Object Oriented Programming
Part I of II

Steve Ryder
Session 8351
JSR Systems (JSR)
sryder@jsrsys.com

Objectives

- Compare/Contrast OO Programming to Procedural Programming
- Introduction to these Object Oriented concepts:
  - Classes
  - Objects
  - Class Data
  - Methods
- Understand the lifecycle of an object

Shape Shifter Program

- Specifications
  - Shapes on a GUI
    - Square
    - Circle
    - Triangle
  - When user clicks on shape
    - Shape will rotate clockwise 360 degrees
    - An AIF sound file specific to that shape will play

Procedural Design

- Write Important procedures

```
rotate(shapenum) {
//make the shape rotate 360 degrees
}

playSound(shapenum){
//use shapeNum to lookup which
//AIF sound to play, and play it
}
```

Object Oriented Design

- Write a class for each of the shapes

Square	Circle	Triangle

rotate() {
//code to rotate square
}
 	
playSound(){
//code to play AIF
//for a square
}
	rotate() {
	//code to rotate circle
	}
	
	playSound(){
	//code to play AIF
	//for a circle
	}
		rotate() {
		//code to rotate
		// triangle
		}
		
		playSound(){
		//code to play AIF
		//for a triangle
		}

A Specification Change

- Add amoeba shape
- When user clicks on amoeba
  - Shape will rotate
  - An .hif sound file will play

Procedural Design

- Change previously-test code
  - Rotate procedure will work as-is
  - PlaySound procedure must change

```
playSound(shapenum) {
//if the shape is not an amoeba,
     //use shapenum to look up the AIF
//else
    //play amoeba .hif sound
}
```

Object Oriented Design

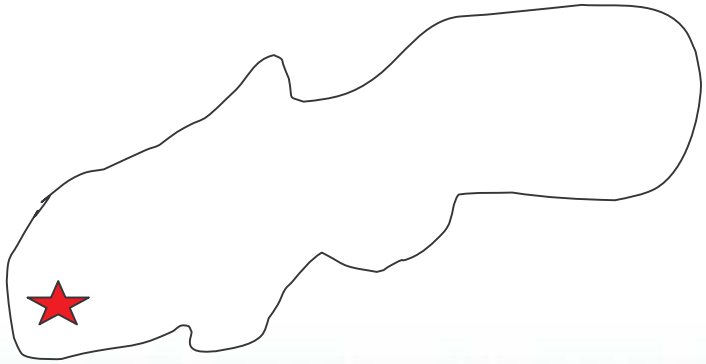
- Write one new class
- No need to touch previously-tested code

Amoeba

rotate() {
//code to rotate
// amoeba
}
 	
playSound(){
//code to play .hif
//for a amoeba
}

User Testing – Another Change

- All of the shapes rotated around the center of the shape.
- The amoeba shape, however, should rotate around a point at one end. Like this:



Procedural Design

- Add rotation point arguments to the rotate procedure
- A lot of code was affected

```
Rotate(shapenum, xPt, yPt) {
//if the shape is not an amoeba
      //calculate the center then rotate
//else
      //us the xPt and yPt as the
      //rotation point then rotate

}
```

Object Oriented Design

- Change rotate only in the amoeba class

Amoeba

int xPoint
int yPoint
 	
rotate() {
//code to rotate //amoeba using
//x and y coordinates
}
 	
playSound(){
//code to play .hif
//for a amoeba
}

Object Oriented Design concepts

- Class
- Object
- Method
- Class Data

Finding Classes

- Look for nouns in the specification
  - "Customers phone in and place an order for one or more items. The customer service representative creates a new order and adds the items to it. Next the shipping address and payment details are taken so that the order can be shipped and the customer's account charged."
  - Customer
  - Order
  - Item
  - Can you find others?

