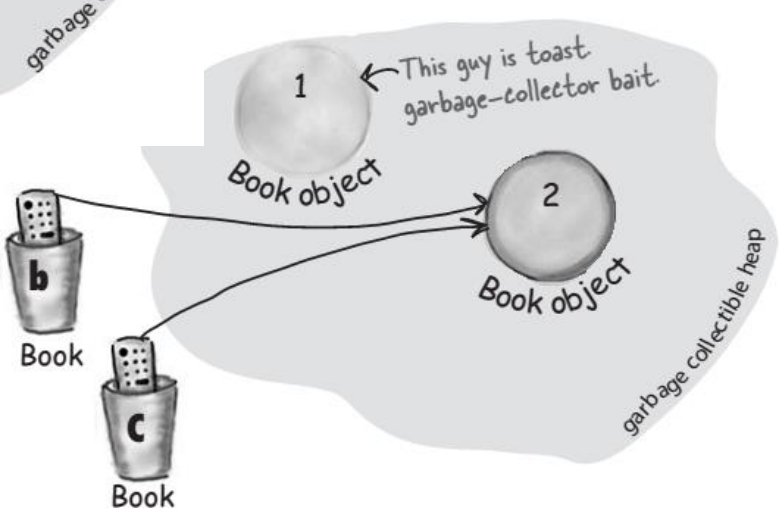
- **c = b;**
  - Both b and c refer to the same object.
  - References: 3, Objects: 2

More Objects!!

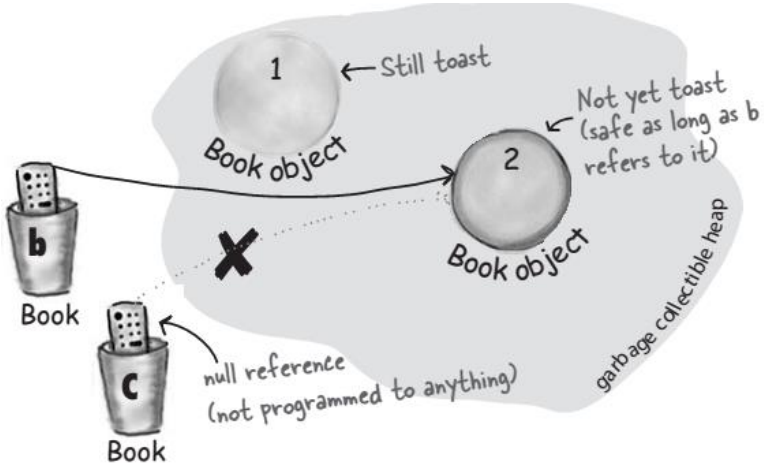
- **Book b = new Book();**
- **Book c = new Book();**
  - References: 2, Objects: 2



- **b = c;**
  - Both b and c refer to the same object. Object 1 is abandoned and eligible for Garbage Collection (GC).
    Active References: 2, Reachable Objects: 1
    Abandoned Objects: 1



- **c = null;**
  - Object 2 still has an active reference (b), and as long as it does, the object is not eligible for GC.
    Active References: 1, *null* References: 1
    Reachable Objects: 1, Abandoned Objects: 1