Objects

- What is the difference between a class and an object?
  - A class is not an object but…
  - It is used to construct them
- A class is a blueprint for an object
  - It explains *how* to make an object of that type
  - Each object made from that class can have its own instance variables

Objects

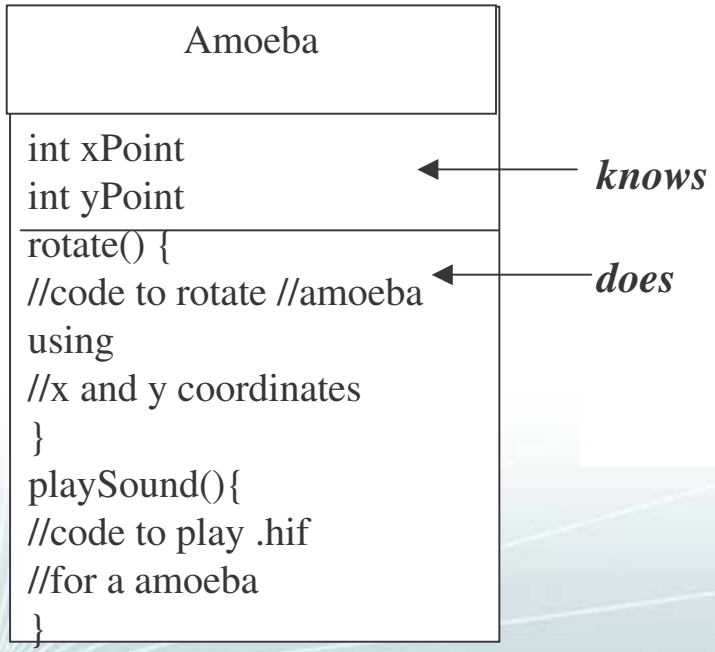
Think of an object like a pack of blank Rolodex™ cards.

- Each card has the same instance variables (blank fields)
- A completed card creates an instance of an object (a contact)
- The specific entries on each line represent the object's state (name, phone, address)

Class Data and Methods

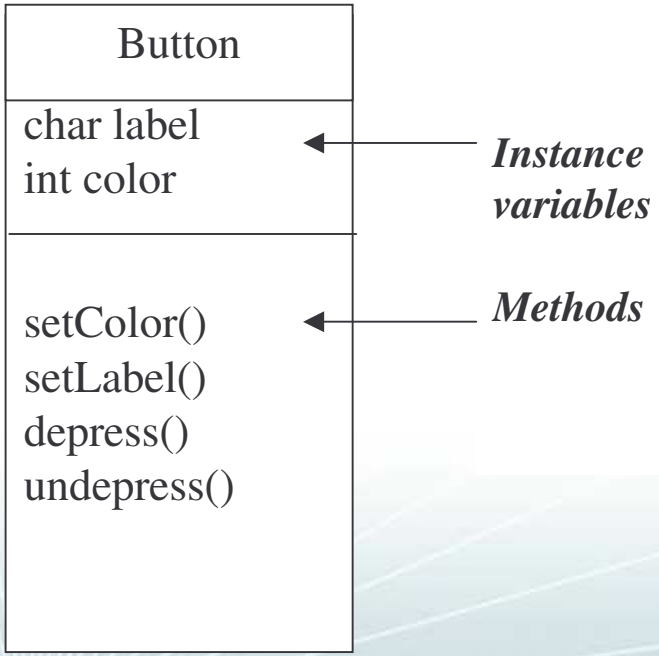
When you design a class, you think about the objects that will be created from that class. You think about:

- Things the object **knows**
- Things the object **does**



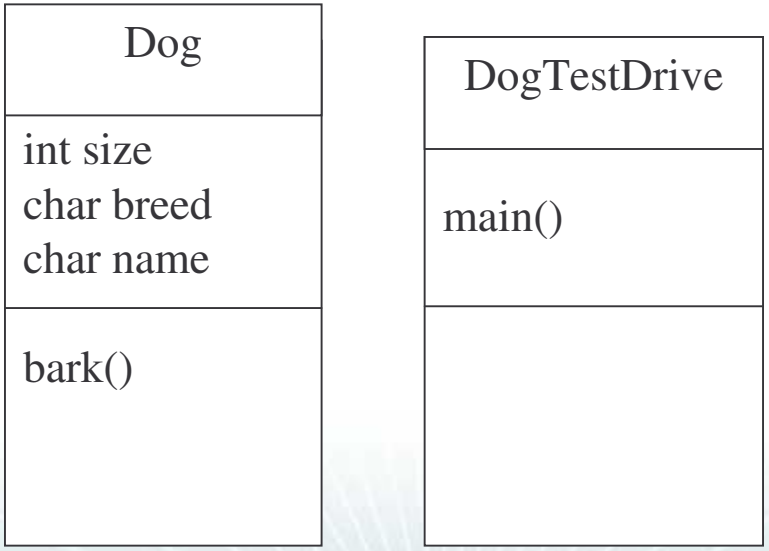
Class Data and Methods

- Things an object knows about itself are called
  - Instance variables
- Things an object can do are called
  - Methods



Your First Object

- What does it take to create and use an object?
  - You need two classes
    - One for the type of object you want to use
    - One to test your new class



Write the Dog class

```
class Dog {
    int size;
    String breed;
    String name;

    void bark() {
        System.out.println("Ruff! Ruff!");
    }

}
```

Write the DogTestDrive class

```
class DogTestDrive {
    public static void main ( String [] args) {
        Dog d = new Dog();
        d.size = 40;
        d.bark();
    }

}
```

The Behavior of an Object

- Instance variables affect method behavior
  - Every instance of a particular class has the same methods
  - But, the methods can behave differently based on the value of the instance variables.

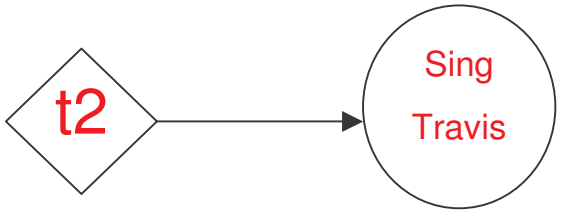
The Song class

- Two instance variables: title and artist.
- Methods to set the title and artist
- A method to play a song

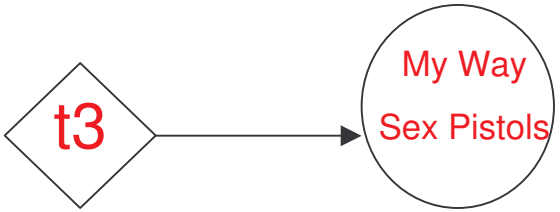
Song
String title String artist
setTitle() setArtist() play()

The Song class

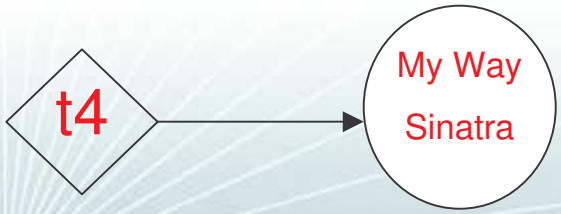
```
Song t2 = new Song();
t2.setArtist("Travis");
t2.setTitle("Sing");
```




```
Song t3 = new Song();
t3.setArtist("Sex Pistols");
t3.setTitle("My Way");
```




```
Song t4 = new Song();
t4.setArtist("Sinatra");
```



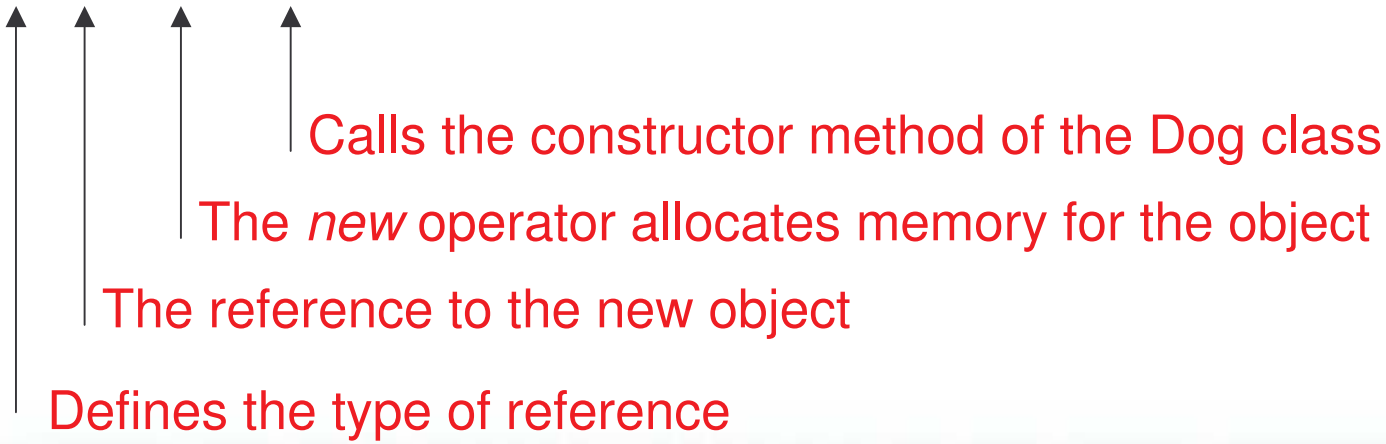
The Lifecycle of an Object

- Creating objects
- Using objects
- Cleaning up unused objects

Creating an object

- This statement initiates a reference to a new object and calls the constructor.