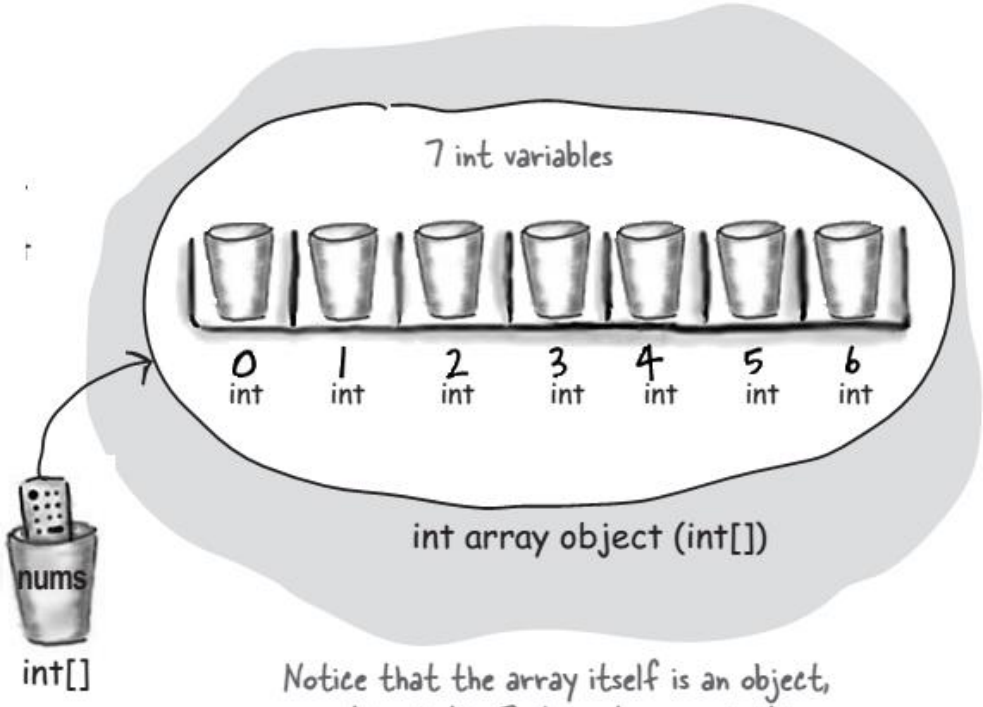


An Array

- Declare an int array variable.
  - ***int[] nums;***
- Create a new int array with a length of 7, and assign it to the previously declared int[] variable nums
  - **nums = new int[7];**
- Give each element in the array an int value.
  -

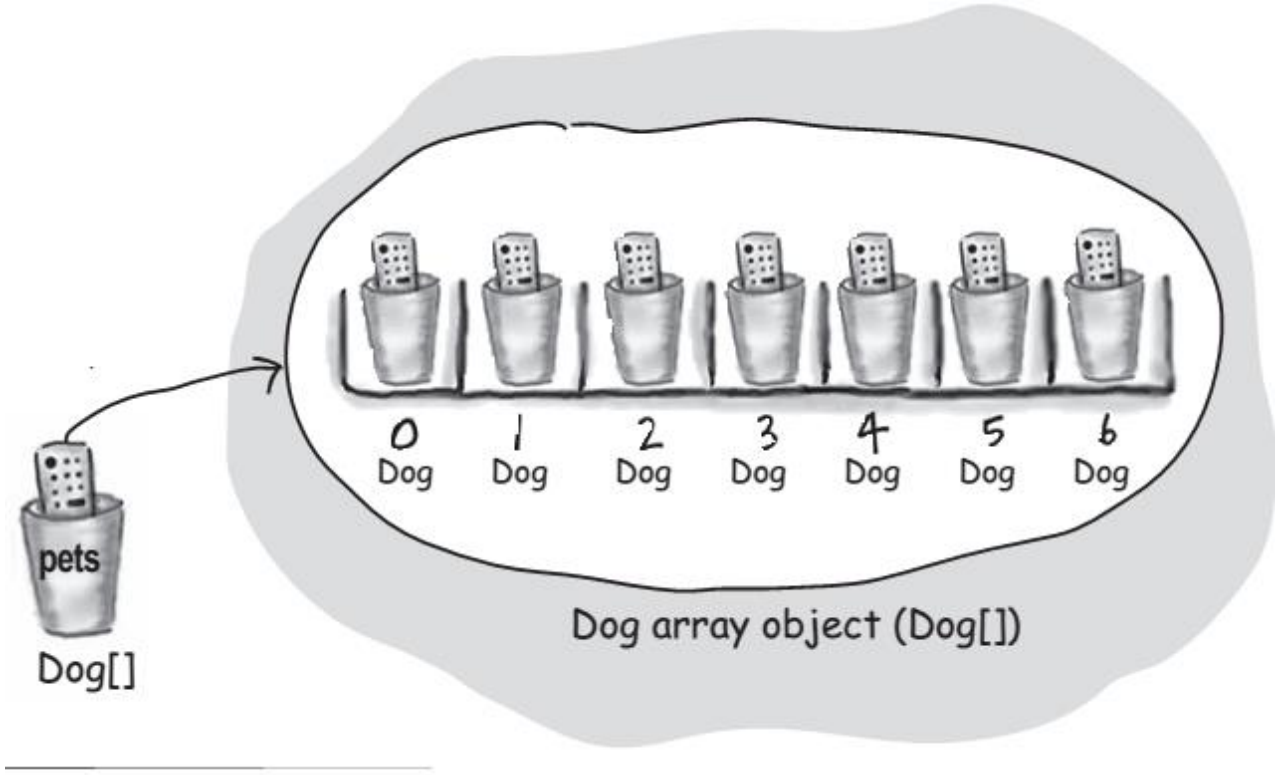
7 int variables

```
nums[0] = 6;
nums[1] = 19;
nums[2] = 44;
nums[3] = 42;
nums[4] = 10;
nums[5] = 20;
nums[6] = 1;
```



Arrays are objects too

- *Arrays are always objects, whether they're declared to hold primitives or object references.*
- Declare a Dog array variable
  - **Dog[] pets;**
- Create a new Dog array with a length of 7, and assign it to the previously-declared Dog[] variable pets
  - **pets = new Dog[7];**
- **We have an array of Dog *references*, but no actual Dog *objects*!**



Arrays are objects too

- Create new Dog objects, and assign them to the array elements.
  - **pets[0] = new Dog();**
  - **pets[1] = new Dog();**
- Example
  -

```
Dog[] myDogs = new Dog[3];
myDogs[0] = new Dog();
myDogs[0].name = "Fido";
myDogs[0].bark();
```