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



Constructors

- A special method defined in the class.
  - Initializes the state of an object.
  - Makes sure the new object is ready for use.
- Every class has a default constructor that takes no arguments.
- You can also provide your own constructors.
  - There can be many as long as each is differentiated by the number and type of arguments.
  - Constructors with arguments are called with statements like this:
    - Dog d = new Dog(name, size);
    - Dog d = new Dog(breed, name, size);

Using an object

The Dot Operator

- The dot operator gives you access to an object's state and behavior.
  - Make a new object

    Dog d = new Dog();
  - Call one of the object's methods

    d.bark();
  - Set one of the object's instance variables

    d.size = 40;

The Java Heap

- Each time an object is created in Java, it goes into an area of memory known as the Garbage-Collection heap.
  - All objects no matter when or how created go on the heap.
  - Upon object creation, Java allocates memory space on the heap according to the object's needs.

Cleaning up

- When an object is no longer in use, it becomes eligible for garbage collection.
- If you're running low on memory, the GC will run and throw out the unreachable objects.

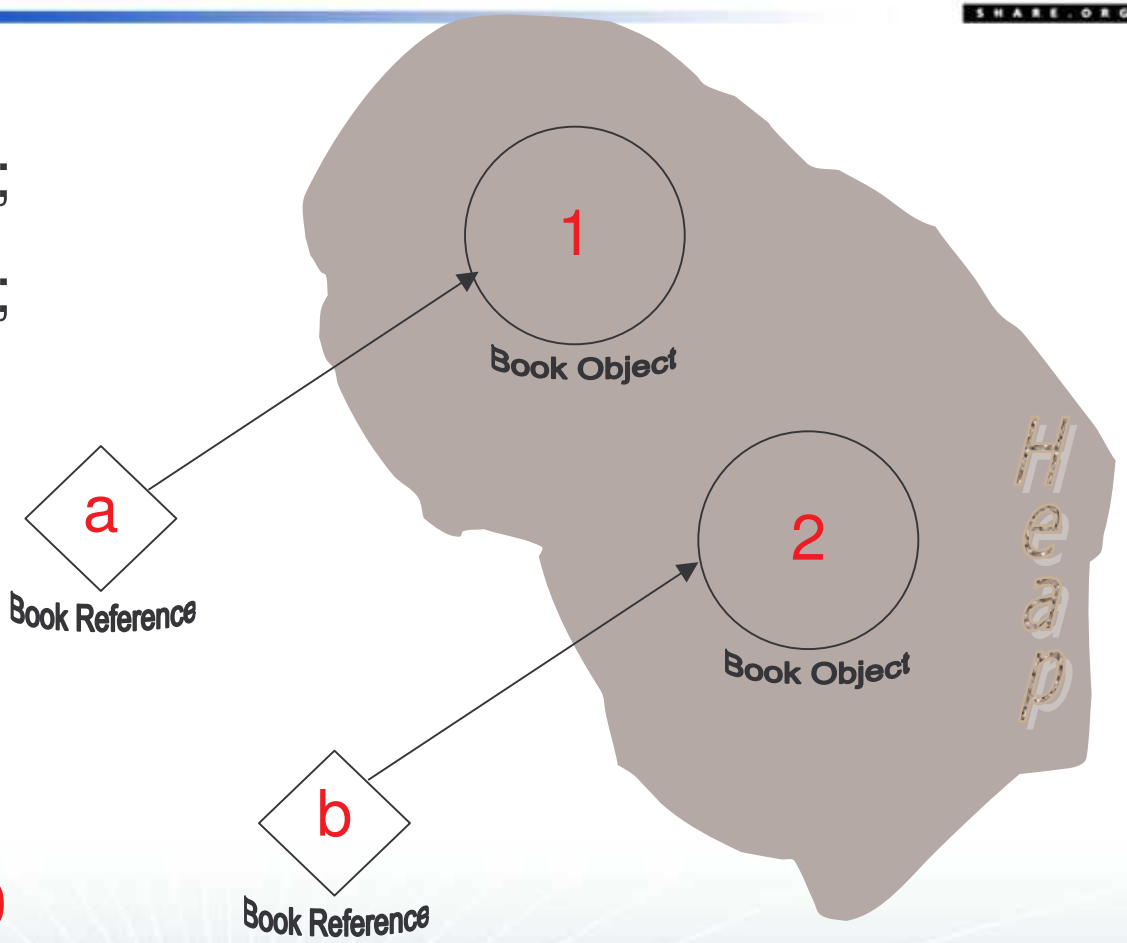
Counting References

- The Java Runtime keeps track of the references to an object.
- When the number of references drops to zero, the object without a reference is marked for collection.

Garbage Collection

Book a = new Book();

Book b = new Book();



Active References: 2

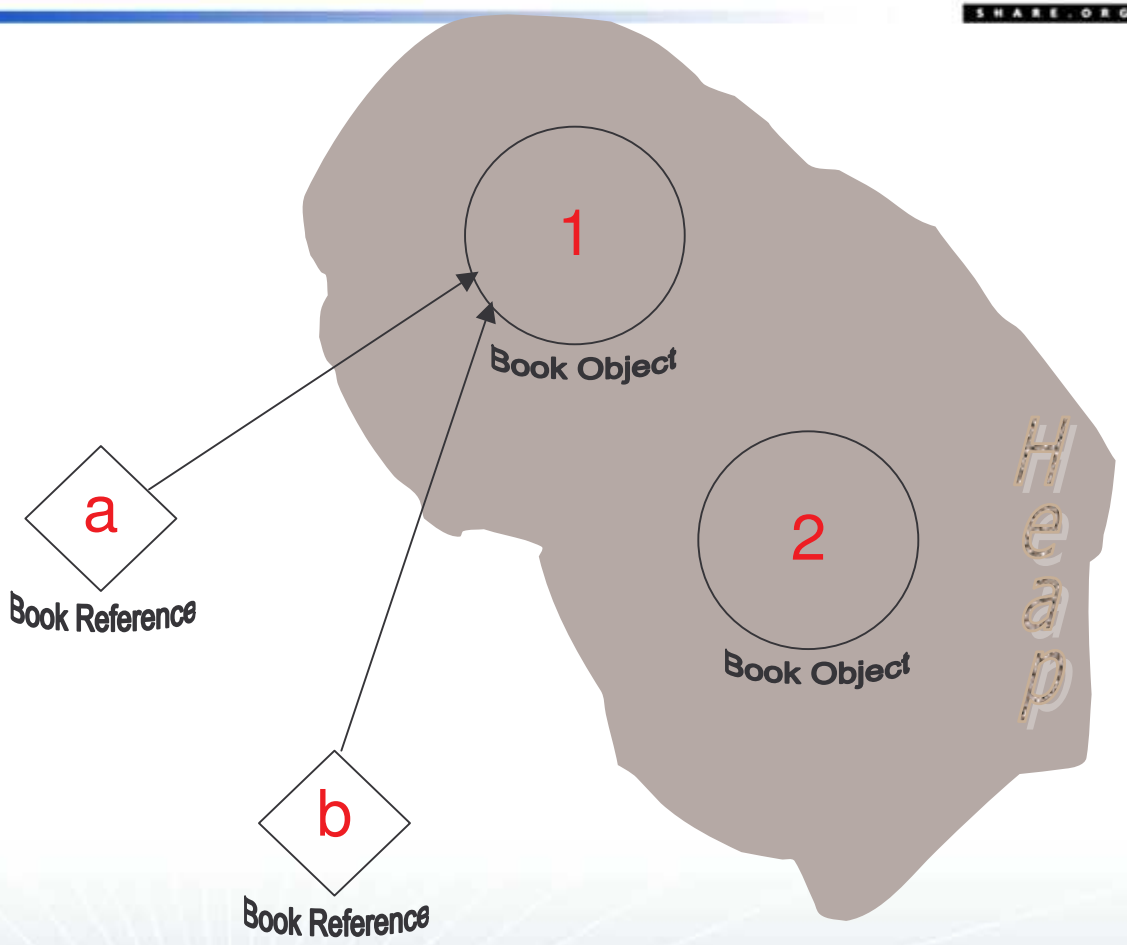
Reachable Objects: 2

Abandoned Objects: 0

Garbage Collection

b = a; *or*

b = null;



Active References: 2

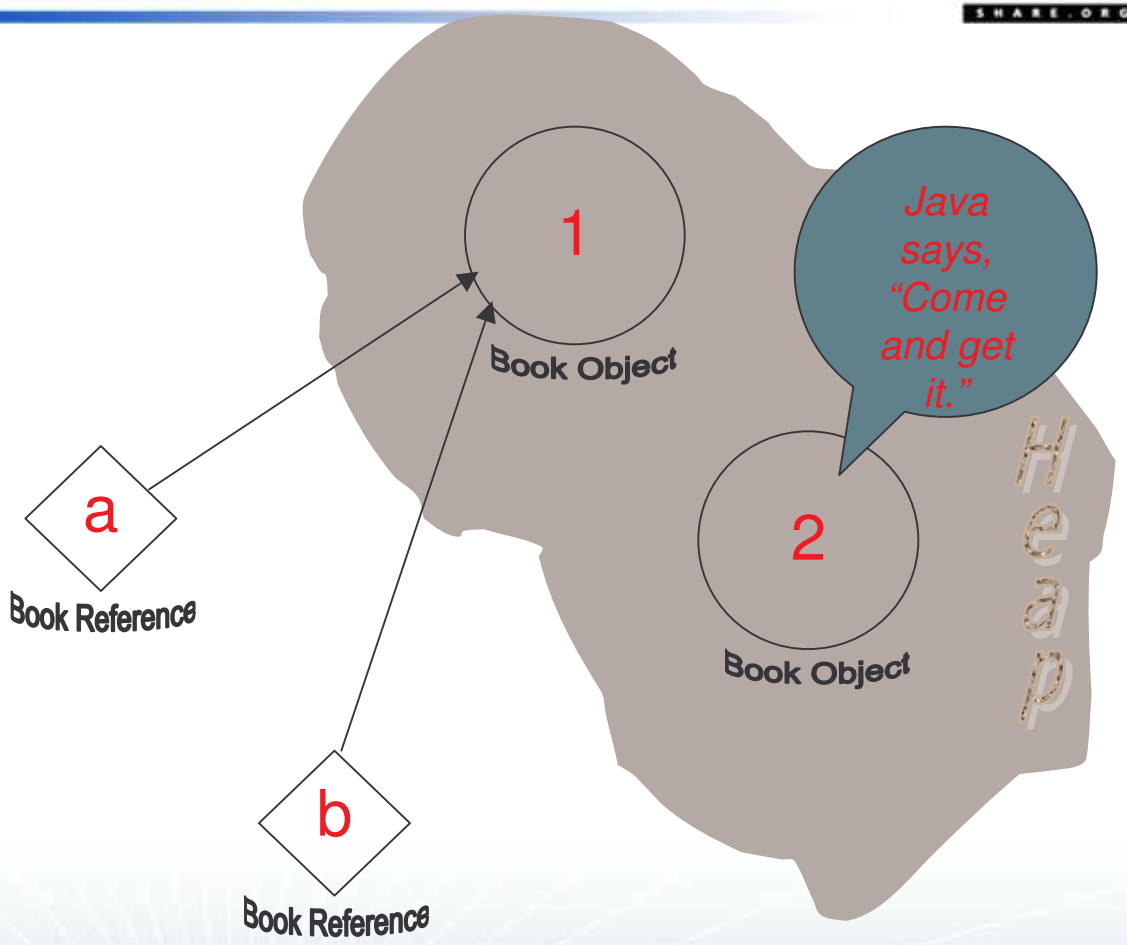
Reachable Objects: 1

Abandoned Objects: 1

Garbage Collection

b = a; *or*

b = null;



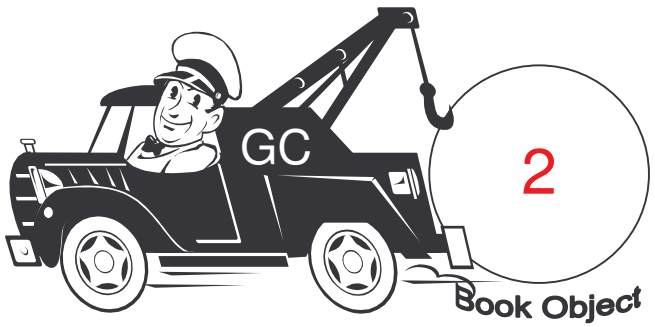
Active References: 2

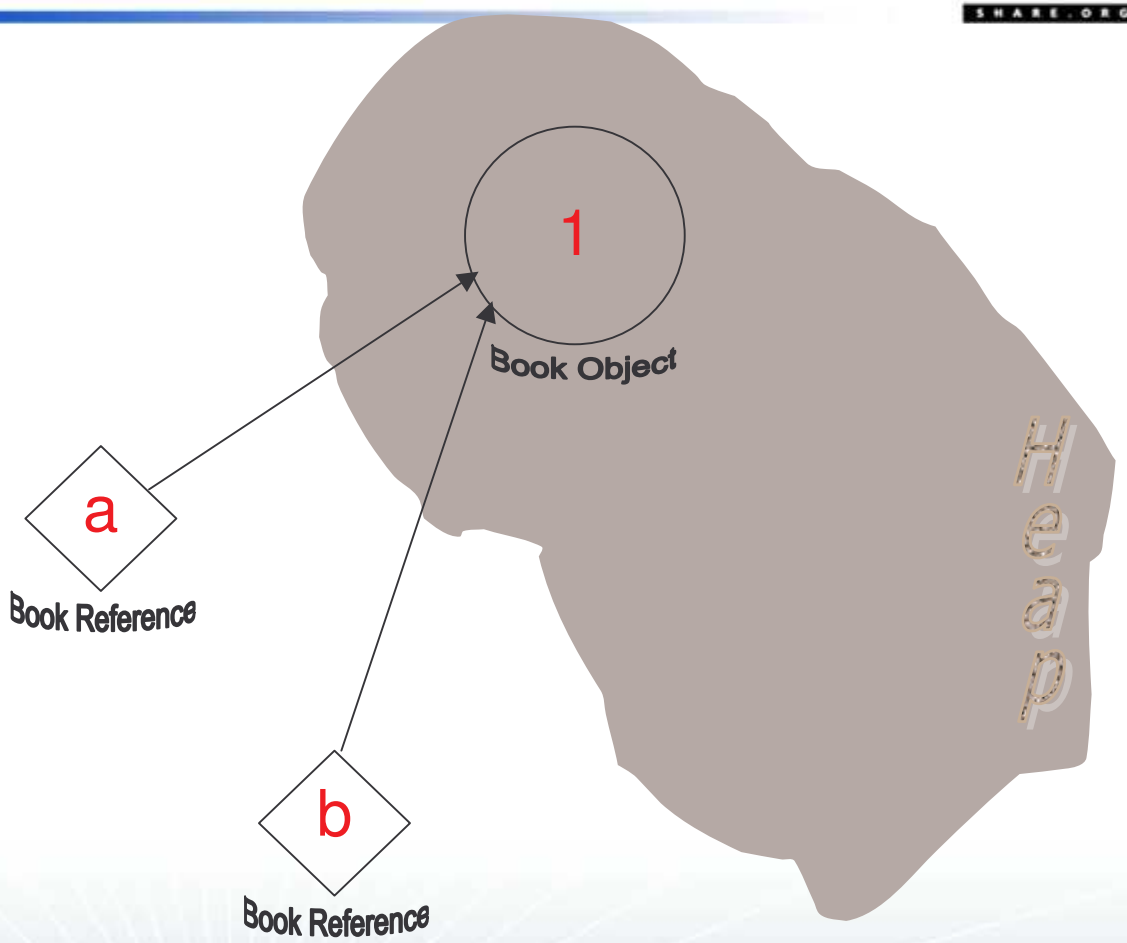
Reachable Objects: 1

Abandoned Objects: 1

Garbage Collection

Sometime later….





Revisiting the Objectives

- Compare/Contrast OO Programming to Procedural Programming
  - Add/change features without touching tested code.

Revisiting the Objectives

- Introduction to these Object Oriented concepts:
  - Classes
    - Look for nouns in specification.
    - The blueprint for an object.
  - Objects
    - The realization of a class.
  - Class Data
    - Things an object knows.
  - Methods
    - Things and object does.

Revisiting the Objectives

- Understand the lifecycle of an object
  - A constructor starts it.
  - The heap holds it.
  - The Garbage Collector clears it.